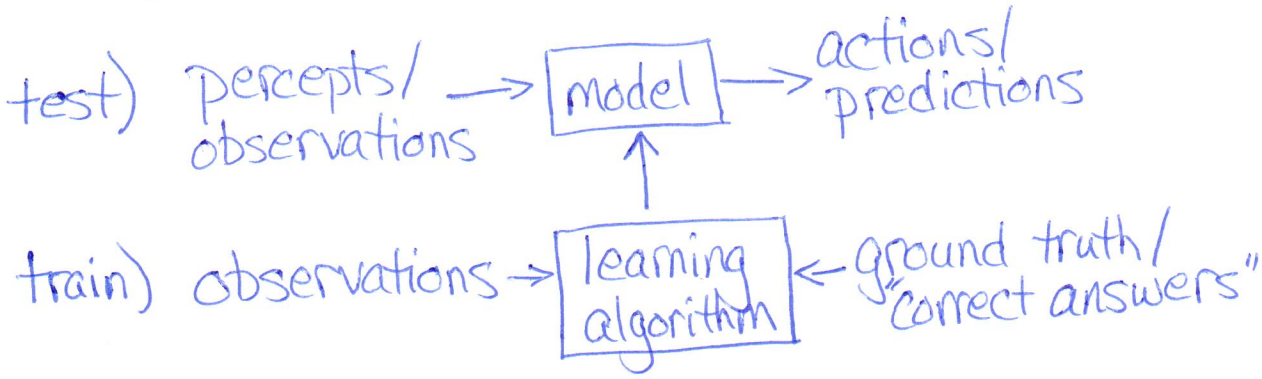
Machine Learning Overview

Learning agents directly improve their own performance at some specific task by exposure to data (i.e. with experience).

Traditionally in AI, we design the model that transforms observations into predictions (or percepts into actions).

percepts/observations → [model] → actions/predictions

In ML, we create a general learning algorithm and it creates the model, improving it through experience (data).



Machine Learning in a Nutshell:

- a model is a construct that turns observations into predictions
- we input values we know (evidence) to get predictions of values we don't know (hidden variables)
- models can be manually created
- we train a learning algorithm with examples to obtain a model, then query the model to make predictions
- inductive rather than deductive learning (generalizing its own "rule" $f(x) \rightarrow y$ from data, possibly incorrectly, rather than using explicit rules or probabilities)

Categorizing Learning Algorithms (and Problems)

Experiences are commonly represented as tuples (X, y), with X a 1-D vector of features and y the desired model output for observation X.

Features are values derived from raw observations.
- example: if raw observations are stock price over time, features could be moving average, standard deviation...

Some Learner Types:
- unsupervised: given observations only, no labels/"answers", learns $P(X)$, can be sampled
- supervised generative: given labelled observations (X, y), learns $P(X, y)$, having FJD it can be sampled to generate new data
- supervised discriminative: given labelled observations (X, y), learns $P(y|X)$, cannot be sampled, like drawing "lines in space" to separate answer regions
- reinforcement learning: uses rewards to learn the optimal action to take for each possible state, produces a policy $\pi: f(s) \rightarrow a$

Some Problem Types:
- classification: discrete, limited number of outputs (e.g. is this photo a cat, dog, or horse?)
- regression: continuous output (e.g. what price will IBM stock be tomorrow?)
- Note! Output determines this. Input features could be discrete, continuous, or both for either problem type!

Consistency:
Hypothesis (prediction function) $h(X) \rightarrow y_{pred}$ is consistent if it fits the real function $f(X) \rightarrow y$, within some allowed error, for all examples. We choose the simplest consistent $h(X)$.

Example Discriminative Supervised Classification Learner

K-Nearest Neighbors (KNN)

- called an instance learner because it memorizes the training examples (all of them)
- also called a nonparametric learner because it does not learn a fixed-size set of parameters (instance is a subset of nonparametric)
- training KNN is trivial but memory-intensive: store all (X, y) training examples
- querying KNN is computationally expensive: given test X vector, computes similarity (usually just Euclidean distance) between test X and each train X (pairwise) to select K nearest neighbors of test X
- result of query is mode(y) of K nearest neighbors (i.e. majority vote)
- KNN is also used for regression, output is mean(y) of K nearest neighbors
- when K=1, querying a training X vector returns the exact answer y from training set (nearest neighbor is self)
- when K=N (size of training set), querying any X vector returns global mode (or mean) of training y values, because the entire training set are always the K nearest neighbors!

Parametric Learner:

opposite of nonparametric - learns a set of parameters that does not vary in size as the training set varies in size (i.e. is fixed-size set of parameters)

Common example is linear regression (see Local Search)

"Fit" and Generalization

We train a learner to fit itself to the training data by altering its parameters, or ourselves altering its hyperparameters (like the K in KNN or the polynomial degree d of a polynomial learner).

Underfit:

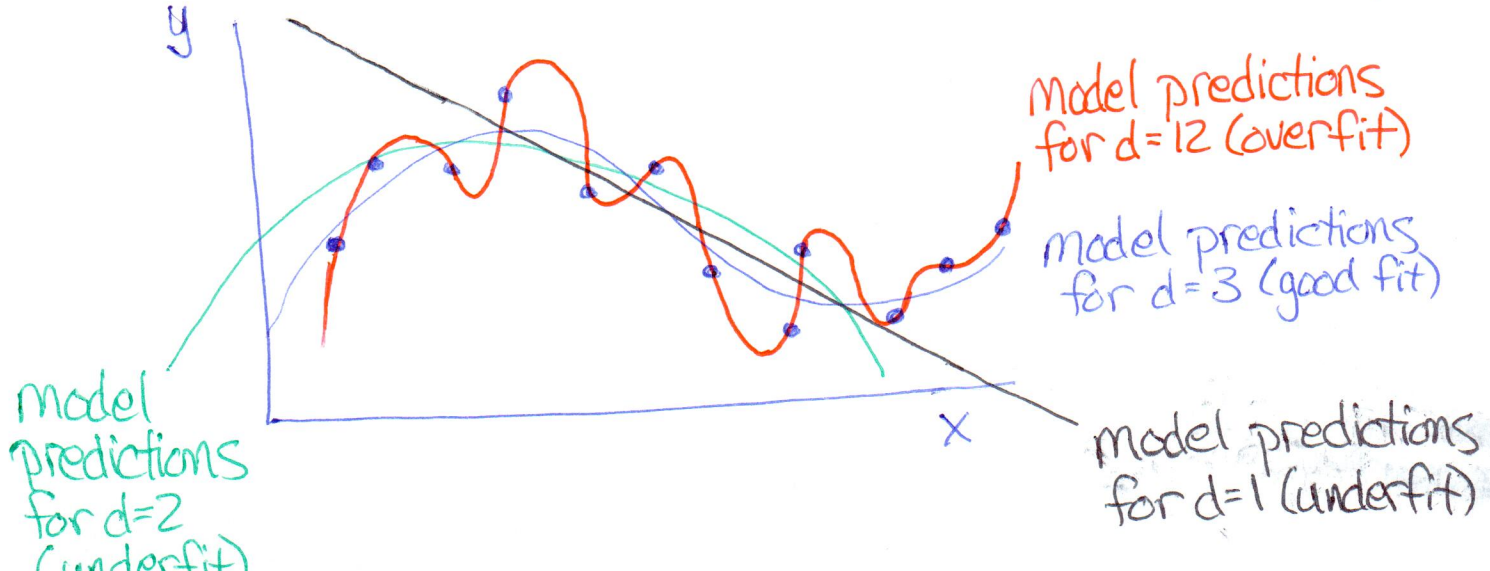
model performs poorly on training and test data
- lacks "power" (capacity, flexibility) to find or fit a pattern in the data

Overfit:

model performs well on training data ("in-sample" error) but poorly on test data ("out-of-sample" error)
- has learned training data too well
- has fit to the pattern + random noise of training set
- does not generalize to unseen data

Example with Polynomial Regression:

Polynomial regression learns a polynomial of degree d.
- $y = m_0 + m_1 x + m_2 x^2 + m_3 x^3 + \ldots + m_d x^d$
- learns m parameters to accurately fit $f(x) \rightarrow y$



Measuring Performance

What does it mean to "generalize well"?

In ML, usually judge performance by an error or loss function.
- i.e. how wrong are the predictions?
- assuming a valid loss function, minimizing its value (for out-of-sample testing) should produce the "best" learner

For regression, root mean squared error or Pearson's correlation coefficient are common

For simple binary classification, 0-1 loss is often enough
- meaning loss += 1 for each wrong prediction

Why not just use error rate $\frac{\#\text{errors}}{\#\text{predictions}}$?
- not all errors are equally important
- loss function allows us to reflect that, guiding learner to especially avoid "bad" errors
- can penalize a trivial error a little (+1 loss) and a terrible error a lot (+100 loss)

Cross-Validation

If we give our learner all the data we have for training, how can we tell if it will generalize to new data?

Hold out some data to test the learner.
- split data into a training set and a testing set
- 60-80% of data for training is common
- never allow the learner to update (learn) from test data

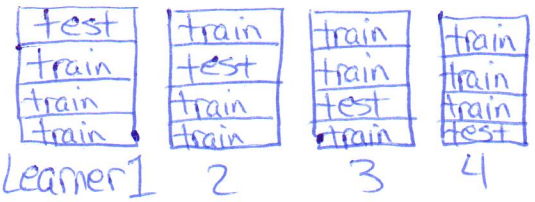
Learner re-queried on training data after training:
- unacceptable loss? underfit
- acceptable loss? proceed

Learner queried on test data after training:
- unacceptable loss (but train error was good)? overfit
- acceptable loss? good fit

What if we just got an unlucky partition of the data?
- we can do better than a single train/test split!

N-fold Cross Validation:

- split into N sets
- train & test N learners
- for each learner, train on all but one set, test that set
- average results



4-fold CV

Leave One Out Cross Validation:

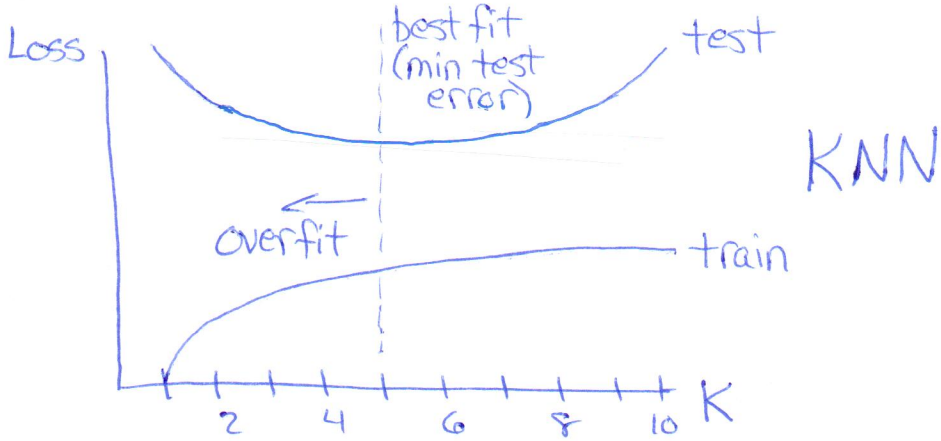
- train & test N learners (N = # of data points)
- each learner trains all but one data point
- tests the one
- average results
- for small data sets

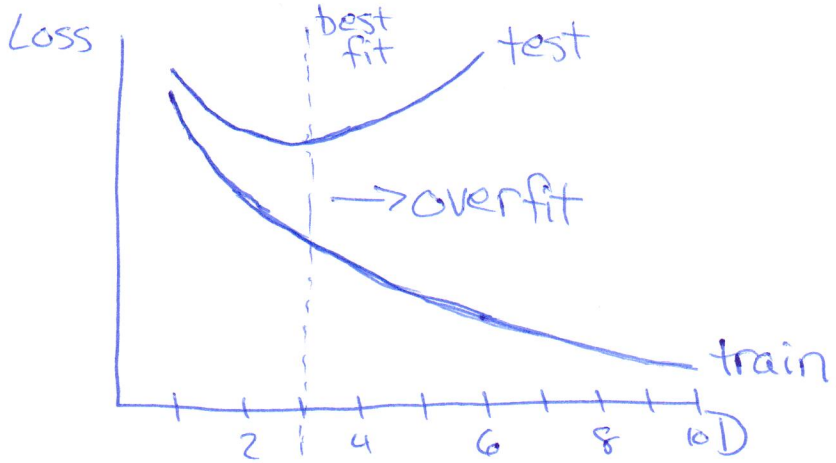
Exact Detection of Overfitting

Train and cross-validate many times while varying model hyperparameters (max depth in decision tree, K in KNN, d in polynomial regression, shape of neural network, ...)

Plot average train and test loss against hyperparameter value you are varying.

Overfitting occurs when the training (in-sample) error is decreasing while the test (out-of-sample) error is increasing.





Polynomial Regression

Underfitting is usually obvious (the model doesn't work at all).

Overfitting is a huge problem, because the model appears to work well, but won't in "real use"!