Machine Learning - part 2

Crowdsourcing and Human Computation
Lecture 13
Instructor: Chris Callison-Burch
TA: Ellie Pavlick

Website: crowdsourcing-class.org

Classifier components

- Input: a **vector** of discrete and/or continuous **feature values**
- Output: a single discrete value, the **class label**.
- For example, a spam filter classifies email messages into "**spam**" or "not **spam**,"
- Its input may be a Boolean **vector x = $(x_1,\ldots,x_j,\ldots,x_d)$**
- $x_j = 1$ if the jth word in the dictionary appears in the email and $x_j = 0$ otherwise

Learning a classifier

- A **learner** takes a training set of examples as input
- Its input are pairs of feature vectors and class labels ($\mathbf{x_i}$, y_i),
- where $\mathbf{x_i} = (x_{i,1}, \ldots, x_{i,d})$ is an observed input and y_i is the corresponding output
- The learner **outputs a classifier**

Learning =
Representation +
Evaluation +
Optimization

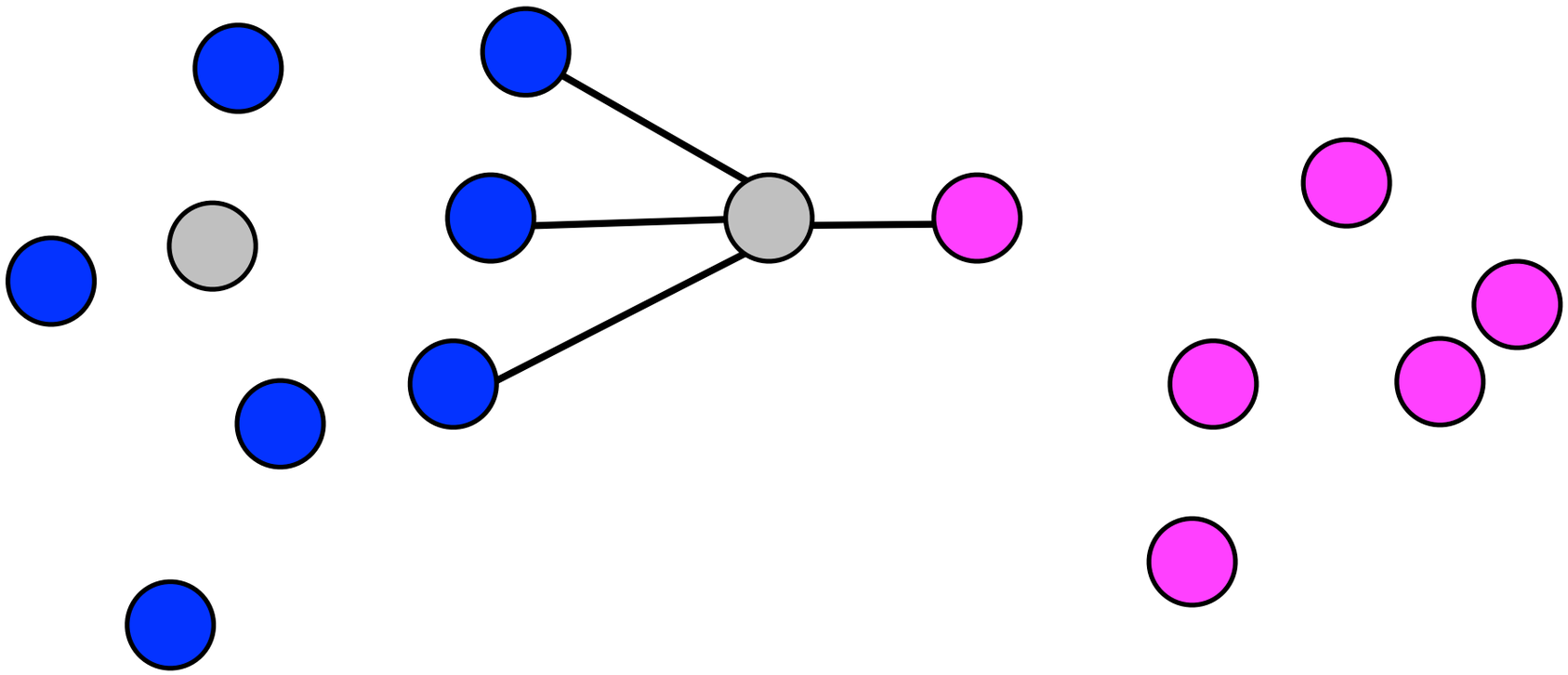
Representation

- Representation has two different meanings in machine learning:
  - How the *hypothesis space* is represented, which is related to the choice of classifier
  - How the *input* to the learner is represented, or what *features* we use to represent the data

Representations

- **Instances:** K-nearest neighbor, Support vector machines
- **Hyperplanes:** Naive Bayes, Logistic regression
- **Decision trees**
- **Sets of rules:** Propositional rules, Logic programs
- **Neural networks**
- **Graphical models:** Bayesian networks, Conditional random fields

k-nearest neighbors



Evaluation

- An evaluation function (also called *objective function* or *scoring function*) is needed to distinguish good classifiers from bad ones
- Used internally by the algorithm for setting its parameters through optimization

Objective functions

- Accuracy/Error rate
- Precision and recall
- Squared error
- Likelihood
- Posterior probability
- Information gain
- K-L divergence
- Cost/Utility
- Margin

Precision and Recall

	Actual Class	
Predicted class	True positive: Correct result	False positive: Unexpected result
	False negative: Missing result	True negative: Correct absence of result

$$\text{Precision} = \frac{tp}{tp + fp} \qquad \text{Recall} = \frac{tp}{tp + fn}$$

Precision and Recall

	Actual Class	
Predicted class	True positive: Correct result	False positive: Unexpected result
	False negative: Missing result	True negative: Correct absence of result

$$\text{Precision} = \frac{tp}{tp + fp}$$

Precision and Recall

	Actual Class	
Predicted class	True positive: Correct result	False positive: Unexpected result
	False negative: Missing result	True negative: Correct absence of result

$$\text{Recall} = \frac{tp}{tp + fn}$$

Accuracy

	Actual Class	
Predicted class	True positive: Correct result	False positive: Unexpected result
	False negative: Missing result	True negative: Correct absence of result

$$\text{Accuracy} = \frac{tp + tn}{tp + tn + fp + fn}$$

Optimization

- We need a method to search for the best instance of a classifier among its different possible instantiations

Optimization

- **Combinatorial optimization:** Greedy search
  Beam search Branch-and-bound
- **Unconstrained continuous optimization**: Gradient descent, Conjugate gradient, Quasi-Newton methods
- **Constrained continuous optimization:** Linear programming, Quadratic programming

Experimental design in machine learning

- Typically we have a fixed set of labeled data that we run experiments on
- In our experiments we typically split the data into a training set, and a disjoint test set
- Why?

Experimental design in machine learning

- Contamination of your classifier by test data can occur if you use test data to tune parameters, so might want to split into training / test / dev
- Splitting the data reduces the amount of available data for training
- Mitigated through cross-validation: randomly dividing your training data into 10 pieces, train on 9 test on 1, average results

It is generalization that counts

- The fundamental goal of machine learning is to generalize beyond the examples in the training set
- No matter how much data we have, at test time we are unlikely to see exactly the same items

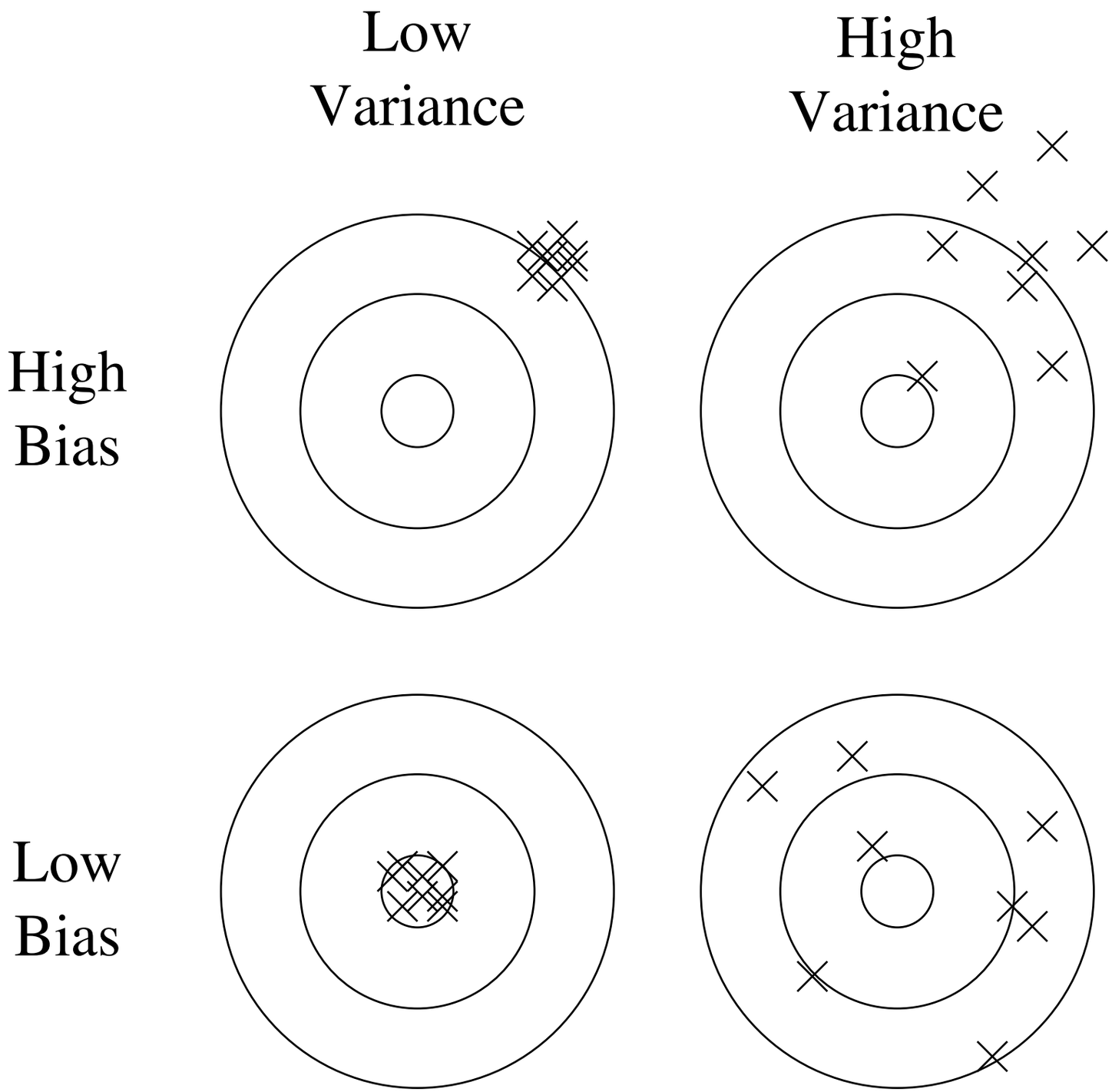
The problem of overfitting

- Sometimes our classifier *overfits* the data
- It encodes random quirks of the data instead of learning good generalizations
- Symptom: your learner creates a classifier that is 100% accurate on the training data but only 50% accurate on test data

Sources of generalization error

- Errors can be generalized into bias, variance, and noise
- Bias is a learner's tendency to consistently learn the same wrong thing
- Variance is the tendency to learn random things irrespective of the real signal
- Noise is caused by incorrectly labeled items in the training set

Bias and Variance contribute to errors



Ways to combat overfitting

- Cross-validation can help to combat overfitting: e.g. use it to choose the best size of decision tree to learn
- A regularization term can be added to the evaluation function: penalize classifiers with more structure, thereby favoring smaller ones with less room to overfit.

The curse of dimensionality

- The items in our training data are represented as d-dimensional vectors
- A spam filter might represent an email message with a vector $\mathbf{x} = (x_1,\ldots,x_j,\ldots,x_d)$ representing whether each word in the dictionary is in the email
- *Cialis* is a dimension, *homework* is a dimension

The curse of dimensionality

- Generalizing correctly becomes harder as the number of features grows
- A fixed-size training set covers a dwindling fraction of the possible inputs
- With 100 binary features, and a training set of *one trillion* examples, the training still only covers 10^{-18} of the input space

The blessing of non-uniformity

- Examples are not usually spread uniformly throughout the instance space
- K-nearest neighbor works for handwritten digit recognition
  - Images of digits have have high dimensionality (one dimension per pixel)
  - But actual the space of digit images is much smaller than all possible images

The blessing of non-uniformity

- Some learners implicitly take advantage of this lower effective dimension
- Some algorithms explicitly reducing the dimensionality

Data Alone is Not Enough

- Since generalization is our goal, data alone is not enough, no matter how much of it we have
- We can include our own knowledge about the function we want to learn
  - Similar examples have similar classes
  - The function we learn should be smooth
  - It should have limited complexity, etc.

Using knowledge to pick a representation

- If we know what makes examples similar, instance-based methods may be a good choice
- If we know about probabilistic dependencies, graphical models are a good fit
- If know what preconditions are required by each class, “IF . . . THEN . . .” rules may be the the best option

Feature Engineering is the key to success

- Why do some machine learning projects succeed and others fail?
- The most important factor is often the features used.
- If you have many independent features that each correlate well with the class, learning is easy
- If the class is a very complex function of the features, you may not be able to learn it

Raw data -> Features

- Raw data is not usually in a form that is best to learning, but you can derive useful features from it
- This is where most of the effort in a machine learning project goes
- It is often also one of the most interesting parts
- It is where you get to use your intuition and creativity
- It is the secret “black art” that is as important as the technical stuff

Time spend on a machine learning project

- Gathering data, integrating it, cleaning it and pre-processing it
- Doing trial and error on feature design
- Iteratively: run the learner, analyze the results, modify the data or the learner
- Can be surprising by how little time is spent actually doing machine learning

Feature engineering is the hard part

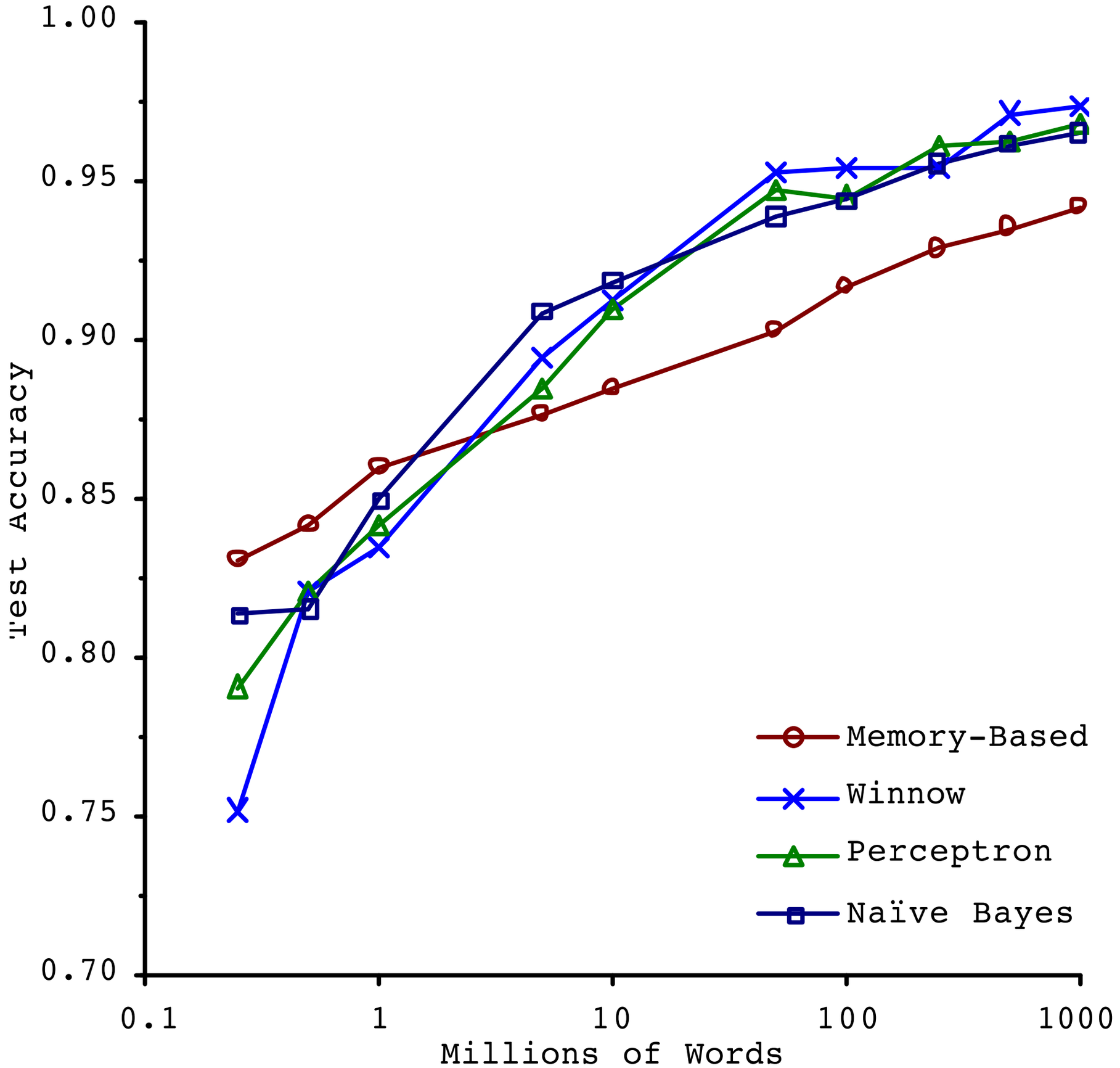
- Feature engineering is more difficult because it's domain-specific, while learners can be largely general-purpose
- Running a learner with a very huge number of features to find out which ones are useful may be too time-consuming, or cause overfitting
- Ultimately there is no replacement for the smarts you put into feature engineering

More data versus better algorithms

- What if you have constructed the best set of features you can, but the classifiers you're getting are still not accurate enough?
- There are two main choices on how to improve the algorithm:
  - design a better learning algorithm
  - or gather more data

More data versus better algorithms

- Machine learning researchers are mainly concerned with better algorithsm
- Pragmatically the quickest path to success is often to just get more data
- As a rule of thumb, a dumb algorithm with lots and lots of data beats a clever one with modest amounts of it.
- Machine learning is all about letting data do the heavy lifting


1.00
0.95
0.90
0.85
0.80
0.75
0.70
Test Accuracy
0.1
1
10
100
1000
Millions of Words
Memory-Based
Winnow
Perceptron
Naïve Bayes

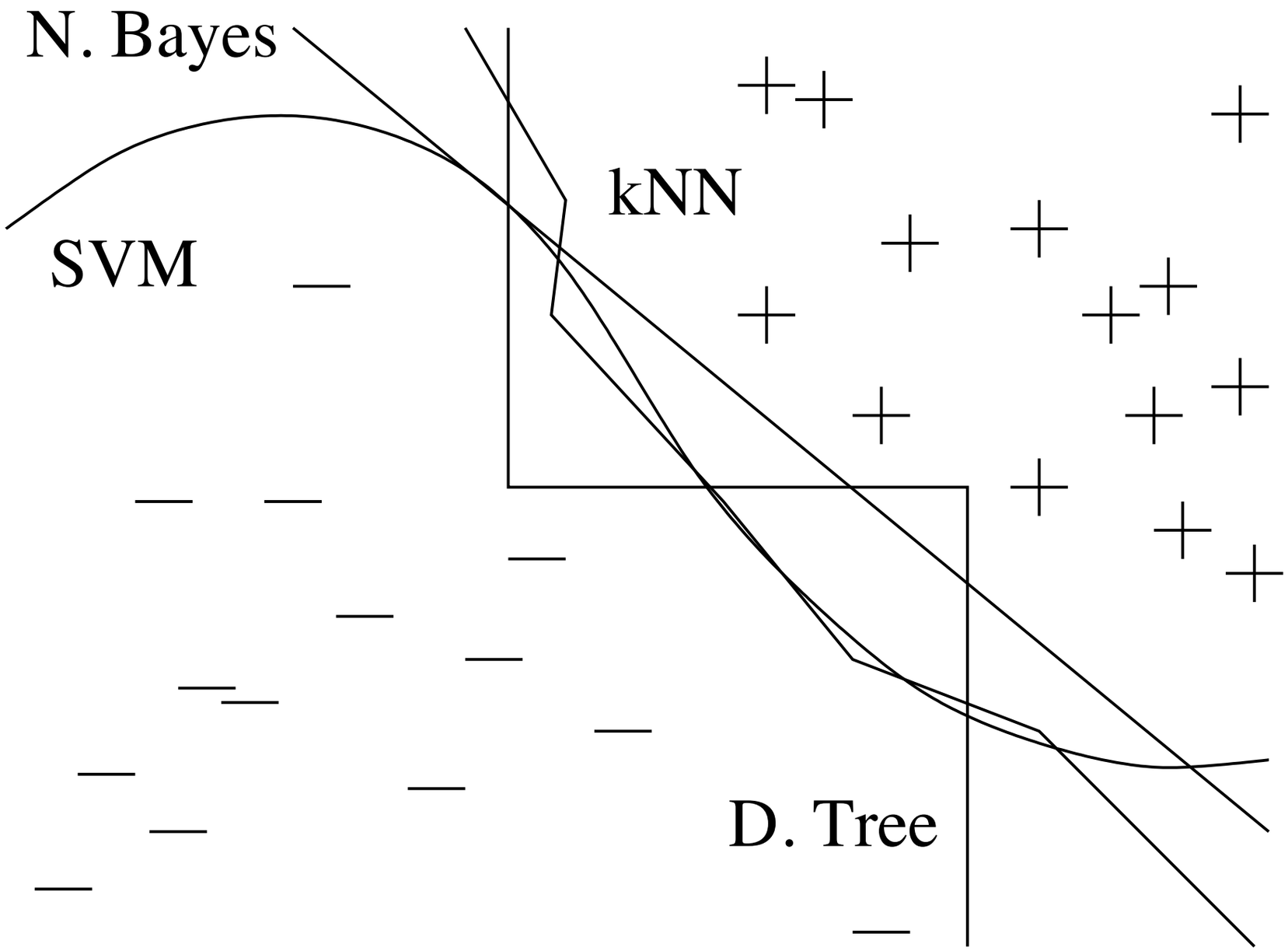
Data wins

- Crowdsourcing gets you data
- Crowdsourcing FTW

Why doesn’t choice of algorithms change outcome

- Clever algorithms have a smaller payoff than you might expect, since they all do the approximately the same thing
- This is surprising when you consider how different representations when using sets of rules versus Neural Nets

Different decision boundaries yield similar class predictions



What should you do?

- Try the simplest learners first (e.g., naive Bayes before logistic regression, k-nearest neighbor before support vector machines)
- More sophisticated learners are exciting, but they are usually harder to use, because they have more parameters you need to turn to get good results
- Put your efforts into good feature engineering and iterative experimentation

Many Models Is Better than One Model

- The single the best learner varies from application to application
- Combining multiple models (“ensemble learning”) often results in better results
- Creating such model ensembles is now standard

Ensemble techniques

- Bagging – generate random variations of the training set by resampling, learn a classifier on each, and combine the results by voting
- Boosting – each new classifier focuses on the examples the previous ones tended to get wrong, and weighs those examples more
- Stacking – the outputs of individual classifiers become the inputs of a “higher-level” learner that figures out how best to combine them

Netflix Prize

- Teams competed to build the best video recommender system
- Teams found that they obtained the best results by combining their learners with other teams', and merged into larger and larger teams.
- The winner and runner- p were both stacked ensembles of over 100 learners
- Combining the two ensembles further improved the results

Leaderboard

Showing Test Score. Click here to show quiz score

Display top 20 **leaders.**

Rank	Team Name	Best Test Score	% Improvement	Best Submit Time
Grand Prize - RMSE = 0.8567 - Winning Team: BellKor's Pragmatic Chaos				
1	BellKor's Pragmatic Chaos	0.8567	10.06	2009-07-26 18:18:28
2	The Ensemble	0.8567	10.06	2009-07-26 18:38:22
3	Grand Prize Team	0.8582	9.90	2009-07-10 21:24:40
4	Opera Solutions and Vandelay United	0.8588	9.84	2009-07-10 01:12:31
5	Vandelay Industries !	0.8591	9.81	2009-07-10 00:32:20
6	PragmaticTheory	0.8594	9.77	2009-06-24 12:06:56
7	BellKor in BigChaos	0.8601	9.70	2009-05-13 08:14:09
8	Dace_	0.8612	9.59	2009-07-24 17:18:43
9	Feeds2	0.8622	9.48	2009-07-12 13:11:51
10	BigChaos	0.8623	9.47	2009-04-07 12:33:59
11	Opera Solutions	0.8623	9.47	2009-07-24 00:34:07
12	BellKor	0.8624	9.46	2009-07-26 17:19:11

Take aways

- Learning = Representation + Evaluation + Optimization
- Generalization is the most important thing to learn
- Data enough is not alone; use knowledge about the problem too
- High dimensional problems are tricky
- Feature engineering is the key
- More data beats better algorithms
- Learn many models, not just one