Machine Learning

Ensemble learning

Outline

• What is ensemble learning?
• Ensemble methods
  – Bagging
  – Boosting
  – Feature set subsampling
  – Error-correcting output coding
• Why does ensemble learning work?

One model is not enough

• If the problem is difficult, an expert may call for a consultation
• What is the optimal way of combining results of many systems?
• Create different models (experts), and let them vote
• Committee is less biased, results should improve and stabilize, in complex domain modular learning is needed, but ...
  – reasons for decisions taken are difficult to understand (each expert has other reasons)
• A collection of models is called an ensemble, or a committee, hence the ensemble or committee learning
• Good models are desired, but those used in committee should specialize, otherwise committee will not give anything new

Single Learner

Ensemble Learning

How to Make an Effective Ensemble?

Two basic decisions when designing ensembles:
1. How to generate the base classifiers? $h_1, h_2, ...$
2. How to integrate/combine them? $F(h_1(x), h_2(x), ...)$
Question 2: How to integrate them

- Models $M_k$ should return estimation $P(\omega|X, M_k)$
- Simple voting: majority decision
- Weighted majority voting:
  \[ P(\omega|X, M) = \sum_k w_k P(\omega|X, M_k) \]
- Final model $M$ includes partial models $M_k$ and weights $w_k$; optimize weights to minimize some cost measure on the data
- Select the most confident models - perhaps they know better?
  - ex. take only $P(\omega|X, M_k) > 0.8$ into account
- Competent voting: in some regions of feature space some models are more competent than others (local learners)
  - Use a referee meta-model to select who is right

Question 1: How to generate base classifiers

- Homogeneous - Heterogeneous
- Manipulating the training set
  - Bagging
  - Cross-Validating Committees
  - Boosting
- Manipulating the feature set
  - Ensemble feature selection
- Manipulating the output classes
  - Error correcting output codes

Heterogeneous Models

- Different type of models are trained on the same training data
  - for example, decision trees, kNN, LDA, kernel methods combined in one committee
- Advantage: explores different biases of classifiers, takes advantage of hyperplanes and local neighborhoods in one system
- Rarely used

BAGGing = Bootstrap AGGregation (Breiman, 1996)

- for $i = 1, 2, \ldots, K$:
  - $T_i \leftarrow$ randomly select $M$ training instances with replacement
  - $h_i \leftarrow$ learn($T_i$) [ID3, NB, kNN, neural net, ...]
- Combine the $T_i$ together with uniform voting ($w_i=1/K$ for all $i$)

Bagging Example
Cross-Validated Committees

- Divide the data set into k subsets
- k overlapping feature subsets are constructed by dropping out a different one of these k subsets
- Train k classifiers based on a different training set
- Combine the classifiers together with uniform voting

Boosting

- Successive classifiers depends upon its predecessors
  - Previous methods: Individual classifiers were independent
- Training Examples may have unequal weights
- Look at errors from previous classifier step to decide how to focus on next iteration over data
- Set weights to focus more on ‘hard’ examples. (the ones on which we committed mistakes in the previous iterations)
- Iterative procedure

Boosting

Idea:
- Assign a weight to every training set instance
- Initially, all instances have the same weight
- As boosting proceeds, adjusts weights based on how well we have predicted data points so far
  - data points correctly predicted → low weight
  - data points mispredicted → high weight

Results: as learning proceeds, the learner is forced to focus on portions of data space not previously well predicted
AdaBoost Algorithm
(Freund and Schapire)

- \( W(x) \) is the distribution of weights over the \( N \) training points \( \sum W(x) = 1 \)
- Initially assign uniform weights \( W_0(x) = 1/N \) for all \( x \).
- At each iteration \( k \):
  - Find best weak classifier \( C_k(x) \) using weights \( W_k(x) \)
  - Compute \( t_k = \frac{\sum W_k(x) \cdot I(y \neq C_k(x))}{\sum W_k(x)} \)
  - weight \( a_k \) the classifier \( C_k \)'s weight in the final hypothesis Set
  - \( a_k = \log \left( \frac{1 - t_k}{t_k} \right) \)
  - For each \( x \), \( W_{k+1}(x) = W_k(x) \cdot \exp[a_k \cdot I(y \neq C_k(x))] \)
- \( C_{\text{final}}(x) = \text{sign} \left( \sum a_k C_k(x) \right) \)

Learning from weighted instances?

- One piece of the puzzle missing...
- So far, learning algorithms have just taken as input a set of learning instances, and all assumed to be equally important.
- What if we also get a weight vector saying how important each instance is?
- Turns out, it's very easy to modify most learning algorithms to deal with weighted instances:
  - ID3: Easy to modify entropy, information-gain equations to look at weight of such-and-such a set of instances, rather than the count (which simply assumes all weights=1).
Manipulating the Feature Set

- Multiple base classifiers can be constructed by subsampling the feature set
- Random subspace method
  - Random forests
- Search for the best subsampling method
- It is applied to problems with
  - High dimensional spaces
  - Multiple redundant features

Error Correcting Output Codes

- So far, we’ve been building the ensemble by tweaking the set of training instances
- ECOC involves tweaking the output (class) to be learned
Example: Handwritten number recognition

7 | 4 | 3 | 1 | 2
7, 4, 3, 5, 2

"obvious" approach: learn function: Scribble \( \rightarrow \{0,1,2,\ldots,9\} \)

\( \rightarrow \) doesn't work very well (too hard!)

What if we "decompose" the learning task into six "subproblems"?

1. learn an ensemble of classifiers, one specialized to each of the 6 "sub-problems"
2. to classify a new scribble, invoke each ensemble member, then predict the class whose code-word is closest (Hamming distance) to the predicted code

Designing code-words for ECOC learning

- Coding: \( k \) labels \( \rightarrow m \) bit codewords
- Good coding:
  - 1. row separation: "assigned" codes to be well-separated by lots of "unassigned" codes
  - 2. column separation: each bit \( i \) of the codes should be uncorrelated with all other bits \( j \)
- Selecting good codes is hard!

Results

- 20% decrease in error of ECOC over an ID3-like learning algorithm
- % decrease in error of ECOC over a neural network learner

Why Do Ensembles Work?

- Because uncorrelated errors of individual classifiers can be eliminated by averaging
- Assume: 40 base classifiers, majority voting, each error rate 0.3
- Probability of getting \( r \) incorrect votes from 40 classifiers

\[
P(r) = \frac{r!}{40^r} \left(0.3\right)^r \left(0.7\right)^{40-r}
\]

- \( p(\text{Ensemble is wrong}) = p(>20 \text{ incorrect votes}) \approx 0.01 \)
- This analysis makes lots of assumptions; can we say something "deeper"?
1. Statistical

• Given a finite amount of data, many hypotheses are typically equally good.
• How can the learning algorithm select among them?

h_ensemble = hypothesis from all data averaged h_1, h_2, ... may be better approximation to f than h_i

2. Representational

• The desired target function may not be implementable with individual classifiers, but may be approximated by ensemble averaging.

Representational (Example)

• Consider a binary learning task over [0,1] x [0,1], and the hypothesis space H of "discs"

• All learning algorithms do some sort of search through some space of hypotheses to find one that is "good enough" for the given training data.
• Since interesting hypothesis spaces are huge/infinite, heuristic search is essential (e.g. ID3 does greedy search in space of possible decision trees).
• So the learner might get stuck in a local minimum.
• One strategy for avoiding local minima: repeat the search many times with random restarts.

⇒ bagging
Summary...

- Ensembles: basic motivation – creating a committee of experts is typically more effective than trying to derive a single super-genius

- Key issues:
  - Generation of base models
  - Integration of base models

- Popular ensemble techniques
  - Manipulate training data: bagging and boosting (ensemble of “experts”, each specializing on different portions of the instance space)
  - Manipulate output values: error-correcting output coding (ensemble of “experts”, each predicting 1 bit of the (multibit) full class label)

- Why does ensemble learning work?