Introduction to Machine Learning

Feature Selection

Nils M. Kriege

WS 2023

Data Mining and Machine Learning
Faculty of Computer Science
University of Vienna

Credit: Slides based on the IML Lectures by Sebastian Tschiatschek and Andreas Krause

Feature selection

- In many high-dimensional problems, we may prefer not to work with all potentially available features
- **Why?**

Feature selection

- In many high-dimensional problems, we may prefer not to work with all potentially available features
- **Why?**
  - Interpretability (would like to “understand” the classifier, identify important variables/features)
  - Generalization (simpler models may generalize better)
  - Storage / computation / cost (don’t need to store / sum / acquire data for unused features)

Feature selection

- In many high-dimensional problems, we may prefer not to work with all potentially available features
- **Why?**
  - Interpretability (would like to "understand" the classifier, identify important variables/features)
  - Generalization (simpler models may generalize better)
  - Storage / computation / cost (don't need to store / sum / acquire data for unused features)
- **How?**

Feature selection

- In many high-dimensional problems, we may prefer not to work with all potentially available features
- **Why?**
  - Interpretability (would like to "understand" the classifier, identify important variables/features)
  - Generalization (simpler models may generalize better)
  - Storage / computation / cost (don't need to store / sum / acquire data for unused features)
- **How?**
  - Naive: try all subsets, and pick best (via cross-validation)

Demo: Feature selection for regression

[feature-selection/feature-selection.ipynb]

Greedy feature selection

- General purpose approach:
  Greedily add (or remove) features to maximize
  - Cross-validated prediction accuracy
  - Mutual information or other notions of informativeness (not discussed)
- Can be used for any method (not only linear regression/classification)

Details

- Let $\mathcal{D} = \{(\mathbf{x}_1, y_1), (\mathbf{x}_2, y_2), \ldots, (\mathbf{x}_n, y_n)\}$
- Set of all features: $\mathcal{V} = \{1, \ldots, d\}$ for $\mathbf{x}_i \in \mathbb{R}^d$
- Define cost function for scoring subsets S of $\mathcal{V}$:
  - Map instances to the features in $S = \{j_1, j_2, \ldots, j_k\}$:

$$\mathbf{x}_i = (x_{i,1}, x_{i,2}, \ldots, x_{i,d})^T \mapsto \mathbf{x}_{S,i} = (x_{i,j_1}, x_{i,j_2}, \ldots, x_{i,j_k})^T$$

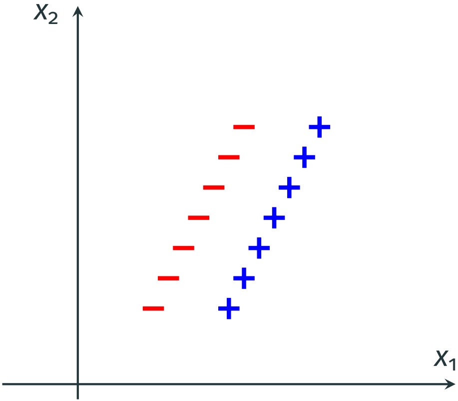
  - Evaluate on $\mathcal{D}_S = \{(\mathbf{x}_{S,1}, y_1), (\mathbf{x}_{S,2}, y_2), \ldots, (\mathbf{x}_{S,n}, y_n)\}$
  - $\hat{L}(S)$ is cross-validation error on $\mathcal{D}_S$

Greedy forward selection

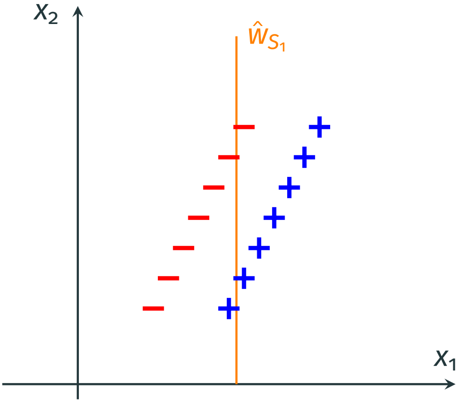
Greedy forward selection

- Start with $S = \emptyset$ and $E_0 = \infty$
- For $i = 1, 2, \ldots, d$:
  - find best element to add: $s_i = \text{argmin}_{j \in \mathcal{V} \setminus S} \hat{L}(S \cup \{j\})$
  - compute error: $E_i = \hat{L}(S \cup \{s_i\})$
  - If $E_i > E_{i-1}$ break, else set $S \leftarrow S \cup \{s_i\}$

Example

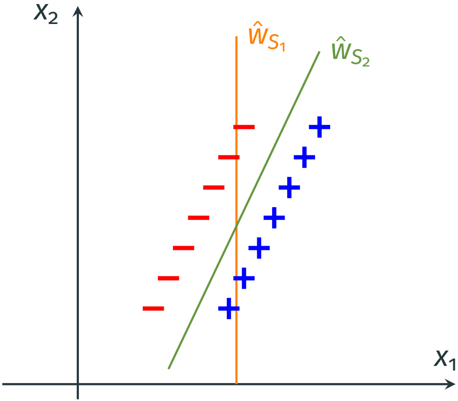


Example



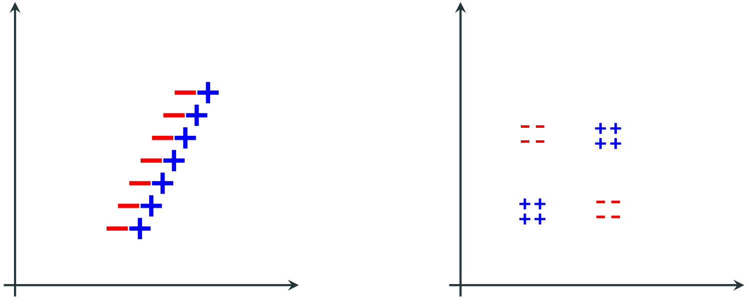
- $S_1 = \{x_1\}$

Example



- $S_1 = \{x_1\}$
- $S_2 = \{x_1, x_2\}$

Problems with greedy forward selection



Demo: Forward Selection for Regression

[feature-selection/feature-selection.ipynb]

Greedy backward selection

Greedy backward selection

- Start with $S = \mathcal{V}$ and $E_{d+1} = \infty$
- For $i = d, d-1, \ldots, 1$:
  - find best element to remove: $s_i = \text{argmin}_{j \in S} \hat{L}(S \setminus \{j\})$
  - compute error: $E_i = \hat{L}(S \setminus \{s_i\})$
  - If $E_i > E_{i+1}$ break, else set $S \leftarrow S \setminus \{s_i\}$

Demo: Backward Selection for Regression

[feature-selection/feature-selection.ipynb]

Comparison: FW vs. BW selection

Method	*Forward (FW)*	*Backward (BW)*
Advantages	Usually faster (if few relevant features)	Can handle “dependent features”

Problems with greedy feature selection

- ⚠ Computational cost (need to retrain models many times for different feature combinations)

Problems with greedy feature selection

- ⚠ Computational cost (need to retrain models many times for different feature combinations)
- ⚠ Can be suboptimal

Problems with greedy feature selection

- ⚠ Computational cost (need to retrain models many times for different feature combinations)
- ⚠ Can be suboptimal
- ❓ Can we solve the learning & feature selection problem simultaneously via a single optimization?

Linear models: Feature selection = Sparsity

- So far: explicitly select a subset of features:

$$\mathbf{x} = [x_1, \ldots, x_d] \quad \Rightarrow \quad \mathbf{x}_S = [x_{i_1}, \ldots, x_{i_k}]$$

  optimize over coefficients $\mathbf{w}_S = [w_{i_1}, \ldots, w_{i_k}]$:

$$\hat{\mathbf{w}}_S = \underset{\mathbf{w}_S}{\operatorname{argmin}} \sum_{i=1}^{n} \left(y_i - \mathbf{w}_S^T \mathbf{x}_{i,S}\right)^2$$

- This is equivalent to constraining $\mathbf{w}$ to be sparse (i.e., contain at most k non-zero entries)

Joint feature selection and training

- Would like to solve

$$\hat{\mathbf{w}} = \underset{\mathbf{w}}{\operatorname{argmin}} \sum_{i=1}^{n} \left(y_i - \mathbf{w}^T \mathbf{x}_i\right)^2 \quad \text{s.t. } \|\mathbf{w}\|_0 \leq k$$

  where $\|\mathbf{w}\|_0 = |\{i : w_i \neq 0\}|$ is the number of non-zeros in $\mathbf{w}$

- Alternatively, can penalize the number of nonzero entries:

$$\hat{\mathbf{w}} = \underset{\mathbf{w}}{\operatorname{argmin}} \sum_{i=1}^{n} \left(y_i - \mathbf{w}^T \mathbf{x}_i\right)^2 + \lambda \|\mathbf{w}\|_0$$

Making the optimization tractable

- Want to solve:

$$\hat{\mathbf{w}} = \operatorname*{argmin}_{\mathbf{w}} \sum_{i=1}^{n} (y_i - \mathbf{w}^T \mathbf{x}_i)^2 + \lambda \|\mathbf{w}\|_0$$

- ⚠ This is a difficult combinatorial optimization problem
- Can view greedy algorithms before as heuristics for solving it

Making the optimization tractable

- Want to solve:

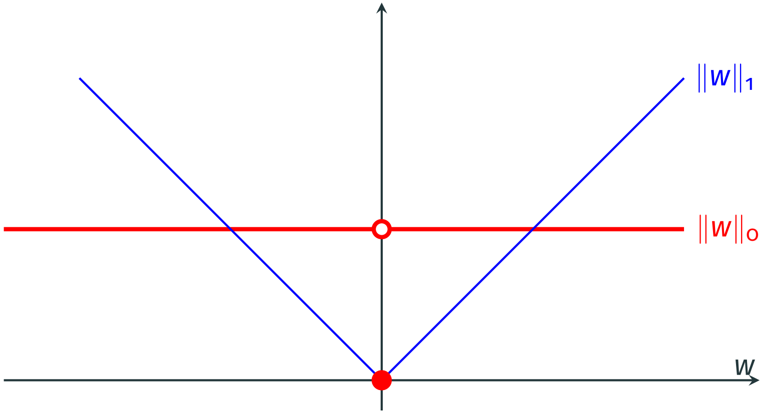
$$\hat{\mathbf{w}} = \operatorname*{argmin}_{\mathbf{w}} \sum_{i=1}^{n} (y_i - \mathbf{w}^T \mathbf{x}_i)^2 + \lambda \|\mathbf{w}\|_0$$

- ⚠ This is a difficult combinatorial optimization problem
- Can view greedy algorithms before as heuristics for solving it

- 💡 Replace $\|\mathbf{w}\|_0$ by a more tractable term:

$$\|\mathbf{w}\|_1 = \sum_i |w_i|$$

L1 as surrogate for L0



The “sparsity trick”

$$||\mathbf{w}||_0 \quad \Rightarrow \quad ||\mathbf{w}||_1$$

Sparse regression: The Lasso

- **Before:**
  - Ridge regression:

$$\min_{\mathbf{w}} \lambda \|\mathbf{w}\|_2^2 + \sum_{i=1}^{n} (y_i - \mathbf{w}^T \mathbf{x}_i)^2$$

  - Uses $\|\mathbf{w}\|_2^2$ to control weights

Sparse regression: The Lasso

- **Before:**
  - Ridge regression:

$$\min_{\mathbf{w}} \lambda \|\mathbf{w}\|_2^2 + \sum_{i=1}^{n} (y_i - \mathbf{w}^T \mathbf{x}_i)^2$$

  - Uses $\|\mathbf{w}\|_2^2$ to control weights
- **Slight modification:** replace $\|\mathbf{w}\|_2^2$ by $\|\mathbf{w}\|_1$
  - L1-regularized regression (Lasso):

$$\min_{\mathbf{w}} \lambda \|\mathbf{w}\|_1 + \sum_{i=1}^{n} (y_i - \mathbf{w}^T \mathbf{x}_i)^2$$

- This alternative penalty encourages coefficients to be exactly 0 $\Rightarrow$ automatic feature selection!

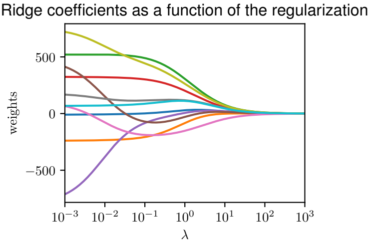
Lasso illustration

Lasso demo

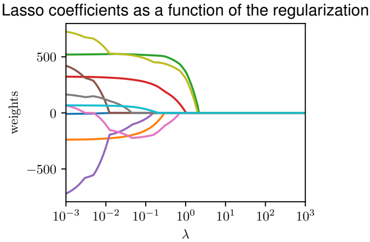
[feature-selection/lasso.ipynb]

Regularization paths

$\|\mathbf{w}\|_2^2$



$\|\mathbf{w}\|_1$



[feature-selection/regularization_path.py]

How to pick the regularization parameter?

How to pick the regularization parameter?

Cross-validation

Another example: L1-SVM

- Before:
  - Support vector machine:

$$\min_{\mathbf{w}} \lambda \|\mathbf{w}\|_2^2 + \sum_{i=1}^{n} \max(0, 1 - y_i \mathbf{w}^T \mathbf{x}_i)$$

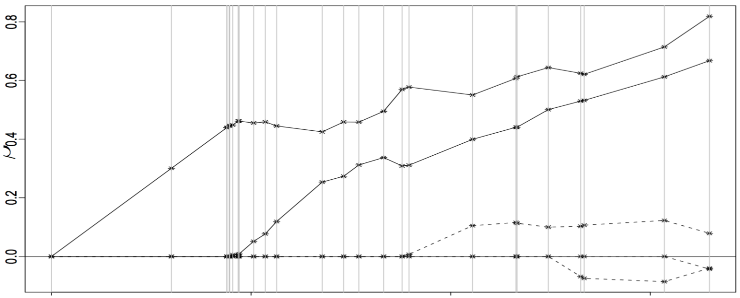
  - Uses $\|\mathbf{w}\|_2^2$ to control weights
- Apply sparsity trick: replace $\|\mathbf{w}\|_2^2$ by $\|\mathbf{w}\|_1$
  - L1-SVM:

$$\min_{\mathbf{w}} \lambda \|\mathbf{w}\|_1 + \sum_{i=1}^{n} \max(0, 1 - y_i \mathbf{w}^T \mathbf{x}_i)$$

- This alternative penalty encourages coefficients to be exactly 0 $\Rightarrow$ ignores those features!

Feature selection with L1-SVM

[Zhu, J., Rosset, S., Tibshirani, R., & Hastie, T. J. (2004). “1-norm support vector machines”. NeurIPS’04]



Experiment

[Zhu, J., Rosset, S., Tibshirani, R., & Hastie, T. J. (2004). "1-norm support vector machines". NeurIPS'04]

- Data:
  - 38 train, 34 test data from a DNA microarray classification experiment (leukemia diagnosis)
  - 7129 dimensions

Method	CV Error	Test Error	# of Genes
2-norm SVM UR	2/38	3/34	22
2-norm SVM RFE	2/38	1/34	31
1-norm SVM	2/38	2/34	17

l1-SVM demo

[feature-selection/l1-svm.ipynb]

Solving l1 regularized problems

- L1-norm is convex
- Combined with convex losses, obtain convex optimization problems (e.g., Lasso, l1-SVM, . . .)
- Can in principle solve using (stochastic) gradient descent
- However, convergence usually slow, and will rarely get "exact 0" entries
- Much recent work in convex optimization deals with solving such problems very efficiently
  E.g., proximal methods (not discussed in this class)

Comparison: Greedy selection vs. L1-Regularization

Method	*Greedy (FW/BW)*	*L1-Regularization*
Advantages	Applies to any prediction method	Faster (training and feature selection happen jointly)
Disadvantages	Slower (need to train many models)	Only works for linear models

What do you need to know

- What is feature selection
- Greedy algorithm (forward and backward)
- l1-regularization to encourage sparsity
  - Example: The Lasso (l1-regression)
  - Example: l1-SVM
- Advantages and disadvantages of the respective methods

Supervised learning summary so far

Representation/ features	Linear hypotheses, non-linear hypotheses through feature transformations
Model/ objective	Loss-function (squared loss, ℓ_p loss, 0/1 loss, Perceptron loss, Hinge loss) + Regularization (ℓ_2 norm, ℓ_1 norm, ℓ_0 penalty)
Method	Exact solution, Gradient Descent, (mini-batch) SGD, Greedy selection
Evaluation metric	Empirical risk = (mean) squared error, Accuracy
Model selection	k-fold cross-validation, Monte Carlo cross-validation

Supervised learning big picture so far