CS-E4710 Machine Learning: Supervised Methods

Lecture 10: Feature engineering and selection

Juho Rousu

November 23, 2021

Department of Computer Science
Aalto University

The importance of input representations

- So far in this course, we have (mostly) assumed the representation of the input data as feature vectors in $\mathbb{R}^d$
- But how do we come up with such representation?
- Is the given representation the best possible?
- It can be argued that good data representation is actually more important than the choice of the learning algorithm
- The importance of good data representations is exemplified by the fat there us nowadays a international conference on the topic (`https://iclr.cc`) with thousands of participants.

Feature engineering techniques

Variety of techniques:

- Feature transformation - convert the features in a form that allows learning better
- Feature selection - aim to reduce the number of input variables that are used by the predictor
- Feature generation - build new features by combining the original ones either manually (using prior knowledge) or by learning representations by optimizing some objective

Note: we used the term "variable" and "feature" interchangeably

Feature transformations

Feature transformations

- Simple transformations of the variables may have a large effect on the models
- One can obtain better error rates or faster optimization by transforming the variables
- Feature transformations are generally performed to make the distribution of the input variables to better represent the domain knowledge or to make the data more suitable to the learning algorithm
- However, feature transformation is largely art rather than science - it is hard to give any guarantees of the success of transforming the data

Examples of feature transformations

Let $\mathbf{f} = (f_1, \ldots, f_m)$ denote the values of a single variable in the dataset

- Centering: $f_i' = f_i - \bar{f}$ where $\bar{f} = \frac{1}{m}\sum_i f_i$ is the mean of variable - Rationale: learning algorithms generally learn from the variance of the data, not the mean. Centering makes the variance more obvious.
- Standardization: $f_i' = \frac{f_i - \bar{f}}{\sqrt{var(f)}}$, where $var(f) = \frac{1}{m}\sum_i (f_i - \bar{f})^2$ is the variance of variable. Rationale: making all variables have zero mean and unit variance may help if the raw variables have very different scales
- Unit range: $f' = \frac{f_i - f_{min}}{f_{max} - f_{min}}$, where f_{min} and f_{max} are the minimum and maximum values for the variable. Rationale: useful if the variable's relative position in the observed range is important
- Clipping: $f_i' = \text{sgn}(f_i)\min(b, |f_i|)$, for some threshold $b > 0$. Rationale: if certain large values are known to be non-informative, clipping may make learning easier (clipping small values: use max instead of min)

Examples of feature transformations

- Logarithmic transformation: $f_i' = \log(b + f_i)$, for some user-defined constant $b > 0$. Rationale: if small differences between small values (e.g. 0 vs. 1) are more important than small differences between large values (1000 vs. 1001), log-transform can be used to emphasize the former.
- Sigmoid transformation: $f_i' = \frac{1}{1+exp(bf_i)}$. Compresses the high absolute values heavily, "soft version" of clipping.
- Normalization of feature vectors: $\mathbf{x}_i' = \frac{\mathbf{x}_i}{\|\mathbf{x}_i\|}$ where $\mathbf{x}_i$ is a single training example. Rationale: useful if the relative values of the variables for a single example are important rather than the absolute values, e.g. if large object produces large average values for all features but the class of the object does not depend on its size

In general, several transformations are used to establish a desired effect, e.g. log-transform + normalization

Feature selection

Potential benefits of feature selection

- Facilitating data visualization and interpretation: a model with fewer features is easier to explain
- Reducing time and space requirements of training models: less data to store and compute over
- Improving the predictive performance: a small subset of variables is less likely to overfit

Guyon, I. and Elisseeff, A., 2003. An introduction to variable and feature selection. Journal of machine learning research, 3(Mar), pp.1157-1182.

Feature selection by exhaustive search

Given a set of d features, an idealized approach to feature selection would be to

- Exhaustively go through all $2^d - 1$ feature subsets
- Train a model using each subset
- Select the feature subset that gives the best predictive accuracy (e.g. by cross-validation)

While this approach would give us the optimal feature subset, it is prohibitively expensive unless d is very small

Feature selection in practice

In general, feature selection approach aim to avoid the exponential complexity of checking each feature subset

- Variable ranking: assess the usefulness of each input feature **individually** in a preprocessing step prior to learning, select a subset of most useful features
- Variable subset selection: generate several different **feature subsets** generally by some greedy search strategy, train a model with each subset, select the subset with the best predictive performance
- Embedded methods: the learning algorithm performs variable selection

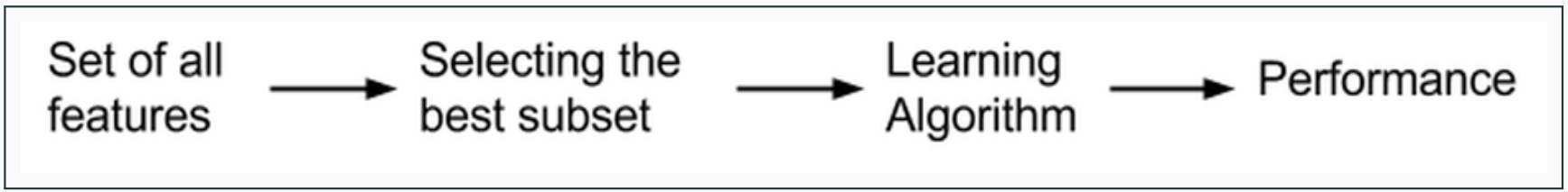
Variable ranking

Variable ranking approach

A generic variable ranking procedure:

- Compute a score s_j for each input variable j using some **scoring criterion**
- Sort the variables in descending order of s_j
- Select a subset of the most highly ranking variables, e.g. by top k variables or all variables exceeding a score threshold θ (k or θ generally decided by the user)

This approach is often called the "filtering" approach, due to the non-selected variables begin "filtered out"



Regression-based scoring criterion : Pearson correlation

- Assume a dataset $S = \{(\mathbf{x}_i, y_i)\}_{i=1}^m$, where $\mathbf{x}_i = (x_{i1}, \ldots, x_{id})^T$, and $y_i \in \mathbb{R}$
- (Empirical) Pearson correlation of the j'th feature and the output is given by

$$r_j = \frac{\sum_{i=1}^m (x_{ij} - \bar{x}_j)(y_i - \bar{y})}{\sqrt{\sum_i (x_{ij} - \bar{x}_j)^2} \sqrt{\sum_i y_i - \bar{y})^2}}$$

- Above, $\bar{x}_j = \frac{1}{m} \sum_{i=1}^m x_{ij}$ denotes the mean of the j'th feature in the dataset and $\bar{y} = \frac{1}{m} \sum_{i=1}^m y_i$ denotes the mean of the output variable
- r_j ranges from $+1$ (perfect correlation) to -1 (perfect anti-correlation)
- For feature scoring, we use $s_j = r_j^2$ to allow selection of both anti-correlated and correlated features

Regression-based scoring criterion : Pearson correlation

- Pearson correlation has a natural interpretation in terms of linear regression
- r_j is the optimal regression co-efficient in a univariate linear model

$$y = rx + b$$

that has the smallest mean squared error on the data

$$r_j = \text{argmin}_{r \in \mathbb{R}} \frac{1}{m} \sum_{i=1}^{m} (rx_{ij} + b - y_i)^2$$

- r_j^2 is the fraction of the variance of the output variable explained by the linear model
- $r_j^2 = 0$ means that the j'th feature **alone** does not explain any of the variance of the output

Classification-based scoring criteria

- Alternatively, one can use univariate classification based scoring criteria
- Given $y \in \{-1, +1\}$, a simple approach of to consider a model

$$y = \text{sgn}\left(ax + \theta\right)$$

where $a \in \{-1, +1\}$ and $\theta \in \mathbb{R}$ is set to optimize the empirical error

$$[a_j, \theta_j] = \text{argmin}_{a \in \{-1,+1\}, \theta \in \mathbb{R}} \sum_{i=1}^{m} \mathbf{1}_{\text{sgn}(ax_{ij}+\theta) \neq y_i}$$

- The above is essentially the "decision stump" we used in the AdaBoost example (Lecture 9)
- For feature scoring, we can use the accuracy: $r_j = 1 - \frac{1}{m} \sum_{i=1}^{m} \mathbf{1}_{\text{sgn}(a_j x_{ij}+\theta_j) \neq y_i}$
- Also can use other evaluation metrics for classification (Lecture 1)

Pros and cons of the filtering approach

- The filtering approach is reasonably efficient:
  - computation of the feature scores can be done in $O(md)$ for correlation and $O(dm \log_2 m)$ time for the decision stump (the dominating cost is sorting the values of the feature j in $O(m \log_2 m)$ time).
  - The ranking of the features given the scores also requires sorting the features in $O(d \log d)$ time.
- However, the method is limited by two issues:
  - Features that are not correlating with the output may still be useful when combined by other variables
  - As the filtering is made independently of the predictive model that will use the selected features, the selected subset might not be optimal

Example 1: Useless variables may be useful in combination of other variables

In the figure, we have toy dataset of two input variables and binary output (open and closed circles)

- The data lies in four clusters in a "XOR-like" layout
- The marginal distribution of neither input variable (top left and bottom right) give any separation of the classes
- Combination of the two variables can perfectly separate the classes (e.g. by intersection of two hyperplanes)

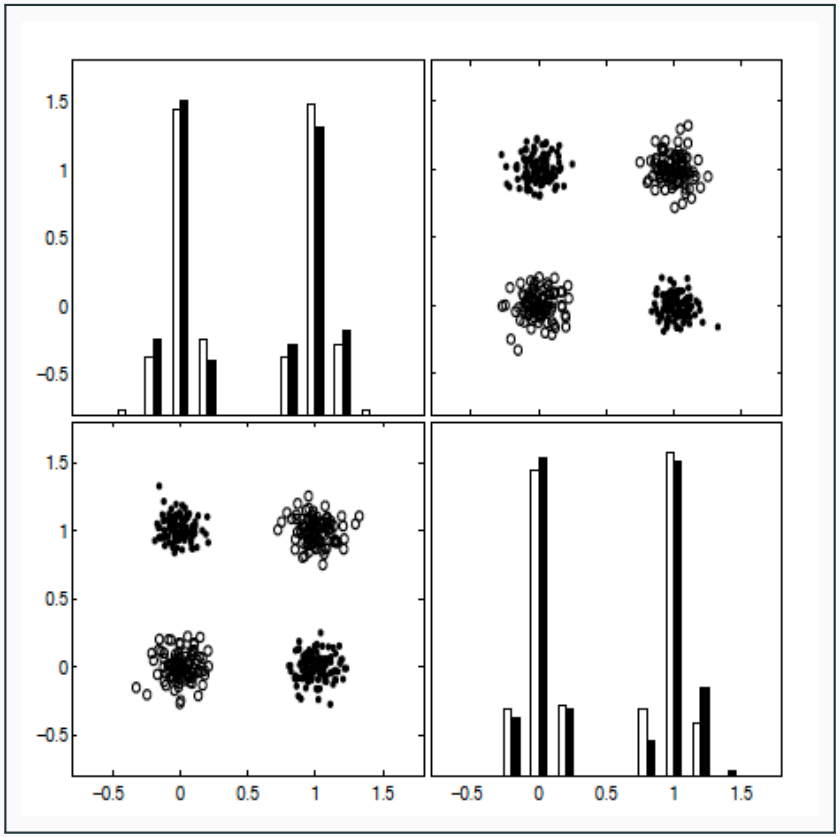


Figure : Guyon and Elisseef, 2003. Top let and bottom right pane show the distribution of the data along the ranges two features, respectively. The top right and bottom left panes show the same data, with the coordinate axes swapped

Example 2: Useless variables may be useful in combination of other variables

In the second example the two classes have high within class covariance in the two input variables, which gives the elongated clusters

- One of the variables alone does not give any separation of the classes (top left)
- The other variable gives partial separation, but with some overlap (bottom right)
- However, using both variables gives perfect separation in 2D plane (e.g. by a single hyperplane)

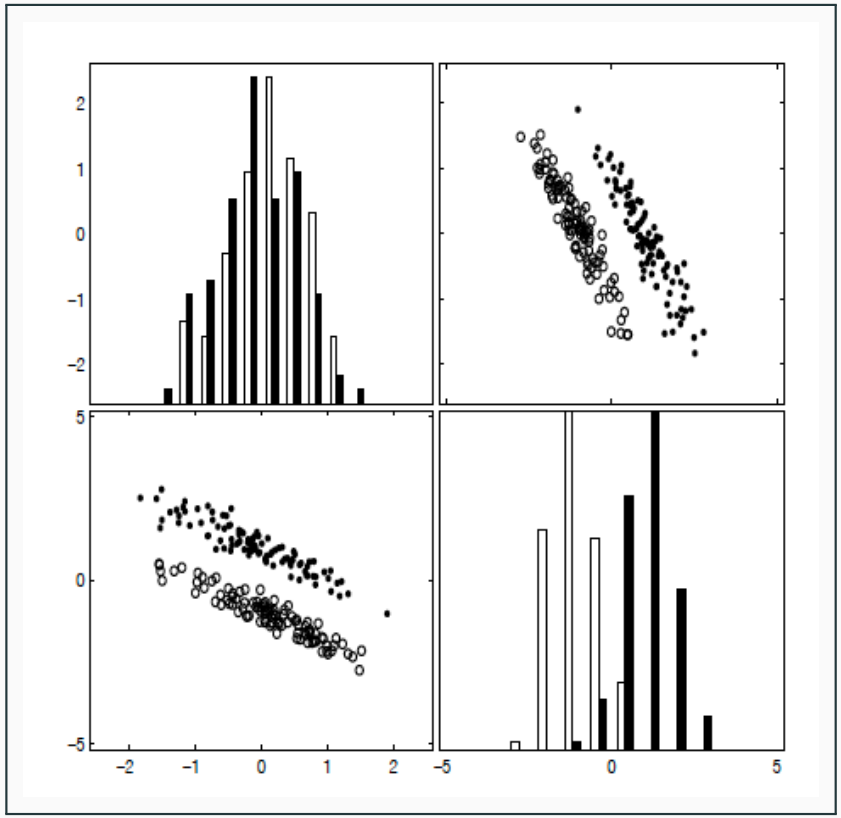


Figure : Guyon and Elisseef, 2003. Top left and bottom right pane show the distribution of the data along the ranges two features, respectively. The top right and bottom left panes show the same data, with the coordinate axes swapped

Variable subset selection

Wrapper approach

- The wrapper approach to variable selection uses the same learning algorithm that is used for the final model to evaluate variable subsets
- It iterates two steps:
  - Generate a variable subset (by some fixed procedure)
  - Evaluate the subset by training a model with the subset

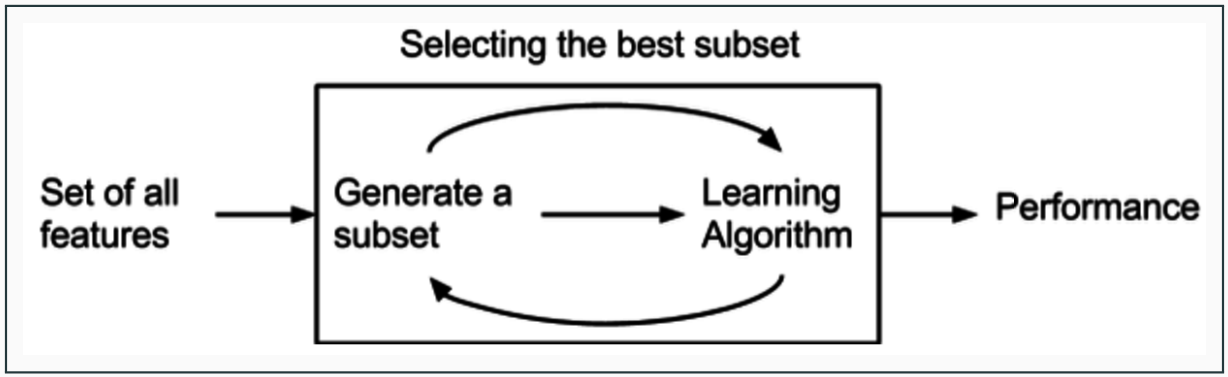
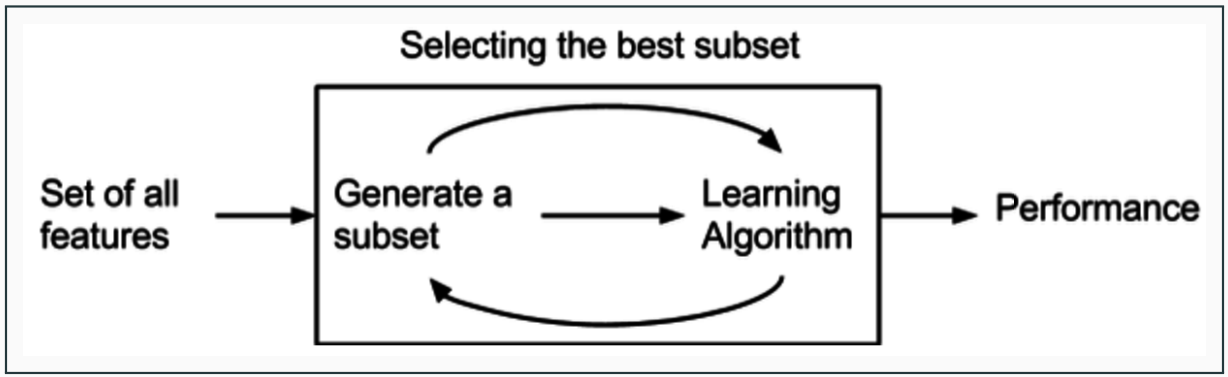


Image By Lastdreamer7591 - Own work, CC BY-SA 4.0, https://commons.wikimedia.org/w/index.php?curid=37208688

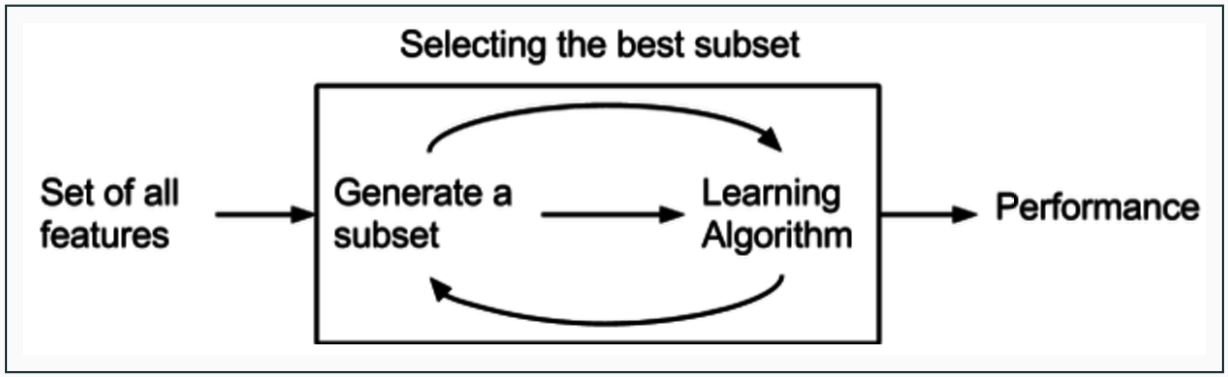
Wrapper approach

- Two generic procedures for generating a subset:
  - Forward selection - grow the set of selected variables by iteratively
  - Backward elimination - start from set of all variables and iteratively eliminate variables until a given stopping criterion is fulfilled
  - Also more thorough search strategies can be used (best-first, branch-and-bound, etc.)



Variable scoring in wrapper approach

- In a wrapper algorithm it is natural to use the risk of the hypothesis (either on training or validation data) as the variable scoring criterion
- This involves training and testing a hypothesis for each variable subset considered
  - Computationally heavier than the filter approach
  - But can find better variable subsets than the filter approach



Greedy forward selection

- In greedy forward selection, we add a variables iteratively, choosing in each step a variable that gives the maximum improvement
- The variables that have been chosen to the model are not removed, even if they become redundant (no back-tracking)
- For each iteration, models are trained for every candidate variable to be added
- In total $O(Kd)$ models are trained where K is the upper bound for the number of variables selected

Greedy forward selection pseudo-code

Input: Dataset $S = \{(\mathbf{x}_i, y_i)\}_{i=1}^m$

Input: Maximum number of variables K

Output: A subset of selected variables $J \subset \{1, \ldots, d\}$

Initialize $J \leftarrow \emptyset, s_J \leftarrow 0; k \leftarrow 0$

repeat

for $j \in \{1, \ldots, d\} \setminus J$ **do**

Train a model $f_{J \cup j}$ with input variables $J \cup j$

$s_{J \cup j} \leftarrow$ accuracy of the model $f_{J \cup j}$ (training set or validation set)

end for

Select the best variable: $j^* \leftarrow \text{argmax}_{j \in \{1,\ldots,d\} \setminus J} s_{J \cup j}$

if $s_{J \cup j^*} > s_J$ **then**

$J \leftarrow J \cup j^*; k \leftarrow k + 1$

else

$stop \leftarrow TRUE$

end if

until $k = K$ or $stop$

Backward elimination

- In backward elimination, one starts from the full set of input variables
- It is argued that backward elimination is more effective in finding good variable subsets than forward selection
- In each stage, the input variable whose elimination has the best effect on the model is eliminated
    - Largest increase or smallest decrease of accuracy on validation set
- Accuracy on validation set generally used, since it will generally have an optimum for some number of selected variables
- In total $O(d^2)$ models are trained

Backward elimination pseudo-code

Input: Training set $S = \{(\mathbf{x}_i, y_i)\}_{i=1^m}$, validation set $V = \{(\mathbf{x}_i, y_i)\}_{i=1}^{n}$

Output: A subset of selected variables $J_{max} \subset \{1, \ldots, d\}$

Initialize $J \leftarrow \{1, \ldots, d\}$; $s_{max} = 0$; $J_{max} = J$

repeat

for $j \in J$ **do**

Train a model $f_{J \setminus j}$ with input variables excluding j

$s_{J \setminus j} = \leftarrow$ accuracy of the model $f_{J \setminus j}$ on the validation set

end for

Variable whose elimination gives the best model:

$j^* \leftarrow \mathrm{argmax}_{j \in J} s_{J \setminus j}$

$J \leftarrow J \setminus j^*$

Keep track of the best model:

if $s_J > s_{max}$ **then**

$s_{max} = s_J$; $J_{max} = J$

end if

until $J = \emptyset$

The case for backward selection

A simple example demonstrates a case where forward selection may miss the best subset

- Variable 3 separates the two classes best when used as a single variable: forward selection would likely choose Variable 3
- However, variables 1 and 2 separate the classes perfectly as a pair: backward selection is likely to remove variable 3, since its redundant when variables 1 and 2 are in the subset.

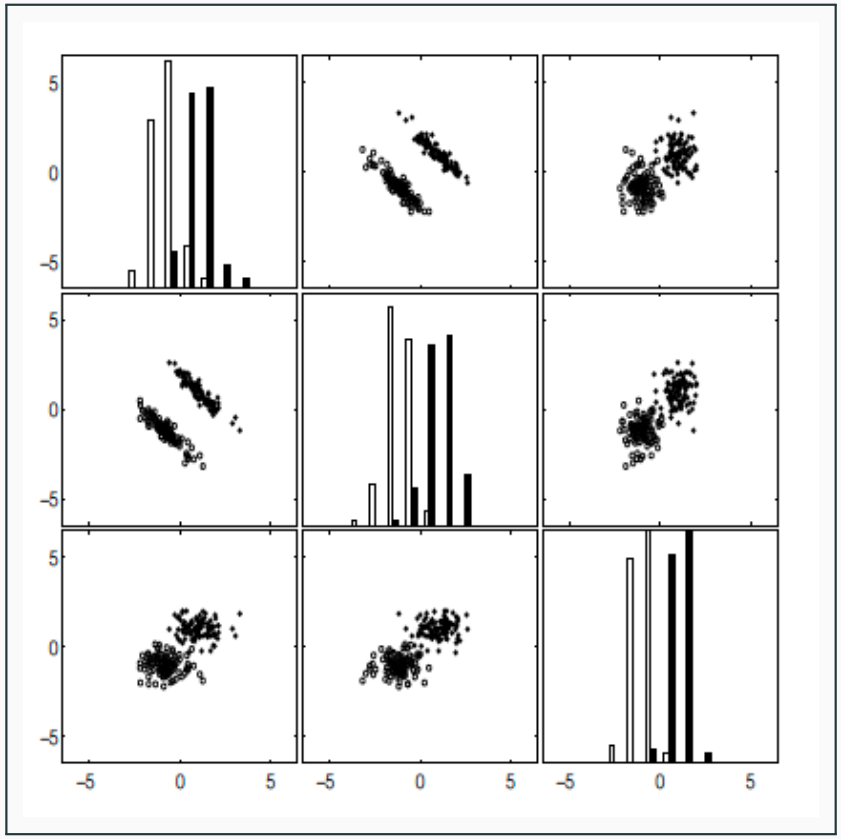
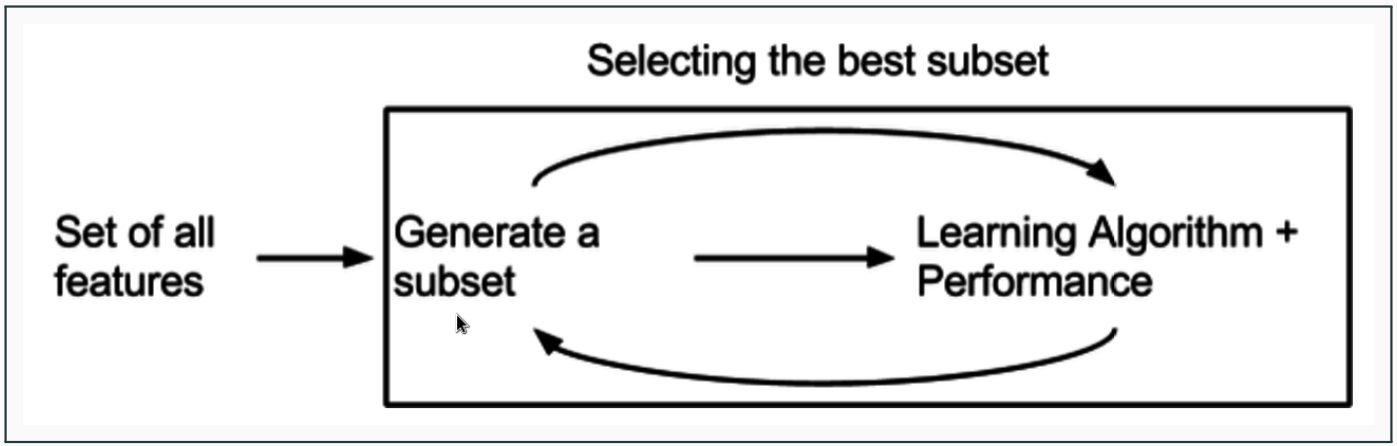


Figure : Guyon and Elisseef, 2003. The diagonal panes show the histograms of the class distributions for each variable. The off diagonal blocks show the scatter plots of the data using each pair of the variables

Embedded feature selection methods

Embedded feature selection algorithms refer to cases where the learning algorithm itself is selecting features as part of searching for the best model

- Models based on logical tests on single variable values e.g. decision trees
- Boosting algorithms using base hypotheses on single variables
  - AdaBoost on decision stumps can be seen as an embedded feature selection method
- Sparse modelling methods: focused on penalizing the weights with **sparsity inducing norms**



Sparse modelling

Sparsity-inducing norms: : ℓ_0

- Consider the problem of minimizing the empirical risk of a linear model subject to a budget of K variables

$$\min_{\mathbf{w}\in\mathbb{R}^d} \sum_{i=1}^{m} L(\mathbf{w}^T\mathbf{x}_i, y_i) s.t. \qquad |\{j|w_j \neq 0\}| \leq K$$

  where $\mathbf{w}^T\mathbf{x}$ is a model with the weight vector $\mathbf{w}$

- The size of the set $\{j|w_j \neq 0\}$ corresponds to the ℓ_0 norm of $\mathbf{w}$:

$$\|\mathbf{w}\|_0 = |\{j|w_j \neq 0\}|$$

- With small K, the optimal solutions to the above problem are sparse (with at most K non-zero weights)

Sparsity-inducing norms: ℓ_0

- Alternatively, the optimization problem can be written as a regularized learning problem

$$\min_{\mathbf{w}\in\mathbb{R}^d} \frac{1}{m}\sum_{i=1}^{m} L(\mathbf{w}^T\mathbf{x}_i, y_i) + \lambda\|w\|_0$$

- if $\lambda > 0$ is sufficiently large, the optimal solution will be a sparse containing a small number of non-zero weights
- However, both the constraint version (previous slide) and the above penalization version are computationally hard to solve (non-convex, NP-hard)

Sparsity-inducing norms: ℓ_1

- Replacing the non-convex $\|\mathbf{w}\|_0$ by the ℓ_1 norm

$$\|\mathbf{w}\|_1 = \sum_{j=1}^{d} |w_j|$$

gives a convex objective (assuming the loss L is also convex):

$$\min_{\mathbf{w}\in\mathbb{R}^d} \frac{1}{m} \sum_{i=1}^{m} L(\mathbf{w}^T \mathbf{x}_i, y_i) + \lambda \|\mathbf{w}\|_1$$

- It turns out that the above ℓ_1 regularized problem often returns sparse solutions
- But what is so special about ℓ_1 norm?

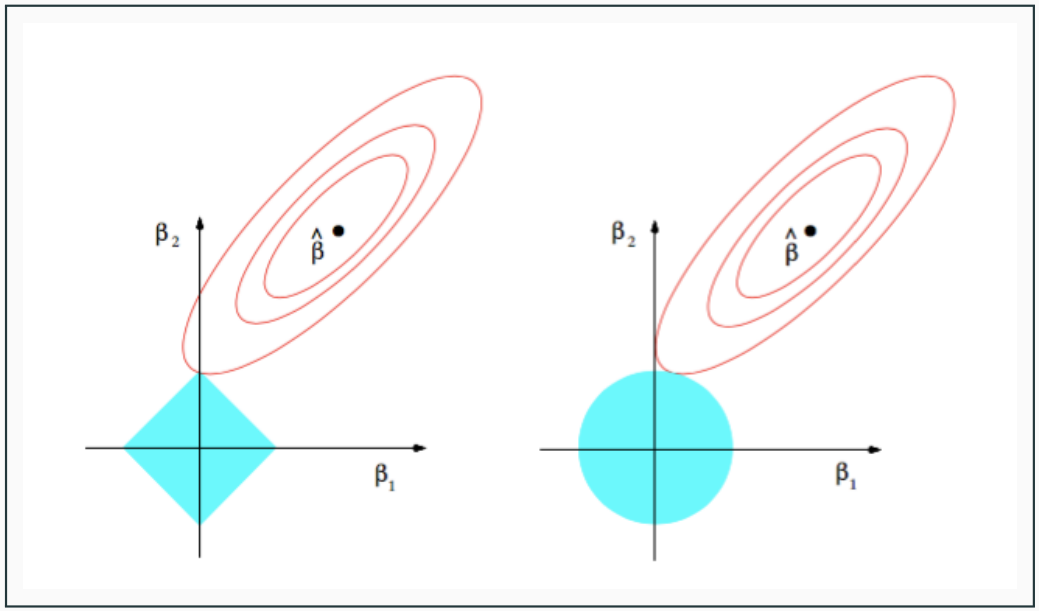
Sparsity-inducing norms: ℓ_1

Consider a simple example of minimizing mean squared error (MSE) on two input variables

$$\hat{\beta} = \text{argmin}_{\beta_1,\beta_2} \frac{1}{m} \sum_{i=1}^{m} (\beta_1 x_{1i} + \beta_2 x_{2i} - y_i)^2 + \lambda \|\boldsymbol{\beta}\|_p$$

where $p = 1, 2$

- The turqoise diamond ($p = 1$) and ball ($p = 2$)denote the constraint regions $\|\boldsymbol{\beta}\|_p \leq 1$
- The co-centric ellipses denote the level sets of equal MSE



Hastie, T., Tibshirani, R. and Wainwright, M., 2015. Statistical learning with sparsity: the lasso and generalizations. CRC press.

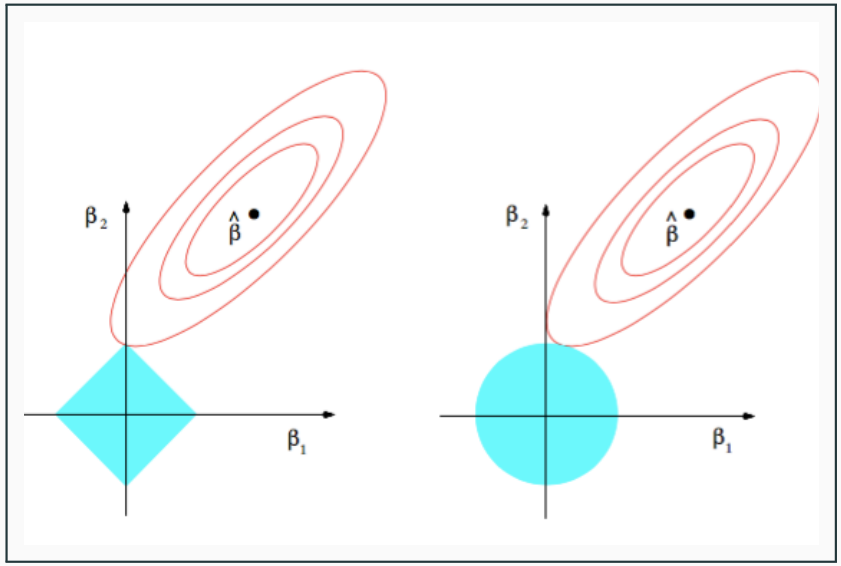
Sparsity-inducing norms: ℓ_1

Consider a simple example of minimizing mean squared error (MSE) on two input variables

$$\hat{\beta} = \text{argmin}_{\beta_1,\beta_2} \frac{1}{m} \sum_{i=1}^{m} (\beta_1 x_{1i} + \beta_2 x_{2i} - y_i)^2 + \lambda \|\boldsymbol{\beta}\|_p$$

where $p = 1, 2$

- The optimal model is at the intersection of the smallest ellipse
- With $p = 1$ this intersection is often at the corners of the diamond
- At each corner, one of the variables will have zero weight $\Rightarrow$ sparsity
- With $p = 2$ the solutions are more likely to have non-zero weights for all variables



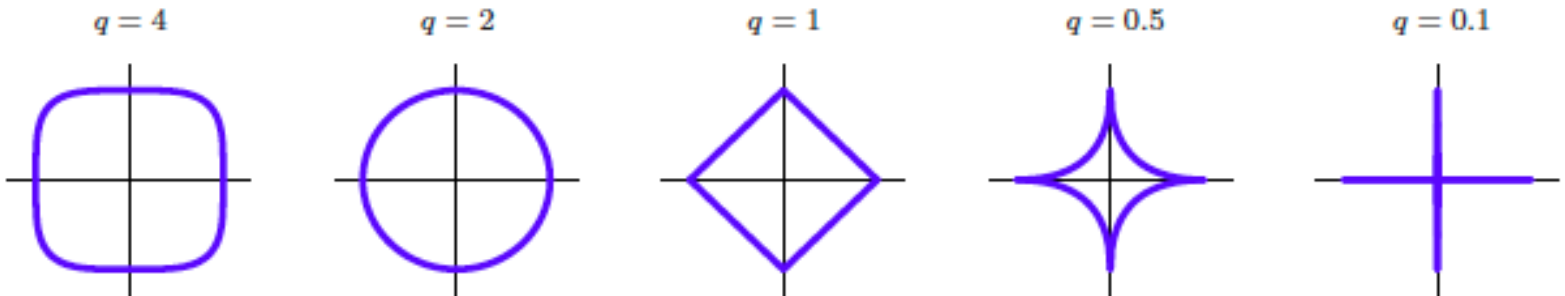
Hastie, T., Tibshirani, R. and Wainwright, M., 2015. Statistical learning with sparsity: the lasso and generalizations. CRC press.

Sparsity-inducing norms: ℓ_p

- But is some other p also possible?
- For general p, the ℓ_p norm is given by

$$\|\mathbf{w}\|_p = (\sum_{j=1}^{d} |w_j|^p)^{1/p}$$

- For $p < 1$ the constraint region $\|\mathbf{w}\|_p \leq 1$ becomes non-convex with "spikes" extending towards the corners: will give sparsity but hard to optimize
- For $p > 1$ the the constraint region is more and more "ball-like" and "box-like" and will give less and less sparsity
- $p = 1$ gives the most sparse solutions while keeping optimization problem convex



Sparse learning problems

Combining ℓ_1 regularisation with different loss functions gives sparse variants of well-known models

- Hinge loss $\Rightarrow$ Sparse SVM: $\min_{\mathbf{w}} \frac{1}{m} \sum_{i=1}^{m} L_{Hinge}(\mathbf{w}^T \mathbf{x}_i, y_i) + \lambda \|\mathbf{w}\|_1$
- Logistic loss $\Rightarrow$ Sparse logistic regression: $\min_{\mathbf{w}} \frac{1}{m} \sum_{i=1}^{m} L_{Logistic}(\mathbf{w}^T \mathbf{x}_i, y_i) + \lambda \|\mathbf{w}\|_1$
- Squared loss $\Rightarrow$ Sparse regression, LASSO: $\min_{\mathbf{w}} \frac{1}{m} \sum_{i=1}^{m} L_{sq}(\mathbf{w}^T \mathbf{x}_i, y_i) + \lambda \|\mathbf{w}\|_1$

Sparse modelling can also be used in applications that go beyond classification. See e.g. the book Hastie, T., Tibshirani, R. and Wainwright, M., 2015. Statistical learning with sparsity: the lasso and generalizations. CRC press.

Stability of feature selection

Stability of feature selection

A recognized problem with variable subset selection and sparse modelling approaches is their sensitivity to small perturbations

- upon removal or addition of a few variables or examples
- addition of noise
- initial conditions of the algorithms

The lack of stability may sometimes be a problem

- It may be a symptom of a "bad" model, one that will not generalize well
- The results are not reproducible
- One variable subset fails to capture the "whole picture"

Stability of feature selection

- A general approach to tackle the lack of stability is to use bootstrapping
- One runs the variable selection or sparse modelling algorithm on T sub-samples, drawn with replacement from the original training data
- The final variable subset may be selected as
  - The union of all selected features in the T models, or
  - Counting how many models include a given variable j and selecting the variables that occur at least given fraction of the T models

Summary

- Feature transformations can be used to improve the input data prior to learning
- Feature selection is used to improve the interpretability, resource consumption and predictive performance of machine learning
- Hard computational problem: many heuristics to speed it up, including greedy forward selection and backward elimination
- Sparsity inducing norms can be used to learn a model with small number of features