Overview of Inductive Learning
CIML Chapter 1 and 2

Daniel Lowd

Objectives

- Learn to think about **formulating tasks as supervised classification problems**
- **Learn definitions:** inductive learning, loss, error, data distribution, data sample, inductive bias, features, overfitting, hyperparameter, Bayes optimal classifier
- Understand **limits of machine learning**
Inductive Learning

**Informally:** Learning is about using experience to improve performance.

**Formally:** Given a loss function \( \ell \) and a sample \( D \) from some unknown distribution \( \mathcal{D} \), you must compute a function \( f \) that has low expected error \( \epsilon \) over \( \mathcal{D} \) with respect to \( \ell \).

\[
\epsilon = \mathbb{E}_{(x,y) \sim \mathcal{D}} \left[ \ell(y, f(x)) \right]
\]

Examples of loss functions:
Squared loss, absolute loss, 0/1 loss... what else?
What If We Knew $\mathcal{D}$?

\[ \epsilon = E_{(x,y)} \mathcal{D} [\ell(y, f(x))] \]

Suppose knew the data distribution $\mathcal{D}$ exactly. How could choose $f$ to minimize the error?

\[ f^{(BO)}(\hat{x}) = \arg \max_{\hat{y} \in \mathcal{Y}} \mathcal{D}(\hat{x}, \hat{y}) \]

The **Bayes optimal classifier** predicts the most probable $y$ for every $x$.

**Theorem 1** (Bayes Optimal Classifier). The Bayes Optimal Classifier $f^{(BO)}$ achieves minimal zero/one error of any deterministic classifier.

Inductive Bias

Since we don’t know $\mathcal{D}$, we must make do with the **training error** (or “empirical risk”) $\hat{\epsilon}$ over training data $\mathcal{D}$:

\[ \hat{\epsilon} = \frac{1}{N} \sum_{i=1}^{n} \ell(y^{(i)}, f(x^{(i)})) \]

Many different $f$ may have the same training error. **Inductive bias** is how we choose among them.

Without bias we cannot learn!
Tuning and Testing

• How do we know if $f$ is good?
  – Can’t use data generating distribution directly
  – Low training error could be misleading – easy to memorize data and overestimate performance.

• Average loss on previously unseen data is a much better indicator of future performance.
  – Use held out validation data to choose algorithms and hyperparameters (algorithm settings) that are likely to generalize well.
  – Use separate test data for final evaluation.

Important Concepts

• Data: labeled instances, e.g. emails marked spam/ham
  – Training set
  – Held out set
  – Test set

• Features: attribute-value pairs which characterize each $x$

• Experimentation cycle
  – Learn parameters (e.g. model probabilities) on training set
  – (Tune hyperparameters on held-out set)
  – Compute accuracy of test set
  – Very important: never “peek” at the test set!

• Evaluation
  – Accuracy: fraction of instances predicted correctly

• Overfitting and generalization
  – Want a classifier which does well on test data
  – Overfitting: fitting the training data very closely, but not generalizing well
The Big Picture

Problem Domain
(email spam)

Data Representation
("bag of words")

Objective Function
(error rate on training emails)

Evaluation
(error rate on future emails)

Learning Algorithm
(C4.5 algorithm)

Predictive Model
(decision tree)

Useful Predictions
(automated filter)

Domain Insight
(types of spam)

This class mostly focuses on the red pieces, but the blue pieces are just as important!
The general approach is as follows:

1. Split your data into 70% training data, 10% development data and 20% test data.
2. For each possible setting of your hyperparameters:
   (a) Train a model using that setting of hyperparameters on the training data.
   (b) Compute this model’s error rate on the development data.
3. From the above collection of models, choose the one that achieved the lowest error rate on development data.
4. Evaluate that model on the test data to estimate future test performance.

In step 3, you could either choose the model (trained on the 70% training data) that did the best on the development data. Or you could choose the hyperparameter settings that did best and retrain the model on the 80% union of training and development data. Is either of these options obviously better or worse?

---

Real World Applications of Machine Learning

Figure 2.4 shows a typical sequence of decisions that must be made to deploy a machine learning approach in the real world. In the left column, you can see the generic decision being made. In the right column, an example of this decision for the particular case of advertising placement on a search engine we’ve built.

In this sequence, (1) we have some real world goal like increasing revenue for our search engine, and decide to try to increase revenue by (2) displaying better ads. We convert this task into a machine learning problem by (3) deciding to train a classifier to predict whether a user will click on an ad or not. In order to apply machine learning, we must collect some training data; in this case, (4) we collect data by logging user interactions with the current system. We must choose what to log; (5) we choose to log the ad being displayed, the query the user entered into our search engine, and binary value showing if they clicked or not.

---

ML in a Nutshell

- Tens of thousands of machine learning algorithms
- Hundreds new every year
- Every machine learning algorithm has three components:
  - **Model Representation**
  - **Evaluation**
  - **Optimization**
Model Representation

- Decision trees
- Instances
- Linear function (hyperplane)
- Neural networks
- Support vector machines
- Model ensembles
- (Sets of rules / Logic programs)
- (Graphical models (Bayes/Markov nets))
- Etc.

Evaluation

- Accuracy
- Precision and recall
- Squared error
- Likelihood
- Posterior probability
- Cost / Utility
- Margin
- Entropy
- K-L divergence
- Etc.
Optimization

• Combinatorial optimization
  – E.g.: Greedy search
• Convex optimization
  – E.g.: Gradient descent
• Constrained optimization
  – E.g.: Linear programming

Exercise

In pairs or small groups, come up with a supervised classification problem of your own.
• What’s the input (x) and output (y)?
• How would you represent the data (features)?
• How would you get training data?
• What loss function would you use?
• How well do you think the problem could be solved, given great data and a great algorithm?
• What good and bad things could happen if you did solve this problem?