CSDS 440: Machine Learning

Soumya Ray (he/him, sray@case.edu)

Olin 516

Office hours T, Th 11:15-11:45 or by appointment

Problem Statement

- Given: A set of examples $(\mathbf{x}_i, y_i)$ over D features
- Do: Find a *subset* S of features ($|S| \leq D$) so that, for any other subset $S' \neq S$, a learned concept that uses S *generalizes better* than a learned concept that uses S'

Scoring criterion

- Often use information theoretic scoring criteria
- E.g. mutual information or information gain

$$IG(X) = H(Y) - H(Y \mid X) = I(X;Y) = I(Y;X)$$

- Need to be aware of calibration issues as before

Issue

- Just using gain often won't work
- Gain checks *relevance*, but not *redundancy*
- So we need to adjust the criterion to penalize adding multiple correlated features
- In order to account for redundancy, must move to a sequential selection procedure

Redundancy-aware Filtering

- Start with an empty feature set
- At each point, pick a feature that maximizes

$$Score(X \mid S) = I(X;Y \mid S)$$

- Where Y is the label and S the currently selected set of features
- Continue until k features selected

Issues

- In general, computing $I(X;Y|S)$ is difficult on a finite sample as S grows
- Also, beyond these criteria, we may want "stable" feature selection methods
  - i.e., the set of chosen features should not change a lot if the data is perturbed a little

Joint Mutual Information

- A good selection criterion that seems to work well in many cases and is practical to compute is the “JMI” criterion (Yang and Moody 99):

$$Score(X \mid S) = \sum_{X_j \in S} I(X, X_j; Y)$$

- But, as always, no single best technique is possible (ask for paper with detailed empirical study)

Causal Feature Selection

- Often, we want to interpret features as "causes" of the label
  - Leads to greater understanding of the task

- In such cases, we can try to extend the information theoretic approaches to causal approaches, using probabilistic graphical models (CSDS 491/442/600 ask for paper)

Pros and cons

- Filter methods are generally computationally very efficient
  - Often used as a quick and dirty first step
- But may produce suboptimal results
  - Strongly dependent on heuristic scoring function
  - Ignore the “chicken and egg” problem: the learning algorithm that will actually do the classification
  - May require additional ad-hoc assumptions if missing data present

Approach 2: Wrapper methods

- Features need to be selected *in the context of the learner* for best results

- Different features may work well for learners with different inductive biases

- Wrapper methods *search* through the feature set *using the learner's performance* as a guide

Search space

- The search space consists of all possible subsets of features

- It is intractable to search this exhaustively, so heuristic strategies (e.g. greedy search) are generally employed

Forward Feature Selection

- Set up feature selection as a *search for optimization* problem
- Initial search state: empty feature set
- Search operators: add a single feature
- Scoring function: Generalization error of classifier trained on the new set of features
- Termination condition: Generalization error does not improve

Forward Feature Selection

Need validation set/internal CV

- $F = \{\}$
- While *generalization error* improves
  - For each feature f not yet added to F
    - $F' = F \cup f$
    - Train a classifier with F' and evaluate it on a validation sample
    - Pick the f that yields the best error metric
    - If this error rate is better than the error rate using F, store F' as the new F

Methodological note

- It is very important to remember that feature selection must be done *within* cross validation
  - i.e. we must not use the test data to select features!
- This means that *different sets of features may be selected in each fold* during learning
- Also means that the evaluation step in Forward FS must be done with an *internal* cross validation loop

Methodological note

- Sometimes the following strategy is employed when deploying a classifier
- First, select features per fold and evaluate as normal
- After evaluation is complete, build a *consensus* feature set (e.g., features that were used by nearly all folds) and retrain on entire dataset
  – This classifier will be used on future data

Backward Feature Selection

- F={All features}
- While generalization error improves
  - For each feature f in F
  - $F' = F \setminus f$
  - Train a classifier with F' and evaluate it on a test sample
  - Remove the f that yields the best error rate
  - If this error rate is better than the error rate using F, store F' as the new F

Boosted feature selection

- Instead of a heuristic search, we can employ *boosting* as a wrapper to select features

- Here, the base learners are generally "decision stumps": decision trees with a single node
  - Remember this uses Information Gain to pick a feature to classify examples

Boosted feature selection

- The first feature is the one with max IG score
- Because of the way boosting works, the next feature will be the one that is best at classifying *the mistakes of the first feature*
  - The subsequent one will be the best at classifying the mistakes of the first two, etc.
- So each feature provides new, complementary information
  - Good at picking a “diverse” feature set with low redundancy, expected to generalize well
  - Found to work very well in practice

Pros and cons

- Wrapper methods generally provide very good performance because they are paired with a learner
  - Solve the chicken and egg problem iteratively
- Also robust to data issues like missing data (pass on to learner)
- But, need lots of data to work well (multiple cv loops)
- Also generally computationally very expensive (lots of training loops)
- So need a fast learner to work, or some way to heuristically estimate or update the scores
- Boosting provides a very good middle ground in this area

Approach 3: Embedded methods

- Embedded methods alter the objective functions of learning algorithms to *simultaneously* select features and learn the classifier
  - A good way to incorporate the learner into feature selection, but requires modifying learning algorithms

Modifying objectives

- Many learning algorithms optimize a loss function on the training sample:

$$L(\mathbf{w}) = \frac{1}{2}\sum_{i=1}^{m}(y_i - \mathbf{w}\bullet\mathbf{x}_i)^2$$

- Embedded methods add terms to this objective to encourage "sparseness"
  - i.e., encourage many zero w_i's

Modifying objectives

- An objective function that does this might be:

$$L(\mathbf{w}) = \frac{1}{2}\sum_{i=1}^{m}(y_i - \mathbf{w}\bullet\mathbf{x}_i)^2 + \lambda\|\mathbf{w}\|_0$$

The zero-norm is the number of elements in a vector that are non-zero

- This isn't continuous, but it indicates connections to overfitting control

Feature Selection and Overfitting Control

- We also modified learning objectives to control overfitting
- In fact, FS and OC have similar objectives
  - Both are trying to “simplify” the learned model and encourage generalization
  - Selecting a good set of features should help with overfitting control
  - Conversely, a classifier that is robust to overfitting should be using a good set of features

Feature Selection through Overfitting Control

- So if we consider our usual overfitting control strategy

The 2-norm, or "L2 penalty"

$$L_{OC}(\mathbf{w}) = \frac{1}{2}\sum_{i=1}^{m}(y_i - \mathbf{w}\bullet\mathbf{x}_i)^2 + \lambda\|\mathbf{w}\|_2^2, \ \|\mathbf{w}\|_2^2 = \sum_i w_i^2$$

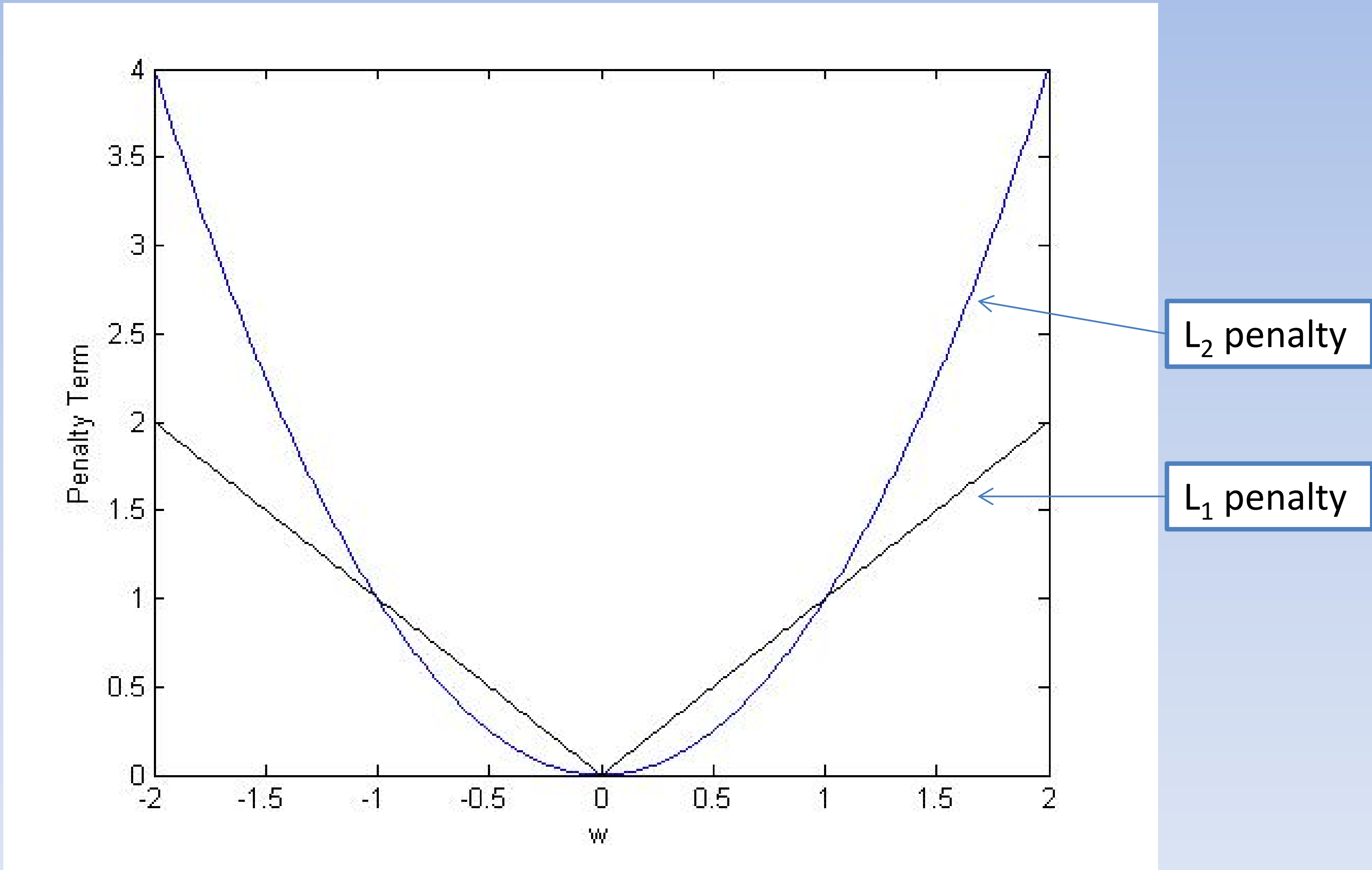
- This also encourages sparseness because each nonzero w pays a penalty in the objective
  - Is this a good way to select features?

Yes and No

- It turns out that the 2-norm penalty on $\mathbf{w}$ is somewhat effective, but it is possible to do better

- This is because the penalty for any nonzero w decreases quadratically as w becomes small, so usually the solution generally ends up with a lot of very small, but nonzero w's

Illustration



The LASSO

The one-norm is the sum of the absolute values of elements in a vector

- The L_1 penalty:

$$L_{FS}(\mathbf{w}) = \frac{1}{2}\sum_{i=1}^{m}(y_i - \mathbf{w}\bullet\mathbf{x}_i)^2 + \lambda\|\mathbf{w}\|_1, \|\mathbf{w}\|_1 = \sum_i |w_i|$$

- Does not have this problem
- This is also called the "LASSO" penalty ("least absolute shrinkage and selection operator") in the statistics literature

Why does the lasso work?

- Another way to write the objectives

$$\min_{\mathbf{w}} L_{FS}(\mathbf{w}) = \min_{\mathbf{w}} \frac{1}{2}\sum_{i=1}^{m}(y_i - \mathbf{w}\bullet\mathbf{x}_i)^2 + \lambda\|\mathbf{w}\|_1$$

$$\equiv \min_{\mathbf{w}} \frac{1}{2}\sum_{i=1}^{m}(y_i - \mathbf{w}\bullet\mathbf{x}_i)^2, s.t. \|\mathbf{w}\|_1 \le B$$

$$\min_{\mathbf{w}} L_{OC}(\mathbf{w}) = \min_{\mathbf{w}} \frac{1}{2}\sum_{i=1}^{m}(y_i - \mathbf{w}\bullet\mathbf{x}_i)^2 + \lambda\|\mathbf{w}\|_2^2$$

$$\equiv \min_{\mathbf{w}} \frac{1}{2}\sum_{i=1}^{m}(y_i - \mathbf{w}\bullet\mathbf{x}_i)^2, s.t. \|\mathbf{w}\|_2^2 \le B$$

Illustration