



Artificial Intelligence Programming

Intro to Machine Learning

Chris Brooks

Department of Computer Science

University of San Francisco



Introduction

- We've talked about learning previously in the context of specific algorithms.
- Purpose: discuss learning more generally.
- Give a flavor of other approaches to learning
- Talk more carefully about how to evaluate the performance of a learning algorithm.

Defining Learning

- So far, we've defined a learning agent as one that can improve its performance over time.
- We've seen two learning algorithms:
 - Decision tree
 - Bayesian Learning
- Let's define the problem a bit more precisely.

Defining Learning

- A program is said to learn from experiences E with respect to a set of tasks T and a performance measure P if its performance on T , as measured by P , improves with experience E .
- This means that, for a well-formulated learning problem, we need:
 - A set of tasks the agent must perform
 - A way to measure its performance
 - A way to quantify the experience the agent receives

Examples

- Speech recognition
 - Task: successfully recognize spoken words
 - Performance measure: fraction of words correctly recognized
 - Experience: A database of labeled, spoken words
- Learning to drive a car
 - Task: Drive on a public road using vision sensors
 - Performance: average distance driven without error
 - Experience: sequence of images and reactions from a human driver.
- Learning to play backgammon
 - Task: play backgammon
 - Performance measure: number of games won against humans of the appropriate caliber.
 - Experience: Playing games against itself.

Discussion

- Notice that not all performance measures are the same.
 - In some cases, we want to minimize all errors. In other cases, some sorts of errors can be more easily tolerated than others.
- Also, not all experience is the same.
 - Are examples labeled?
 - Does a learning agent immediately receive a reward after selecting an action?
 - How is experiential data represented? Symbolic? Continuous?
- Also: What is the final product?
 - Do we simply need an agent that performs correctly?
 - Or is it important that we understand *why* the agent performs correctly?

Types of learning problems

- One way to characterize learning problems is by the sorts of data and feedback our agent has access to.
 - batch vs incremental
 - supervised vs unsupervised
 - active vs passive
 - Online vs Offline

Batch vs Incremental

- We can think about problems or algorithms being batch or incremental.
- A *batch* learning algorithm is one in which all of the data is available at once to the agent.
 - Decision trees are a batch learning algorithm.
- An incremental learning algorithm is one that can continue to incorporate new data over time as it becomes available.
 - Naive Bayes can be used incrementally.
- In principle, batch learning is more effective, but it may not fit the characteristics of all problems.

Supervised vs Unsupervised

- A *supervised* learning algorithm/problem is one in which the learner has access to labeled training data.
 - Decision trees are an example of this.
- *Unsupervised* algorithms/problems are ones in which no labeled training data is available.
 - The recommender systems used by Amazon and Netflix are examples of this.
- Supervised learning is easier, but it assumes that you have access to labeled training data.

Active vs Passive

- In *active learning*, the learning agent is able to construct examples and find out their classification.
 - For example, if our spam classifier could create emails and ask a teacher whether it was spam or not.
- In *passive learning*, the learning agent must work with the examples that are presented.
 - Our decision tree problem was a passive learning problem.
- Active learning is more effective, as the agent can choose examples that lets it better “hone” its hypothesis, but may not fit with a particular problem.

Online vs Offline

- An *offline* learning algorithm is able to separate learning from execution.
 - Learning and performance are separate
 - Batch learning is easier, computational complexity is less of a factor.
- An *online* learning algorithm allows an agent to mix learning and execution.
 - Agent takes actions, receives feedback, and updates its performance component.
 - Incremental learning makes more sense, fast algorithms a requirement.
- We will worry about both training time (time needed to construct a hypothesis) and classification time (time needed to classify a new instance).

Other types of learning

- In this class, we'll focus primarily on inductive supervised learning
 - Well-understood, mature, many applications.
- There are other types of learning
 - Deductive learning
 - Unsupervised learning
 - Reinforcement learning

Induction

- Induction is the process of concluding general knowledge from specific facts.
 - On the last ten days that were sunny, we played tennis. Therefore, when it is sunny, we play tennis.
- Allows us to draw general conclusions from data.
- Most machine learning algorithms use induction.

Deductive Learning

- Deductive learning develops rules about specific situations from general principles.
 - “Knowledge-based” learning might be a better name
 - some induction may take place.
- For example, a deductive learning agent might cache the solution to a previous search problem so that it doesn’t need to re-solve the problem.
- It might even try to generalize some of the specifics of the solution to apply to other instances.
 - Case-based reasoning is another example of this style of learning.
 - Start with a number of “cases” or “recipes”, and try to fit specific situations to one of these.

Unsupervised Learning

- In *unsupervised learning*, there is no teacher who has presented the learner with labeled examples.
- Instead, all the learner has is data.
- Problem: find a hypothesis (or pattern) that explains the data.

Clustering

- One example of unsupervised learning is *clustering*
- Given a collection of data, group the data into k clusters, such that similar items are in the same cluster.
- Challenge: don't know the class definitions in advance.
- We will look at techniques for doing this on Tuesday.

Reinforcement Learning

- In some cases, an agent must learn through interaction with the environment.
- Agent selects and executes an action and receives a reward as a result.
- Learning problem: What is the best action to take in a given state?
- Issue - since learning is integrated with execution, we can't just explore every possibility.
- Approach (in a nutshell) - try different actions to see how they do.
- The more confidence we have in our estimate of action values, the more likely we are to take the best-looking action.

Q-learning

- We want to learn a *policy*
 - This is a function that maps states to actions.
- What we get from the environment are state: reward pairs.
- We'll use this to learn a $Q(s, a)$ function that estimates the reward for taking action a in state s .
- This is a form of *model-free* learning
 - We do no reasoning about how the world works - we just map states to rewards.
 - This means we can apply the same algorithm to a wide variety of environments.

Q-learning

- We keep a table that maps state-action pairs to Q values.
- Every time we are in state s and take an action a , we receive a reward r , and wind up in state s' .
- We update our table as follows:
 - $Q(s, a) \leftarrow Q(s, a) + \alpha(r + \gamma \max_{a'} Q(s', a') - Q(s, a))$
- In other words, we add in the reward for taking an action in this state, plus acting optimally from that point.
- α is the learning rate.

Q-learning

- Q-learning has a distinct difference from other learning algorithms we've seen:
- The agent can select actions and observe the rewards they get.
- This is called *active learning*
- Issue: the agent would also like to maximize performance
 - This means trying the action that currently looks best
 - But if the agent never tries “bad-looking” actions, it can't recover from mistakes.
- Intuition: Early on, Q is not very accurate, so we'll try non-optimal actions. Later on, as Q becomes better, we'll select optimal actions.

Boltzmann exploration

- One way to do this is using Boltzmann exploration.
- We take an action with probability:
- $$P(a|s) = \frac{k^{Q(s,a)}}{\sum_j k^{Q(s,a_j)}}$$
- Where k is a temperature parameter.
- This is the same formula we used in simulated annealing.
- We'll return to Q-learning after we discuss MDPs
 - They're closely related.

Return to supervised learning

- Let's step back and think about supervised learning.
- Given some labeled data, find a hypothesis that best explains this data.
- This can be done using symbolic or numeric data.

A symbolic example

- Consider (once again) our playTennis example.
- Suppose we have the following experience:
 - Sunny, overcast, high, weak : yes
 - Sunny, overcast, low, strong: yes
 - Rainy, overcast, normal, weak: no
- We need to select a hypothesis that explains all of these examples
 - H1: sunny : yes
 - H2: Sunny and Overcast: yes
 - H3: \neg Rainy, overcast, normal, weak : yes
- Which do we pick?

Representing a hypothesis

- Before we can answer, we need to decide how our hypothesis will be represented.
 - All possible prob. logic expressions?
 - Only conjunctions?
 - Negation?
- Simpler hypotheses can be learned more quickly
- May not fit data as well.

Find-S

- Suppose we agree that hypotheses consist of a single attribute value or “don’t care” for each attribute.
 - Sunny, sunny and overcast are possible
 - sunny or rainy is not.
- This is called a *representational bias*.
- Stronger representational biases let us learn more quickly.
- Find the most specific hypothesis that explains our data.

Hypothesis spaces

- We can arrange all potential hypotheses from specific-to-general in a lattice.
- Our learning problem is now to search this space of hypotheses to find the best hypothesis that is consistent with our data.
- the way in which those hypotheses are considered is called the *learning bias*
- Every algorithm has a representational bias and a learning bias.
 - Understanding them can help you know how your learning algorithm will generalize.

A numeric example

- Suppose we have the following data points:
 - (160, 126), (180, 103), (200, 82), (220, 75), (240, 82), (260, 40), (280, 20)
- We would like to use this data to construct a function that allowed us to predict an $f(x)$ for other x .
- There are infinitely many functions that fit this data - how do we choose one?
- Representational bias: Restrict ourselves to straight lines
- Inductive bias: choose the line that minimizes sum of squared errors.
- This is linear regression - most statistics packages can compute it.

Nonlinear Regression

- Linear regression is nice because it's easy to compute.
- Problem: many functions we might want to approximate are not linear.
- *Nonlinear regression* is a much more complicated problem
 - How do we choose a representational bias?
Polynomial? Trigonometric?
- Neural networks are actually nonlinear function approximators.
 - We'll return to them the last week of class and see how they automatically induce a nonlinear function.

Approximation vs. Classification

- Regression is an example of *function approximation*.
 - Find a function that approximates given data and performs well on unseen data.
- A particular kind of function to approximate is a *classification function*
 - Maps from inputs into one or more classes.
 - Task: find a hypothesis that best splits the data into classes.
- This is the task that decision trees and Bayesian learners solve.

Measuring Performance

- How do we evaluate the performance of a classifying learning algorithm?
- Two traditional measures are precision and accuracy.
- Precision is the fraction of examples classified as belonging to class x that are really of that class.
 - How well does our hypothesis avoid *false positives*?
- Recall (or accuracy) is the fraction of true members of class x that are actually captured by our hypothesis.
 - How well does our hypothesis capture *false negatives*?

Precision vs recall

- Often, there is a tradeoff of precision vs recall.
 - In our playTennis example, what if we say we always play tennis?
 - this will have a high accuracy, but a low precision.
 - What if we say we'll never play tennis?
 - High precision, low accuracy.
- Try to make a compromise that best suits your application.
- What is a case where a false positive would be worse than a false negative?
- What is a case where a false negative would be better than a false positive?

Evaluation

- Typically, in evaluating the performance of a learning algorithm, we'll be interested in the following sorts of questions:
 - Does performance improve as the number of training examples increases?
 - How do precision and recall trade off as the number of training examples changes?
 - How does performance change as the problem gets easier/harder?
- So what does 'performance' mean?

Evaluation

- Recall that supervised algorithms start with a set of labeled data.
- Divide this data into two subsets:
 - Training set: used to train the classifier.
 - Test set: used to evaluate the classifier's performance.
 - These sets are disjoint.
- Procedure:
 - Train the algorithm with the classifier.
 - Run each element of the test set through the classifier. Count the number of incorrectly classified examples.
 - If the classification is binary, you can also measure precision and recall.

Evaluation

- How do we know we have a representative training and test set?
- Try it multiple times.
- N-fold cross-validation:
 - Do this N times:
 - Select $1/N$ documents at random as the test set.
 - Remainder is the training set.
 - Test as usual.
 - Average results.

Ensemble learning

- Often, classifiers reach a point where improved performance on the training set leads to reduced performance on the test set.
 - This is called *overfitting*
- Representational bias can also lead to upper limits in performance.
- One way to deal with this is through *ensemble learning*.
 - Intuition: Independently train several classifiers on the same data (different training subsets) and let them vote.
 - This is basically what the Bayes optimal classifier does.

Boosting

- Boosting is a widely-used method for ensemble learning.
- Pick your favorite classifier.
- Idea:
 - For $i = 1$ to M :
 - Train the i th classifier on the training set.
 - For each misclassified example, increase its “weight”
 - for each correctly classified example, decrease its “weight”.

Boosting

- To classify :
 - Present each test example to each classifier.
 - Each classifier gets a vote, weighted by its precision.
- Very straightforward - can produce substantial performance improvement.
 - Combining stupid classifiers can be more effective than building one smart classifier.

Summary

- Learning is the process of improving performance on a set of tasks through experience.
 - This can take many different forms.
- Supervised learning is a (particularly interesting) subset of learning.
- In evaluating learning, we will be interested in precision, recall, and performance as the training set size changes.
- We can also combine poor-performing classifiers to get better results.