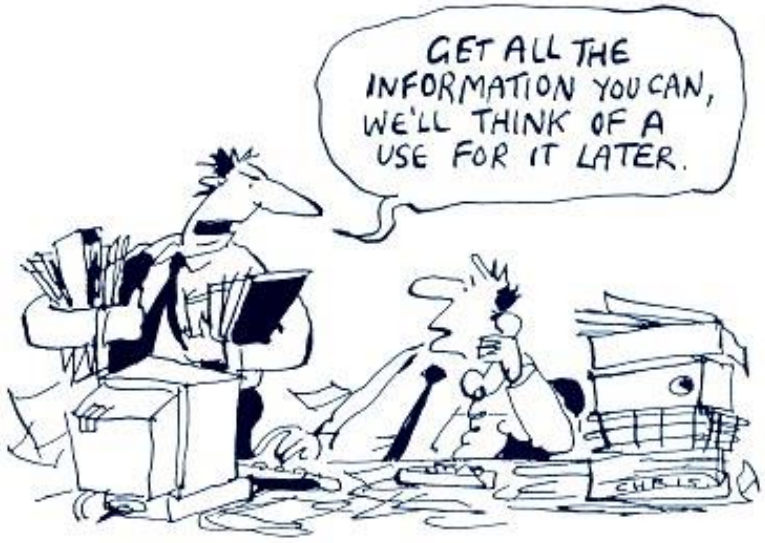
Data preprocessing: feature engineering



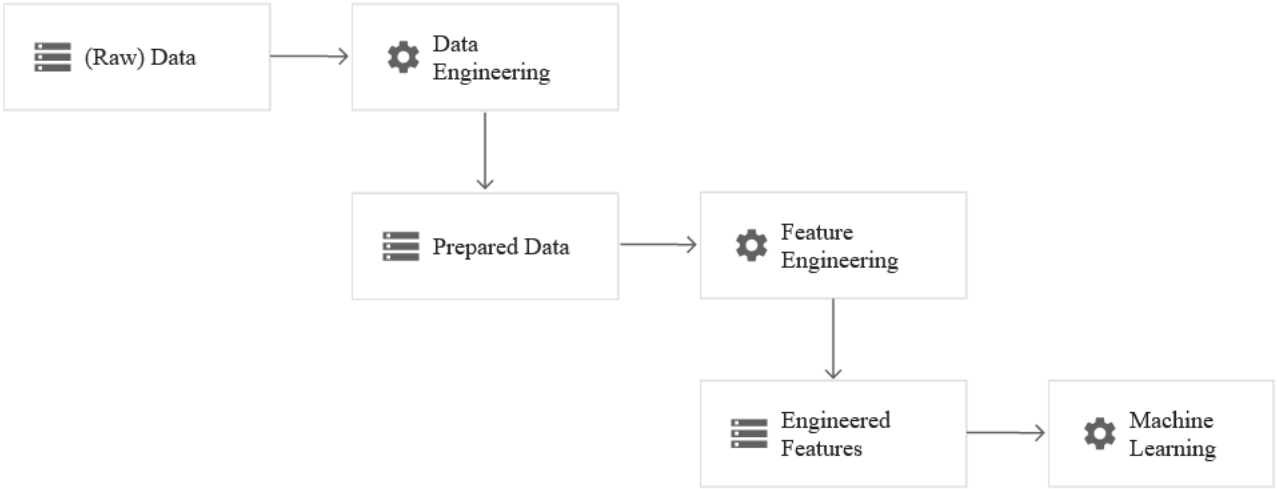
Prof Dr Marko Robnik-Šikonja
Intelligent Systems, Edition 2024

Contents

- Data preprocessing
- Feature subset selection: filter, wrapper and embedded methods
- Model evaluation
- Dimensionality reduction
- Feature selection extensions: unsupervised and semi-supervised learning, multi-task, multi-view, multi-label learning

First steps in ML

- Ther data preparation step is seriously underestimated



Data preprocessing

- **Data cleansing:** removing or correcting records that have corrupted or invalid values from raw data, and removing records that are missing a large number of columns.
- **Instances selection and partitioning:** selecting data points from the input dataset to create training, evaluation (validation), and test sets. This process includes techniques for repeatable random sampling, minority classes oversampling, and stratified partitioning.
- **Feature tuning:** improving the quality of a feature for ML, which includes scaling and normalizing numeric values, imputing missing values, clipping outliers, and adjusting values that have skewed distributions.
- **Feature transformation:** converting a numeric feature to a categorical feature (through discretization), and converting categorical features to a numeric representation (through one-hot encoding, sparse and dense feature embeddings). Some models work only with numeric or categorical features, while others can handle mixed type features. Even when models handle both types, they can benefit from different representations (numeric and categorical) of the same feature.
- **Feature extraction:** reducing the number of features by creating lower-dimension, more powerful data representations using techniques such as PCA, embedding extraction, and hashing.
- **Feature selection:** selecting a subset of the input features for training the model, and ignoring the irrelevant or redundant ones, using filter or wrapper methods. Feature selection can also involve simply dropping features if the features are missing a large number of values.
- **Feature construction:** creating new features by using different ooperators, such as logicač, arithmetical, trigonometrical, etc. Features can also be constructed by using domain knowledge, e.g., business logic from the domain of the ML use case.
- **For unstructured data:** often only modest preprocessing is needed for neural networks
  - text: casing, tokenization, embedding lookup/calculation
  - images: resizing, cropping, filters.

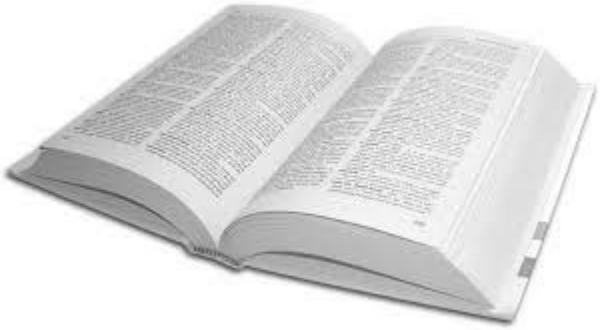
Why Reduce Dimensionality?

- Reduces time complexity: Less computation
- Reduces space complexity: Less parameters
- Saves the cost of observing the feature
- Simpler models are more robust on small datasets
- More interpretable; simpler explanation
- Data visualization (structure, groups, outliers, etc) if plotted in 2 or 3 dimensions

Feature subset selection

- Choose a small subset of the relevant features from the original features by removing irrelevant, redundant and/or noisy features
- The aim: better learning performance, i.e. higher learning accuracy, lower computational cost, or better model interpretability

Huge number of features



- Text classification, ≈ 50,000 words in a dictionary
- Bioinformatics, ≈ 10,000 measurements of gene expression levels
- Computer vision, ≈ 1,000,000 pixels

Evaluation of attributes

- Numerical evaluation and ranking of the attributes
- The success of the evaluation procedure depends on the role it plays in learning:
  - feature subset selection
  - building of the tree-based models
  - constructive induction
  - discretization
  - attribute weighting
  - comprehension
  - prediction
  - etc.

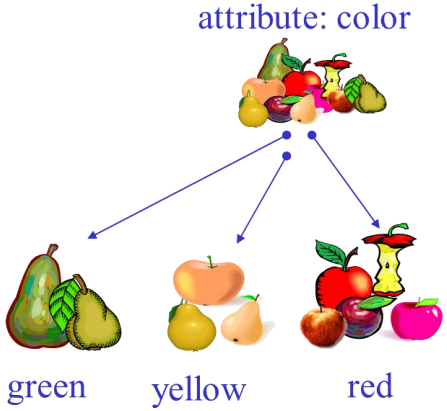
Attribute description

color	weight	shape	size	sort
red	12	round	middle	apple
yellow	20	conic	large	pear
red	15	round	tiny	apple
green	8	round	small	pear
yellow	22	conic	large	apple
mixed	12	conic	small	apple
green	15	round	middle	apple
mixed	8	round	tiny	apple
yellow	6	round	small	pear

- nominal attributes: ordered and unordered
- numeric attributes

Feature evaluation

- in order to select attributes, we have to evaluate (rank) them
- the success of feature evaluation is measured through the success of learning
- an example: feature evaluation in decision tree building
  - in each interior node of the tree an attribute is selected which determines split of the instances
  - the attributes are evaluated to ensure useful split



Three types of feature selection methods

- Filter methods: independent on learning algorithm, select the most discriminative features through a criterion based on the character of data, e.g. information gain and ReliefF

- Wrapper methods: use the intended learning algorithm to evaluate the features, e.g., progressively add features to SVM while performance increases

- Embedded method select features in the process of learning

Feature selection: Filter methods

Heuristic measures for attribute evaluation

- Impurity based
  - information theory based (information gain, gain ratio, distance measure, J-measure)
  - probability based: Gini index, DKM, classification error on the training set
  - MDL
  - statistics G, $\aleph^2$
  - mean squared and mean absolute error (MSE, MAE)
  - assume conditional independence (upon label) between the attributes
- Context sensitive measures:
  - Relief, Contextual Merit,
  - random forests or boosting based attribute evaluation,
  - affinity graph based

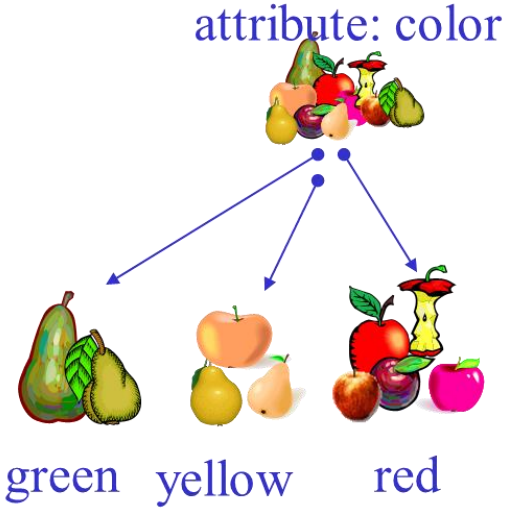
Information gain

- measure purity of labels before and after the split
- impurity = entropy

$$I(\tau) = -\sum_{i=1}^{c} p(\tau_i) \log_2 p(\tau_i)$$

$$I(\tau \mid A) = -\sum_{j=1}^{v_A} p(v_j) \sum_{i=1}^{c} p(\tau_i \mid v_j) \log_2 p(\tau_i \mid v_j)$$

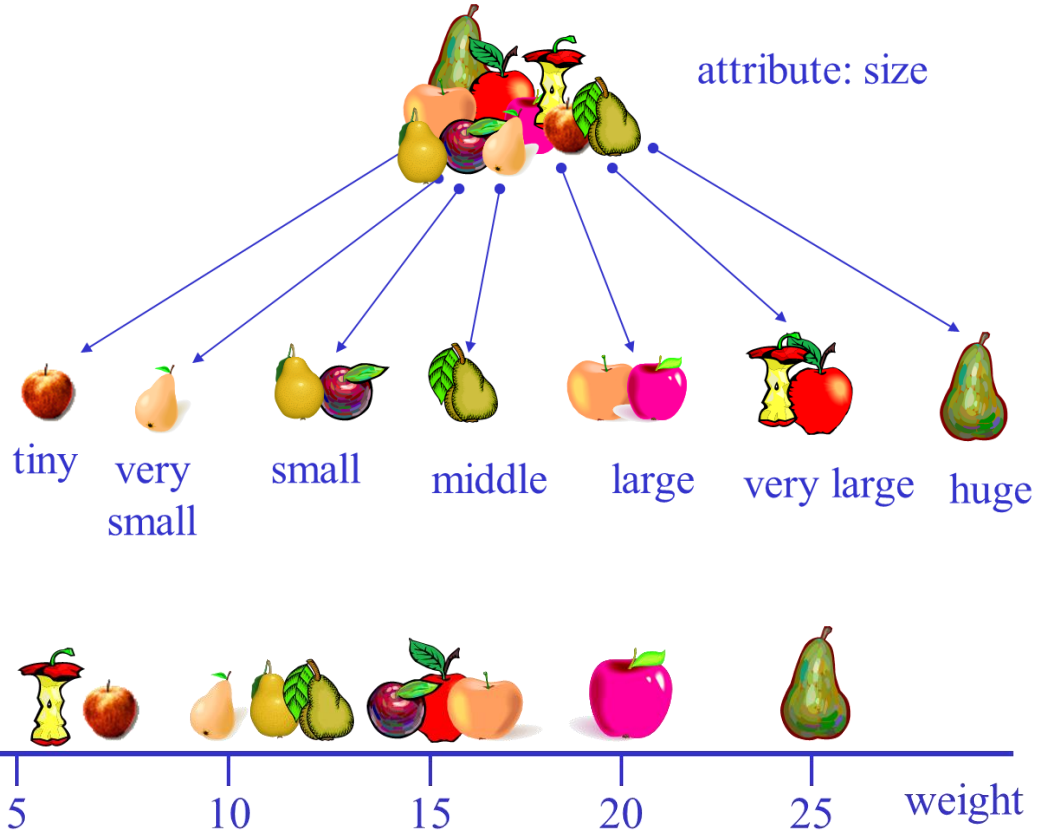
$$IG(A) = I(\tau) - I(\tau \mid A)$$



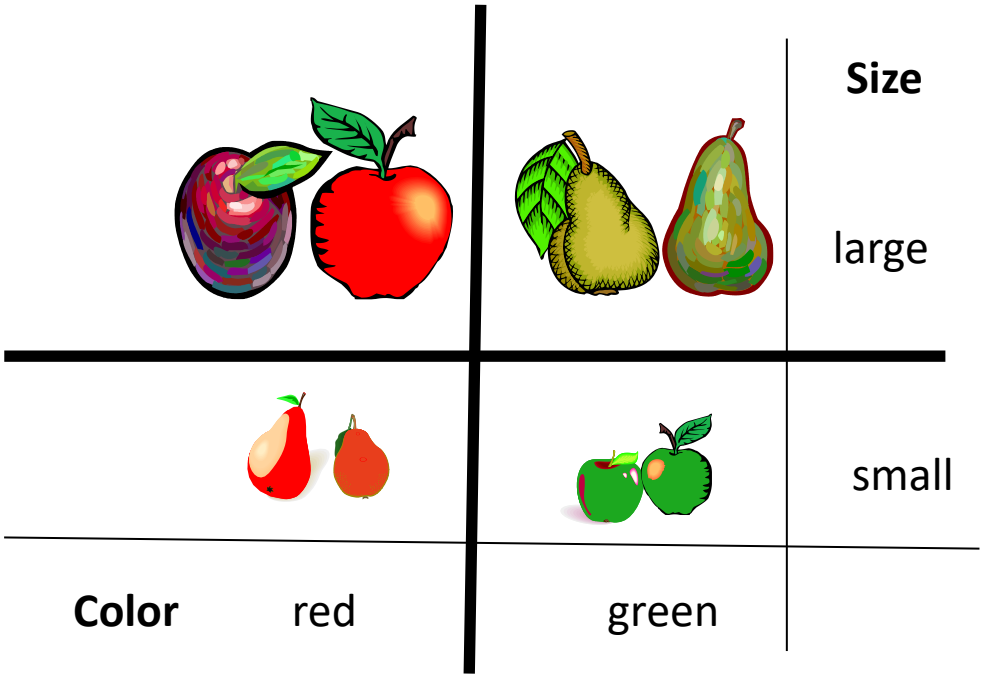
- each attribute is evaluated independently from others

Multivalued and numeric attributes

- multivalued: insufficient statistical support in certain splits
- numeric: sometimes requires prior discretization

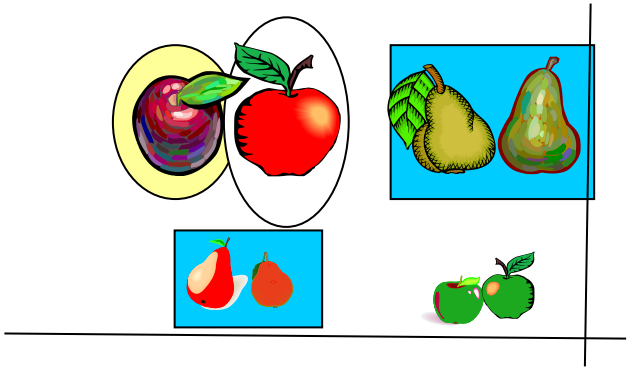


Attribute interactions



Relief algorithms

- criterion: evaluate attribute according to its power of separation between near instances



- values of good attribute should distinguish between near instances from different class and have similar values for near instances from the same class

Relief algorithms

- no assumption of conditional independence
- context sensitive
- reliable also in problems with strong conditional dependencies
- included in several machine learning systems (e.g., Weka, Orange, scikit-learn, R)
    - Relief (Kira in Rendell, 1992): two class classification
    - ReliefF (Kononenko, 1994): multi-class classification
    - RReliefF (Robnik Šikonja in Kononenko, 1997): regression

Marko Robnik-Šikonja, Igor Kononenko: Theoretical and Empirical Analysis of ReliefF and RReliefF. *Machine Learning Journal*, 53:23-69, 2003

Algorithm Relief

Input: set of instances $<x_i, \tau_i>$
Output: the vector W of attributes' evaluations

set all weights W[A] := 0.0;
for i := 1 **to** m **do begin**
  randomly select an instance R;
  find nearest hit H and nearest miss M;
  for A := 1 to #all_attributes **do**
    W[A] := W[A] **-** diff(A,R,H)/m + diff(A,R,M)/m;
end;

Function diff

- **for nominal attributes**

$$\mathrm{diff}(A, I_1, I_2) = \begin{cases} 0; \mathrm{value}(A, I_1) = \mathrm{value}(A, I_2) \\ 1; otherwise \end{cases}$$

- **for numerical attributes**

$$\mathrm{diff}(A, I_1, I_2) = \frac{\left|\mathrm{value}(A, I_1) - \mathrm{value}(A, I_2)\right|}{\max(A) - \min(A)}$$

- **distance between two instances**

$$\delta(\mathrm{I}_1, \mathrm{I}_2) = \sum_{i=1}^{a} \mathrm{diff}(A, I_1, I_2)$$

- **unknown values of attributes**

Extension ReliefF

- multi-class problems
- incomplete and noisy data
- robust
- uses k nearest instances from all the classes

The algorithm ReliefF

Input: set of instances $<x_i, \tau_i>$
Output: the vector W of attributes' evaluations

for v:=1 to a do W_v := 0.0;
for i := 1 **to** m **do begin**
  randomly select an instance R_i
  find k nearest hits H
  for each class t ≠ $R_{i,\tau}$ **do**
      from class t find k nearest misses M(t)
  for v := 1 to a **do**
    update W_v according to update formula
end;

Update formula

$$W_v = W_v - \frac{1}{m}\mathrm{con}(A_v, R_i, H) +$$

$$\frac{1}{m}\sum_{\substack{t=1 \\ t \neq R_{i,\tau}}}^{c} \frac{p(\tau_t)\,\mathrm{con}(A_v, R_i, M(t))}{1 - p(R_{i,\tau})}$$

$$\mathrm{con}(A_v, R_i, S) = \frac{1}{k}\sum_{j=1}^{k} \mathrm{diff}(A_v, R_i, S_j)$$

In regression: RReliefF

$$W[A] := W[A] - diff(A,R,H)/m + diff(A,R,M)/m;$$

$$W[A] = P(\text{different value of } A \mid \text{nearest instances with different prediction}) - P(\text{different value of } A \mid \text{nearest instances with same prediction})$$

$$W[A] = P_{dA|dC} - P_{dA|\neg dC}$$

- after applying the Bayesian rule: P(A|B) = P(A)P(B|A)/P(B)

$$W[A] = \quad = \frac{P_{dC|dA}P_{dA}}{P_{dC}} - \frac{(1-P_{dC|dA})P_{dA}}{1-P_{dC}}$$

- we approximate this formula
- unified view on attribute evaluation in classification and regression

Marko Robnik-Šikonja, Igor Kononenko: An adaptation of Relief for attribute estimation in regression. *Machine Learning, Proceedings of ICML 1997*

Feature selection: Embedded methods

Regularization for feature selection

- feature selection as part of learning (embedded method)
- loss function is composed of two components: prediction error and number/weight of included features

$$L(X, Y, f) = \sum_{i=1}^{n} I(y_i \neq f(x_i)) + \lambda \sum_{j=1}^{a} I(A_j \in X)$$

- in regression we get similar expressions for ridge regression and lasso

Ridge regression

- Ordinary Least Squares (OLS) estimates βs by minimizing

$$\mathrm{RSS} = \sum_{i=1}^{n} \left(y_i - \beta_0 - \sum_{j=1}^{p} \beta_j x_{ij} \right)^2 .$$

- Ridge regression minimizes a slightly different equation

$$\sum_{i=1}^{n} \left(y_i - \beta_0 - \sum_{j=1}^{p} \beta_j x_{ij} \right)^2 + \lambda \sum_{j=1}^{p} \beta_j^2 = \mathrm{RSS} + \lambda \sum_{j=1}^{p} \beta_j^2,$$

Ridge regression adds a penalty on βs !

- The effect of this equation is to add a penalty of the form

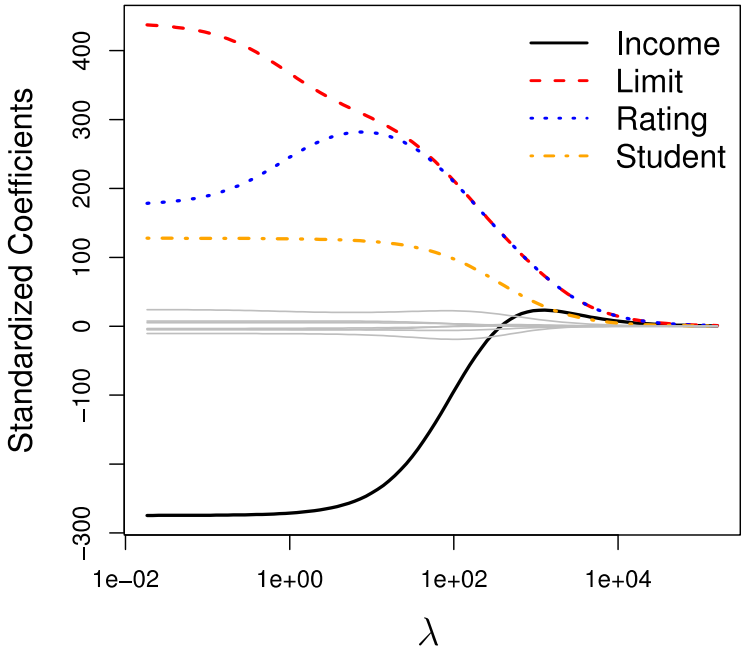
$$\lambda \sum_{j=1}^{p} \beta_j^2,$$

where the tuning parameter λ is a positive value.

- This has the effect of “shrinking” large values of βs towards zero.
- It turns out that such a constraint should improve the fit, because shrinking the coefficients can significantly reduce their variance
- Notice that when λ = 0, we get the OLS!

Credit data: ridge regression

- As λ increases, the standardized coefficients shrink towards zero.



Why can shrinking towards zero be a good thing?

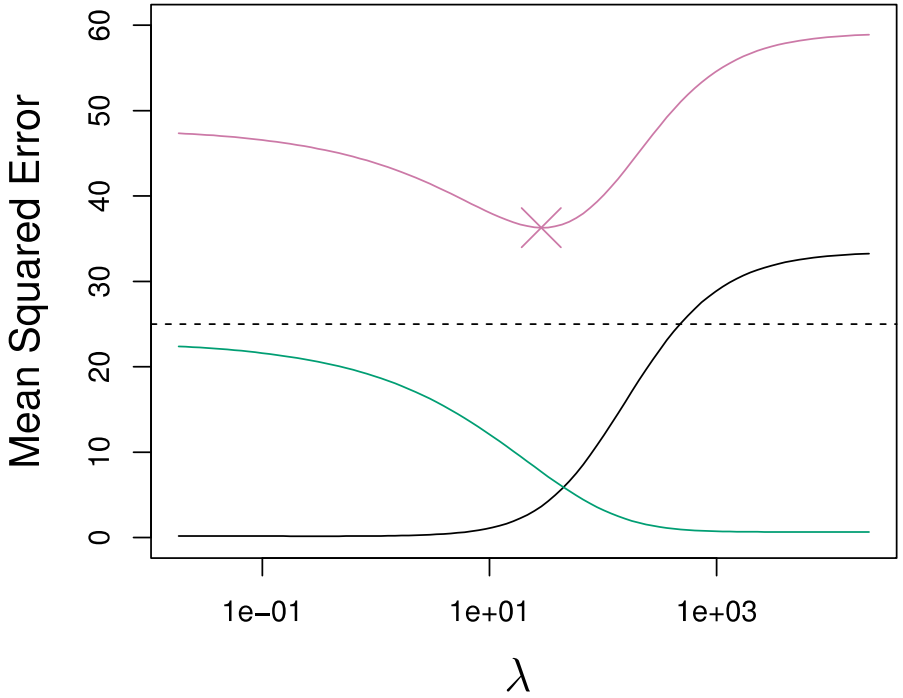
- It turns out that the OLS estimates generally have low bias but can be highly variable. In particular when n and p are of similar size or when $n < p$, then the OLS estimates will be extremely variable.

- The penalty term makes the ridge regression estimates biased but can also substantially reduce variance

- Thus, there is a bias/variance trade-off

Ridge regression bias / variance

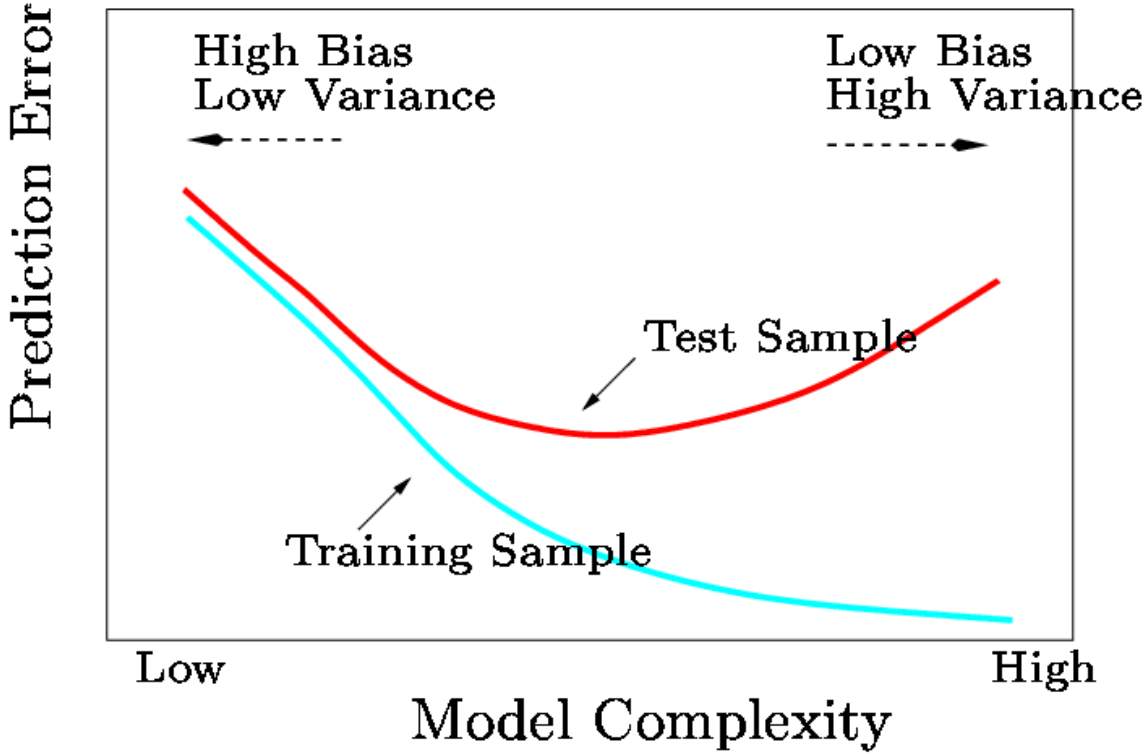
- Black: Bias
- Green: Variance
- Purple: MSE
- Increase of λ increases bias but decreases variance



Bias / variance trade-off

- In general, the ridge regression estimates will be more biased than the OLS ones but have lower variance

- Ridge regression will work best in situations where the OLS estimates have high variance



Computational advantages of ridge regression

- If number of features *p* is large, then using the best subset selection approach requires searching through enormous numbers of possible models

- With ridge regression, for any given λ, we only need to fit one model and the computations turn out to be very simple

- Ridge regression can even be used when $p > n$, a situation where OLS fails completely!

The LASSO method

- Ridge regression isn’t perfect
- One significant problem is that the penalty term will never force any of the coefficients to be exactly zero. Thus, the final model will include all variables, which makes it harder to interpret
- A more modern alternative is the LASSO
- The LASSO works in a similar way to ridge regression, except it uses a different penalty term

LASSO's Penalty Term

- Ridge Regression minimizes

$$\sum_{i=1}^{n}\left(y_i-\beta_0-\sum_{j=1}^{p}\beta_j x_{ij}\right)^2+\lambda\sum_{j=1}^{p}\beta_j^2=\mathrm{RSS}+\lambda\sum_{j=1}^{p}\beta_j^2,$$

- The LASSO estimates the βs by minimizing the

$$\sum_{i=1}^{n}\left(y_i-\beta_0-\sum_{j=1}^{p}\beta_j x_{ij}\right)^2+\lambda\sum_{j=1}^{p}|\beta_j|=\mathrm{RSS}+\lambda\sum_{j=1}^{p}|\beta_j|.$$

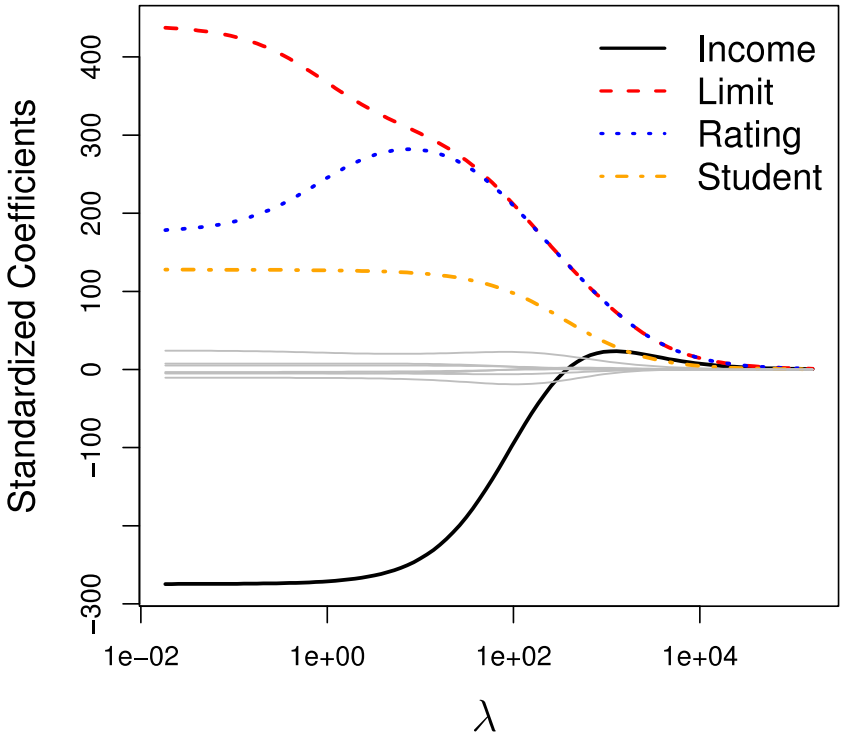
The difference between ridge regression and lasso

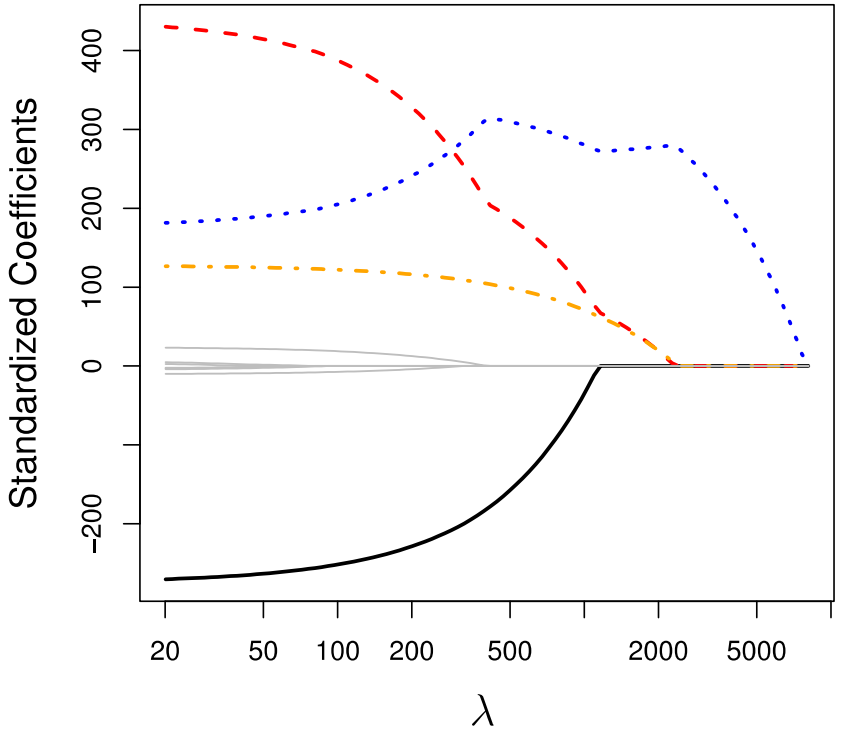
- This seems like a very similar idea but there is a big difference.

- Using LASSO penalty, it could be proven mathematically that some coefficients end up being set to exactly zero.

- With LASSO, we can produce a model that has high predictive power and it is simple to interpret.

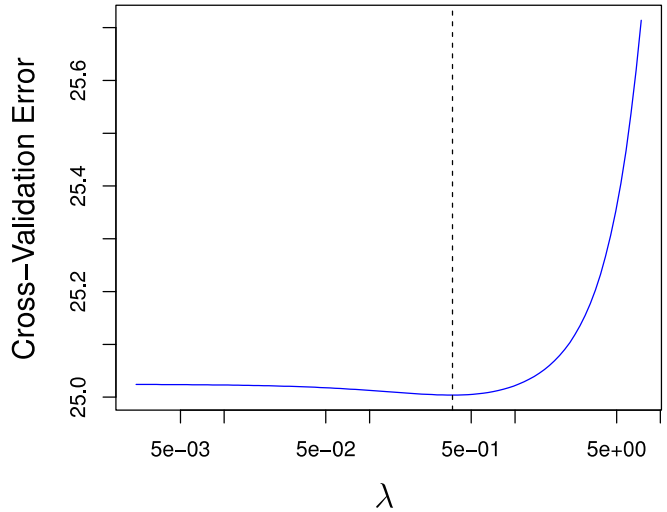
Credit data: Ridge and LASSO

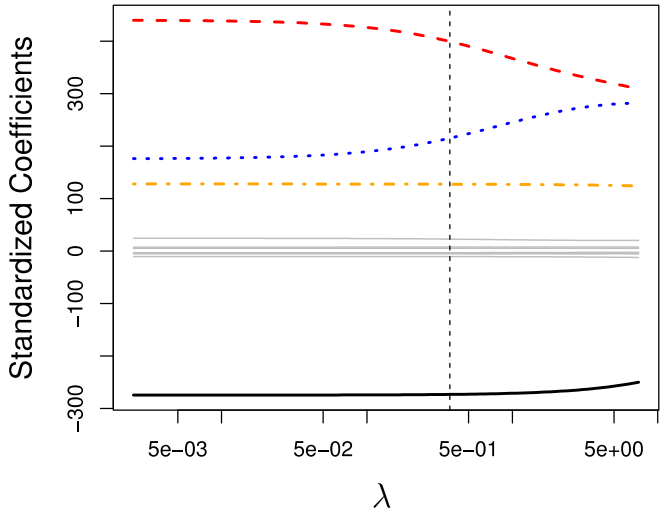




Selecting the tuning parameter λ

- We need to decide on a value for λ
- Select a grid of potential values, use cross validation to estimate the error rate on test data (for each value of λ) and select the value that gives the least error rate.





Feature selection: Wrapper methods

Wrapper approach

start with an empty set of features *S={}* // forward selection

repeat

add all unused features one by one to *S*

train a prediction model with each set *S*

evaluate each prediction model

keep the best added feature in *S*

until all features are added to *S*

return the best set of features encountered

- high computational load but effective for a given learning model; attention to data overfitting
- how would backward selection differ?

Model evaluation

Model evaluation metrics

- Evaluation metrics: How can we measure accuracy? Other metrics to consider?
- Regression: MSE, MAE
- Classification: accuracy, sensitivity, specificity, AUC, precision, recall
- Comparing classifiers:
  - Mean and confidence intervals
  - Cost-benefit analysis and ROC Curves
  - Rank-based tests (Friedman/Nemenyi)
  - Bayesian (hierarchical) tests

Classifier evaluation metrics: confusion matrix aka missclassification matrix

Confusion Matrix:

Actual class\Predicted class	C_1	¬ C_1
C_1	**True Positives (TP)**	**False Negatives (FN)**
¬ C_1	**False Positives (FP)**	**True Negatives (TN)**

Example of Confusion Matrix:

Actual class\Predicted class	buy_computer = yes	buy_computer = no	Total
buy_computer = yes	**6954**	**46**	7000
buy_computer = no	**412**	**2588**	3000
Total	7366	2634	10000

- Given *m* classes, an entry, $\boldsymbol{CM_{i,j}}$ in a confusion matrix indicates # of instances in class *i* that were labeled by the classifier as class *j*
- May have extra rows/columns to provide totals

Classification accuracy, error rate

A\P	C	¬C	
C	**TP**	**FN**	**P**
¬C	**FP**	**TN**	**N**
	P'	**N'**	**All**

- **Classifier Accuracy (CA),** or recognition rate: percentage of test set instances that are correctly classified

  Accuracy = (TP + TN)/All

- **Error rate:** *1 – accuracy*, or **Error rate = (FP + FN)/All**

Sensitivity and specificity

A\P	C	¬C	
C	**TP**	**FN**	**P**
¬C	**FP**	**TN**	**N**
	P'	**N'**	**All**

Class Imbalance Problem:

One class may be *rare*, e.g. fraud, or HIV-positive

Significant *majority of the negative class* and minority of the positive class

Sensitivity: True Positive recognition rate

Sensitivity = TP/P

Specificity: True Negative recognition rate

Specificity = TN/N

Precision, recall and F-measures

- **Precision**: exactness, i.e what % of instances the classifier labeled as positive are actually positive
  Precision = TP/P'

$$precision = \frac{TP}{TP + FP}$$

- **Recall:** completeness, i.e what % of positive instances did the classifier label as positive?
  Recall = TP / P (the same as sensitivity)

$$recall = \frac{TP}{TP + FN}$$

- Perfect score is 1.0
- Inverse relationship between precision & recall
- ***F* measure (*F_1* or *F*-score)**: harmonic mean of precision and recall,

$$F = \frac{2 \times precision \times recall}{precision + recall}$$

- ***F_β*:** weighted measure of precision and recall
  - assigns ß times as much weight to recall as to precision

$$F_\beta = \frac{(1 + \beta^2) \times precision \times recall}{\beta^2 \times precision + recall}$$

Example: precision and recall

Actual Class\Predicted class	cancer = yes	cancer = no	Total	Recognition(%)
cancer = yes	**90**	**210**	300	30.00 (*sensitivity)*
cancer = no	**140**	**9560**	9700	98.56 (*specificity)*
Total	230	9770	10000	96.40 (*accuracy*)

- *Precision* = 90/230 = 39.13% *Recall* = 90/300 = 30.00%

Multiclass evaluation

- no problems for classification accuracy
- most other measures assume binary class, e.g., precision, recall, F_1
- multiclass extensions simulate binary case
- macro average:
  - compute several one-versus-all scores and average
  - assumes balanced class distribution, gives equal weight to each class
- micro average
  - computes TP, FP, TN, FN for each class separately and then compute the measure
  - assumes all instances are of the same importance, in case of imbalanced classes this might be problematic

Multiclass example

- Let us compute precision $P = TP / (TP+FP)$.
- Let us assume multi-class classification system with four classes and the following numbers when tested:
- Class A: 1 TP and 1 FP
- Class B: 10 TP and 90 FP
- Class C: 1 TP and 1 FP
- Class D: 1 TP and 1 FP
- $P(A) = P(C) = P(D) = 0.5$, whereas $P(B)=0.1$.
- A macro-averaged precision: $P_{macro} = (0.5+0.1+0.5+0.5) / 4 = 0.4$
- A micro-averaged precision: $P_{micro} = (1+10+1+1) / (2+100+2+2) = 0.123$

Error depends on decision threshold

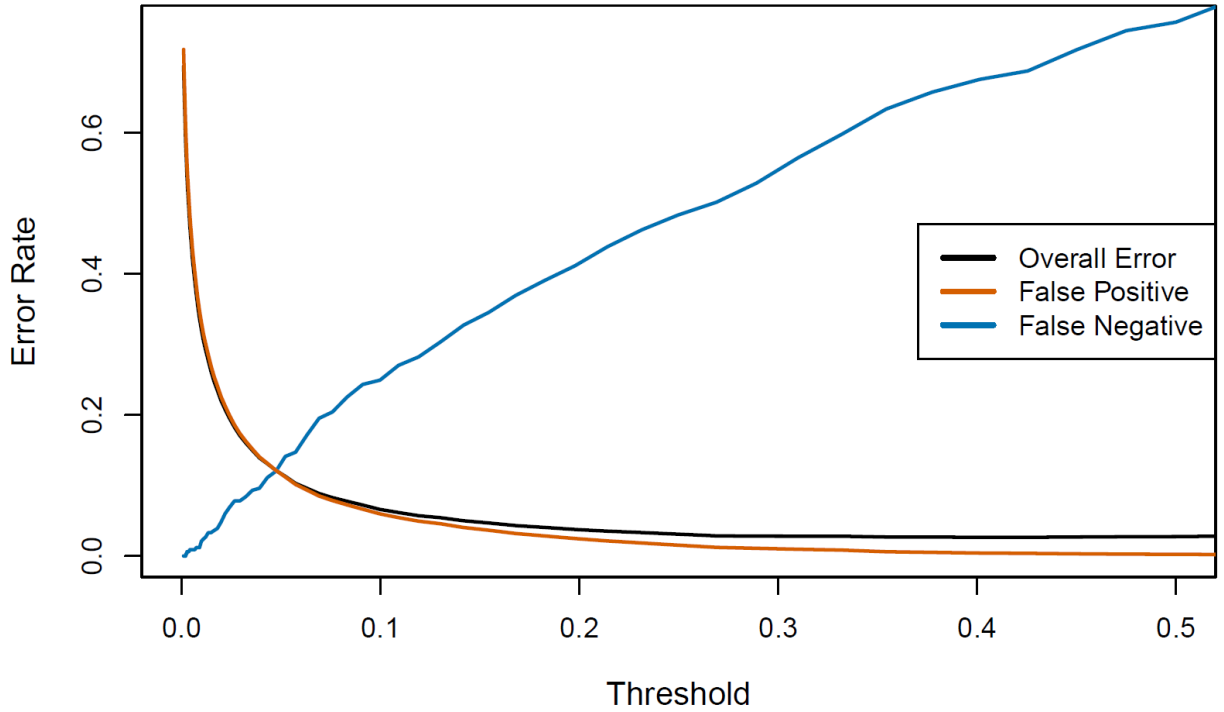
- Example: False positive and false negative rate are computed based on probabilities returned by classifier

$$P(\text{Class=True} \mid X_1, X_2, \ldots) \geq 0.5$$

- We can change the two error rates by changing the threshold from 0.5 to some other value in [0, 1]:

$$P(\text{Class=True} \mid X_1, X_2, \ldots.) \geq \text{threshold}$$

Varying the threshold

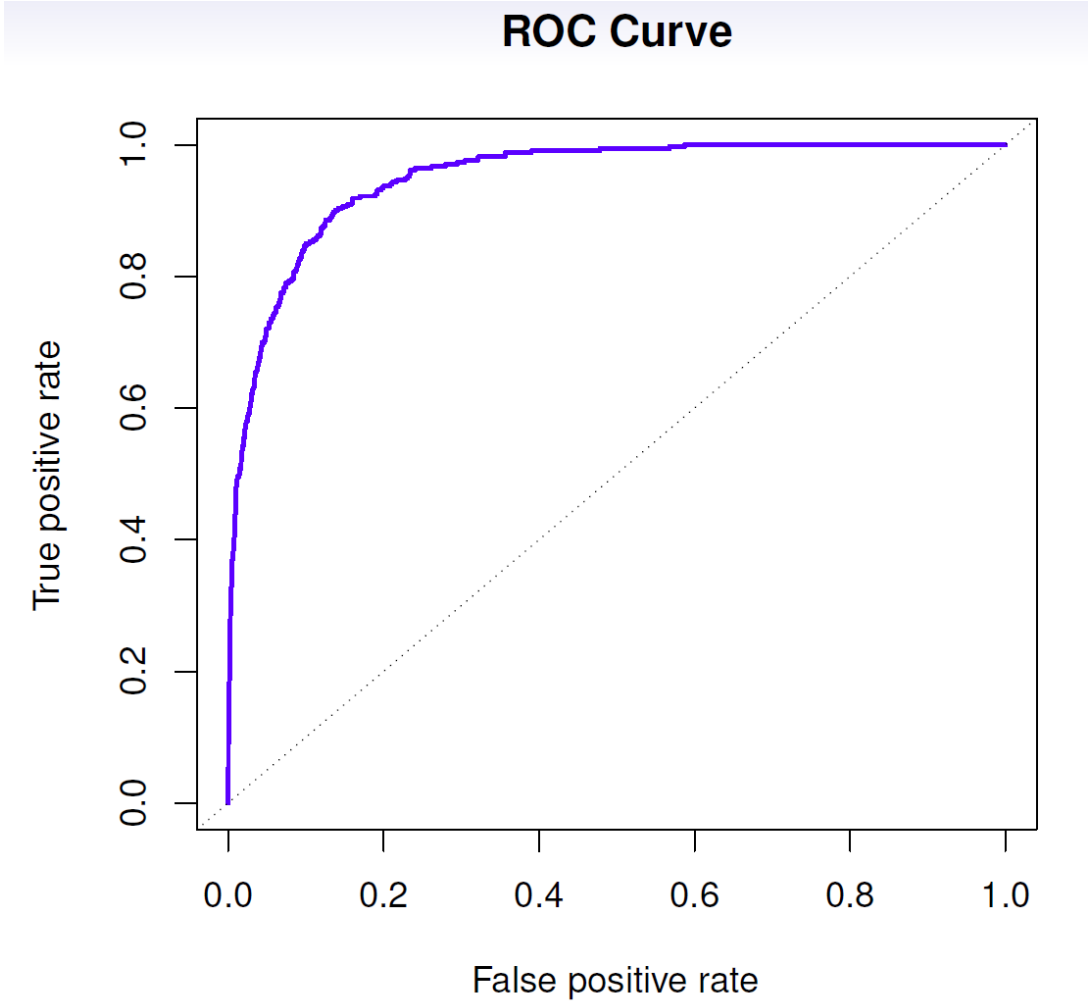


- To reduce false negative rate, we would chose threshold other than 0.5, e.g., threshold ≤ 0.1

ROC curve

- ROC curve shows both TP rate and FP rate simultaneously

- To summarize overall performance, we also use area under the ROC curve (AUC)
- The larger the AUC the better is the classifier. Why? What would be an ideal ROC curve?



Issues affecting model selection

- **Accuracy**
  - classification: classification accuracy, AUC, F_1
  - regression: MSE, MAE
- **Speed**
  - time to construct the model (training time)
  - time to use the model (classification/prediction time)
- **Robustness**: handling noise and missing values
- **Scalability**: efficiency in disk-resident databases
- **Interpretability**
  - understanding and insight provided by the model
  - other measures, e.g., goodness of rules, such as decision tree size or compactness of classification rules

Unsupervised feature selection

- criterion: preserve similarity between instances
- Example: SPEC, spectral feature selection

Zhao Z, Liu H. Spectral feature selection for supervised and unsupervised learning. In Proceedings of ICML 2007, pp. 1151-1157.

Semi-supervised feature selection

- typically a small sample of labelled and a large sample of unlabeled data is available
- principle: use the label information of labeled data and data distribution or local structure of both labeled and unlabeled data to evaluate feature relevance

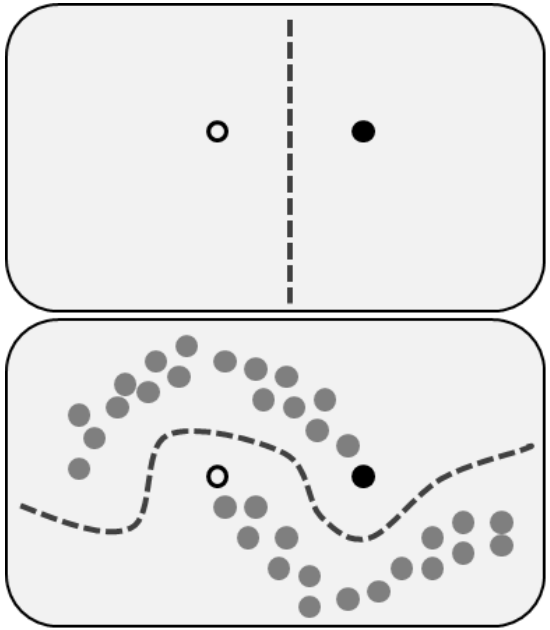
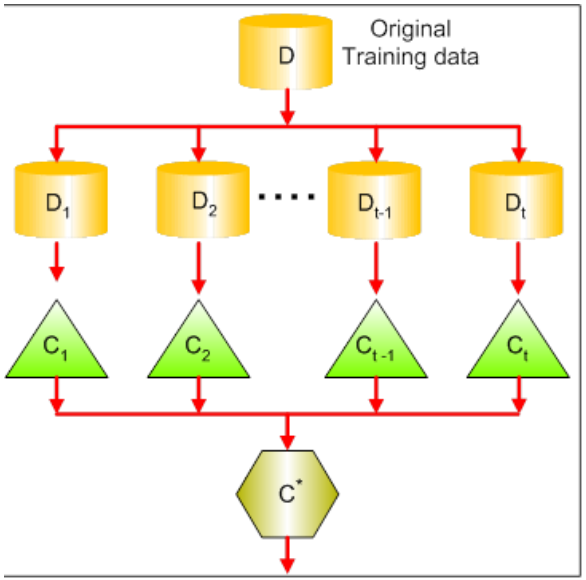


image by Techerin, Wikipedia

Cheng, H., Deng, W., Fu, C., Wang, Y. and Qin, Z., 2011. Graph-based semi-supervised feature selection with application to automatic spam image identification. In Computer Science for Environmental Engineering and EcoInformatics (pp. 259-264).

Stability of feature selection

- for high dimensional small sample data, the stability of feature selection is a pressing issue, e.g., in microarray data, we might get similar classification accuracy with different sets of features
- Solution: **ensemble approach:**
  1. produce diverse feature sets
     - different feature selection techniques,
     - instance-level perturbation
     - feature-level perturbation
     - stochasticity in the feature selector,
     - Bayesian model averaging
     - combinations of the above techniques
  2. aggregate them
     - weighted voting
     - counting



Dimensionality reduction

Feature Selection vs Feature Extraction

- Feature selection: Choosing $k<d$ important features, ignoring the remaining $d - k$

  Subset selection algorithms

- Feature extraction: Project the

  original x_i , i =1,...,d dimensions to

  new $k<d$ dimensions, z_j , j =1,...,k

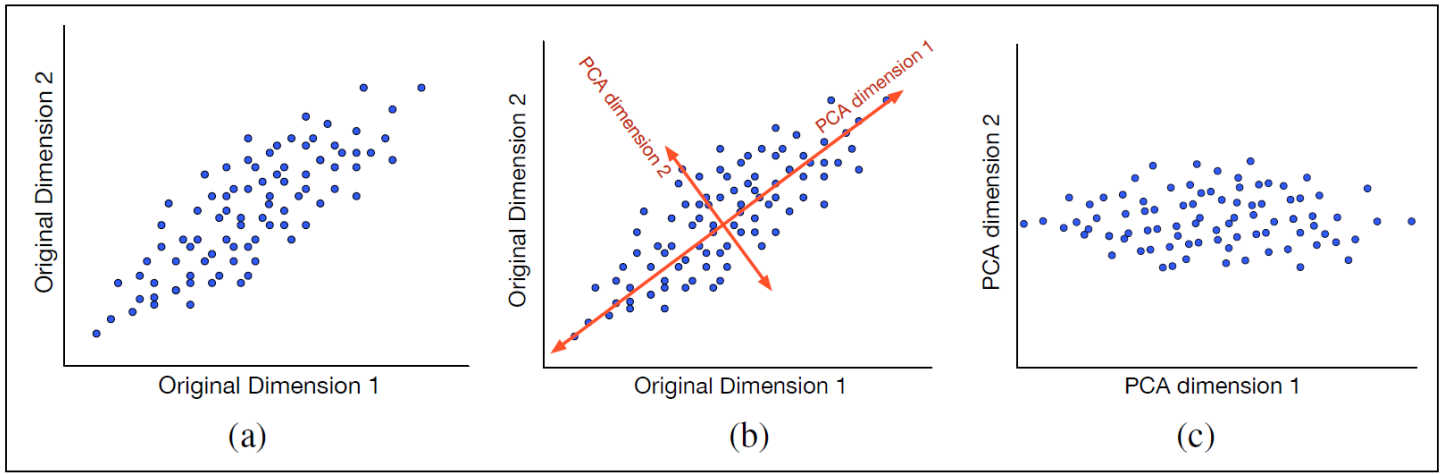
- Typical examples: Principal components analysis (PCA), linear discriminant analysis (LDA), factor analysis (FA)

Feature reduction

- approximation of p-dimensional space of matrix X with lower dimensional space
- also called feature extraction
- Linear transformation: rotation in the direction of largest variance

Principle components analysis

- principle components analysis, PCA
- we iteratively find the orthogonal axes of the largest variance
- we use the new dimensions to approximate the original space



Principal Components Analysis (PCA)

- Find a low-dimensional space such that when $\mathbf{x}$ is projected there, information loss is minimized.
- The projection of $\mathbf{x}$ on the direction of $\mathbf{w}$ is: $z = \mathbf{w}^T\mathbf{x}$
- Find $\mathbf{w}$ such that Var(z) is maximized

$$\text{Var}(z) = \text{Var}(\mathbf{w}^T\mathbf{x}) = E[(\mathbf{w}^T\mathbf{x} - \mathbf{w}^T\boldsymbol{\mu})^2]$$
$$= E[(\mathbf{w}^T\mathbf{x} - \mathbf{w}^T\boldsymbol{\mu})(\mathbf{w}^T\mathbf{x} - \mathbf{w}^T\boldsymbol{\mu})]$$
$$= E[\mathbf{w}^T(\mathbf{x} - \boldsymbol{\mu})(\mathbf{x} - \boldsymbol{\mu})^T\mathbf{w}]$$
$$= \mathbf{w}^T E[(\mathbf{x} - \boldsymbol{\mu})(\mathbf{x} - \boldsymbol{\mu})^T]\mathbf{w} = \mathbf{w}^T \sum \mathbf{w}$$

where $\text{Var}(\mathbf{x}) = E[(\mathbf{x} - \boldsymbol{\mu})(\mathbf{x} - \boldsymbol{\mu})^T] = \sum$

- Maximize Var(z) subject to ||$\boldsymbol{w}$||=1

$$\max_{\mathbf{w}_1} \mathbf{w}_1^T \Sigma \mathbf{w}_1 - \alpha\left(\mathbf{w}_1^T \mathbf{w}_1 - 1\right)$$

$\sum \boldsymbol{w}_1 = a\boldsymbol{w}_1$ that is, $\boldsymbol{w}_1$ is an eigenvector of $\sum$

Choose the one with the largest eigenvalue for Var(z) to be max

- Second principal component: Max Var(z_2), s.t., ||$\boldsymbol{w}_2$||=1 and orthogonal to $\boldsymbol{w}_1$

$$\max_{\mathbf{w}_2} \mathbf{w}_2^T \Sigma \mathbf{w}_2 - \alpha\left(\mathbf{w}_2^T \mathbf{w}_2 - 1\right) - \beta\left(\mathbf{w}_2^T \mathbf{w}_1 - 0\right)$$

$\sum \boldsymbol{w}_2 = a\,\boldsymbol{w}_2$ that is, $\boldsymbol{w}_2$ is another eigenvector of $\sum$

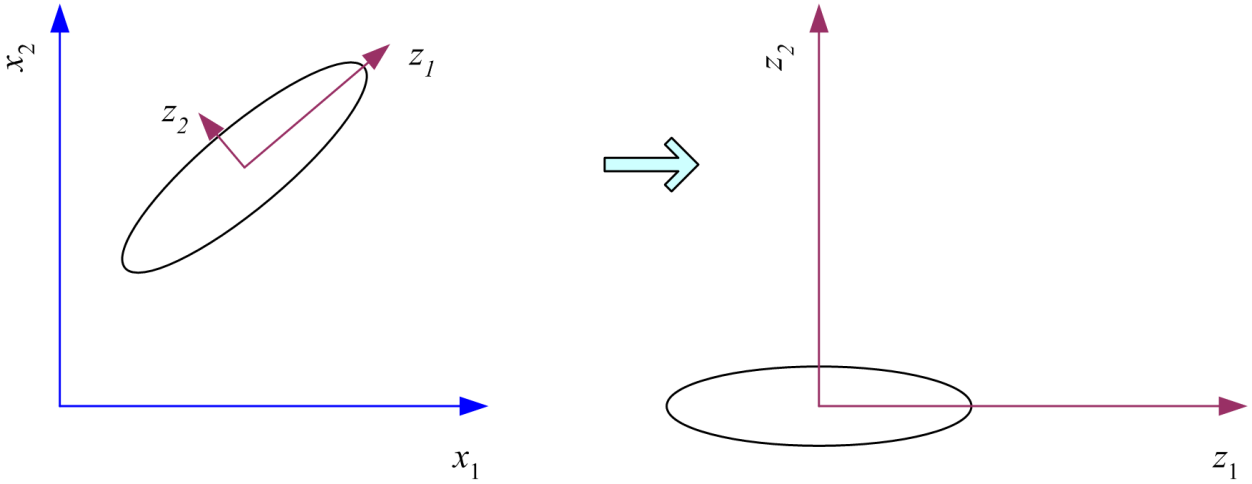
and so on.

What PCA does

$$\mathbf{z} = \mathbf{W}^T(\mathbf{x} - \boldsymbol{m})$$

where the columns of **W** are the eigenvectors of ∑, and ***m*** is sample mean

Centers the data at the origin and rotates the axes

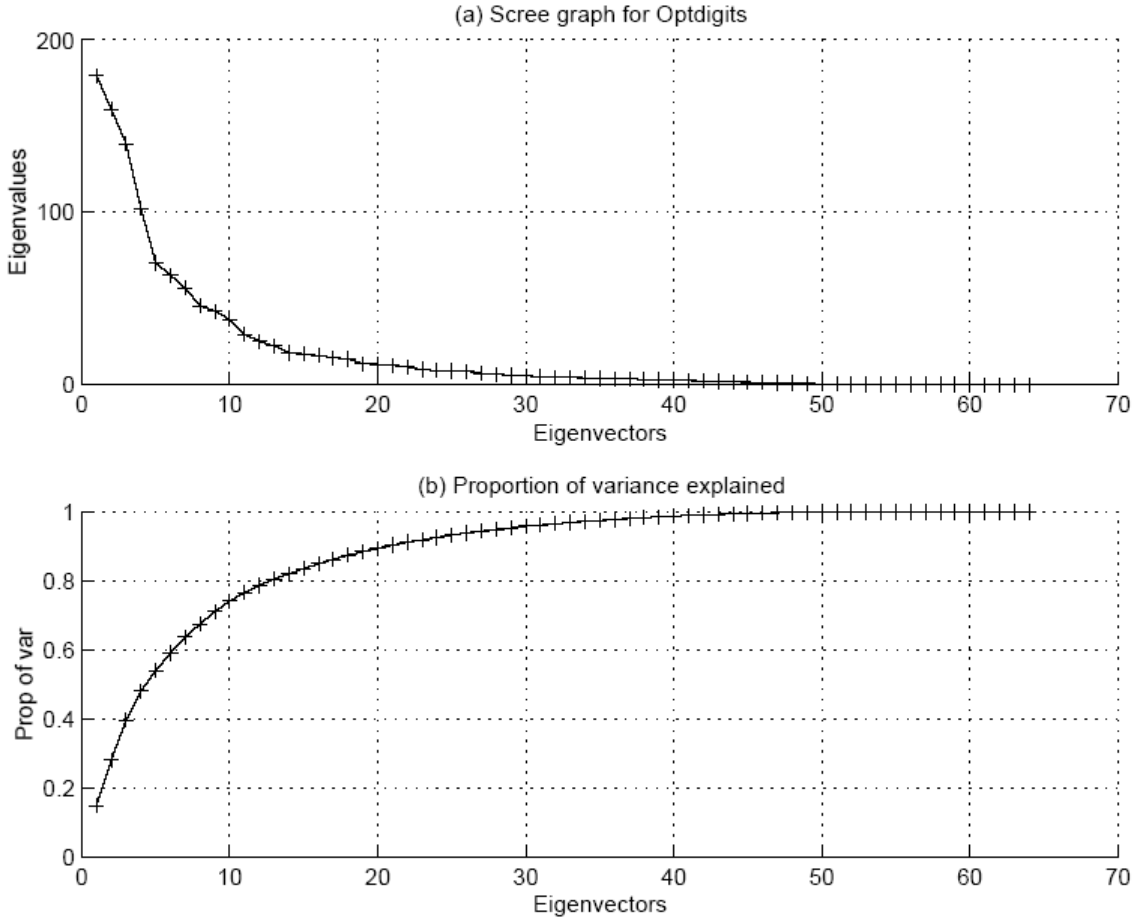


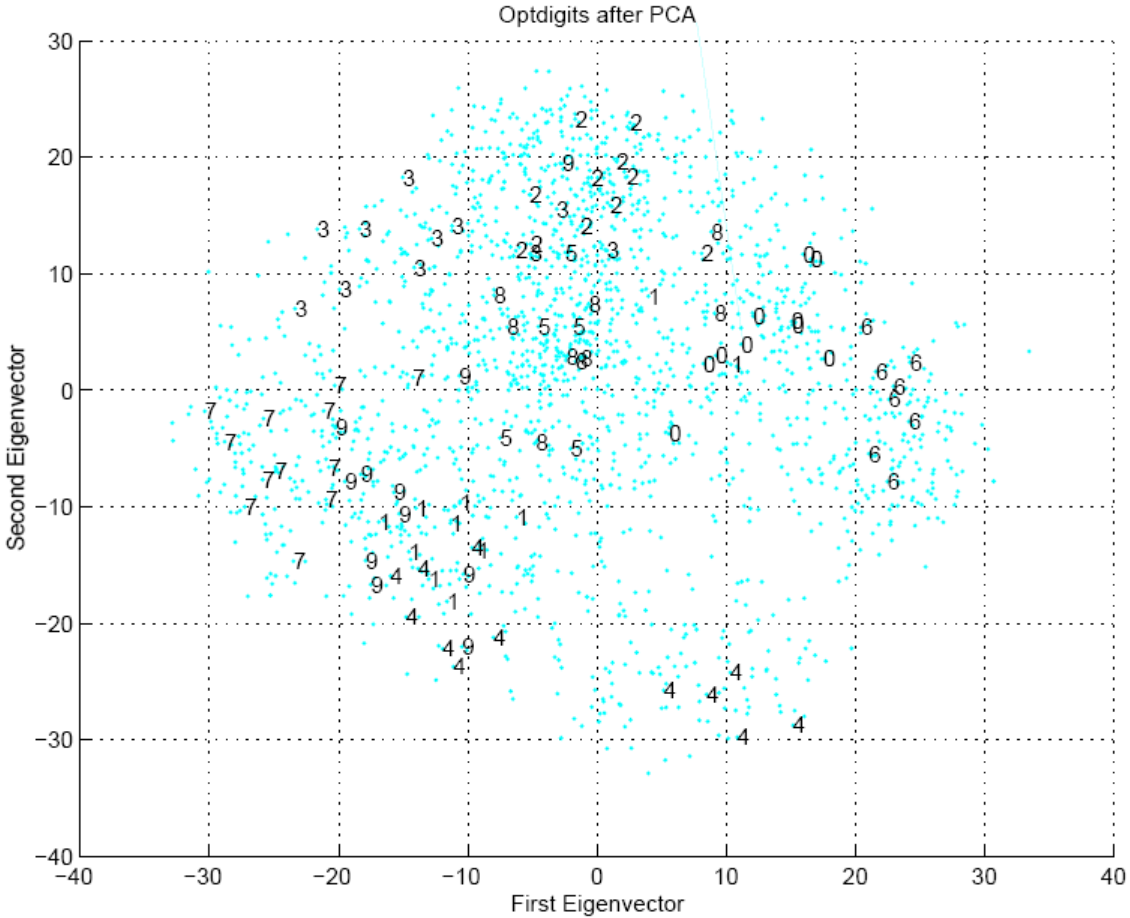
How to choose k ?

- Proportion of Variance (PoV) explained

$$\frac{\lambda_1 + \lambda_2 + \cdots + \lambda_k}{\lambda_1 + \lambda_2 + \cdots + \lambda_k + \cdots + \lambda_d}$$

  when λ_i are sorted in descending order
- Typically, stop at PoV>0.9
- Scree graph plots of PoV vs *k*, stop at "elbow"


(a) Scree graph for Optdigits
Eigenvalues
Eigenvectors
0
100
200
0
10
20
30
40
50
60
70
(b) Proportion of variance explained
Prop of var
Eigenvectors
0
0.2
0.4
0.6
0.8
1
0
10
20
30
40
50
60
70


Optdigits after PCA
Second Eigenvector
First Eigenvector
30
20
10
0
−10
−20
−30
−40
−40
−30
−20
−10
0
10
20
30
40

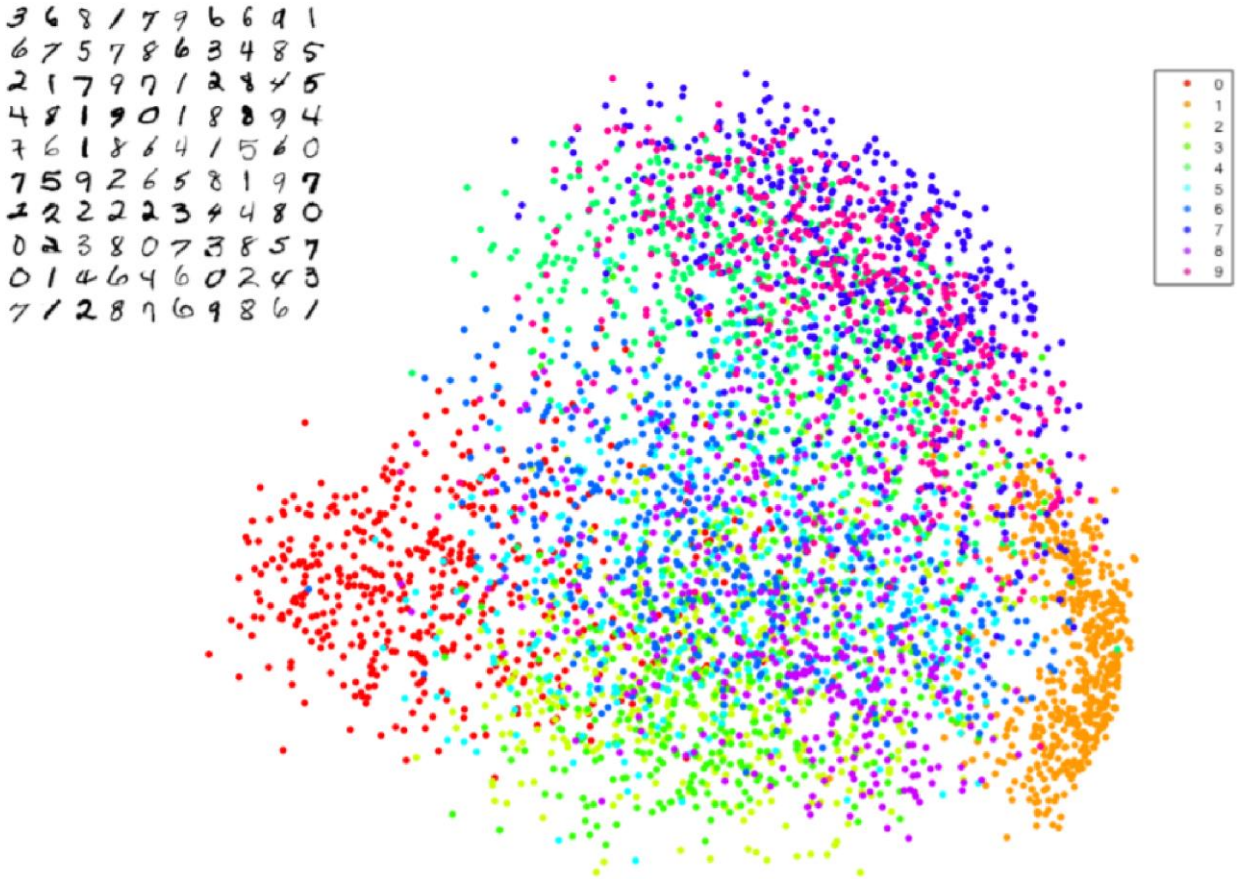
Neighbourhood preserving dimensionality reduction

- also called local embeddings
- SNE - Stochastic Neighbor Embedding
- t-SNE (t-distributed SNE)

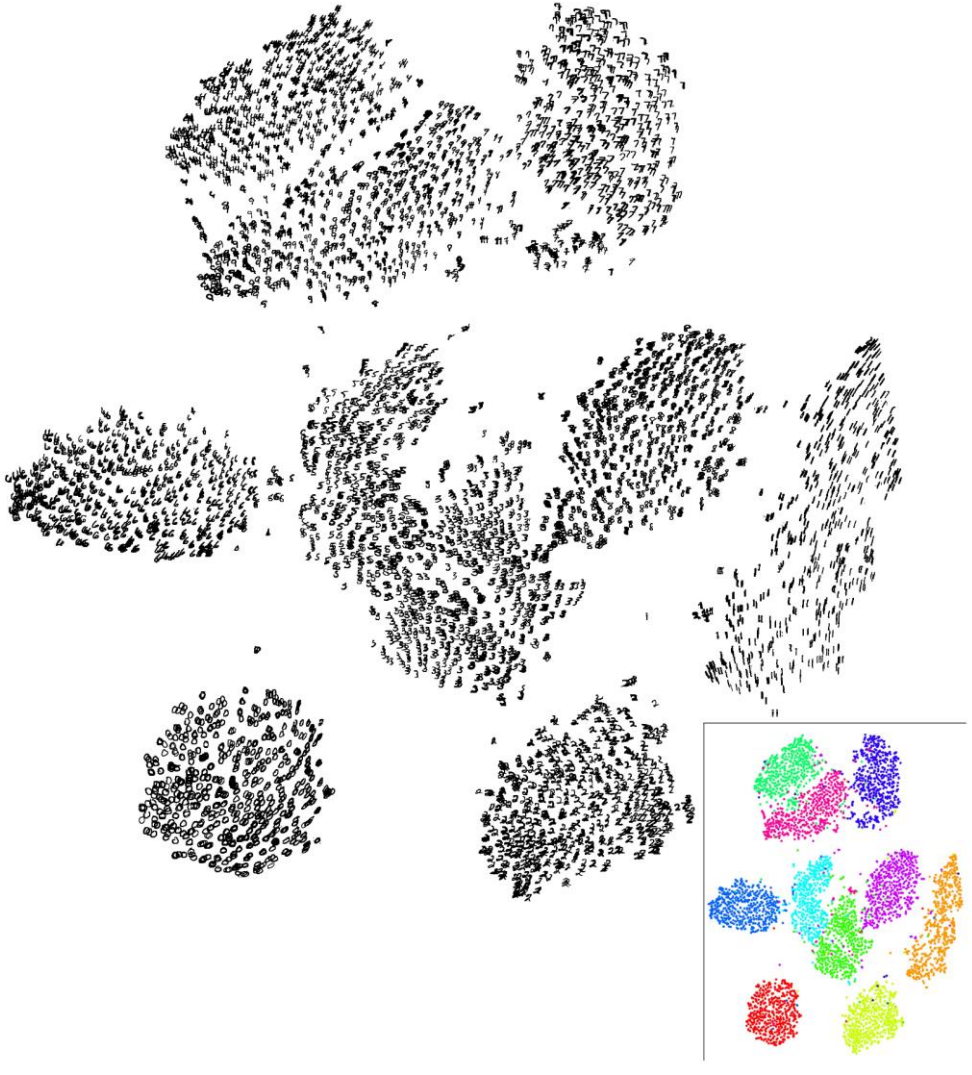
Linear and local embedding

- PCA tries to find a global structure
  - Low dimensional subspace
  - Can lead to local inconsistencies
  - Far away point can become nearest neighbors
- t-SNE is an alternative dimensionality reduction algorithm.
- t-SNE tries to preserve local structure
  - Low dimensional neighborhood should be the same as the original neighborhood.
  - Unlike PCA almost only used for visualization
  - No easy way to embed new points

PCA 2d projection of MNIST dataset



t-SNE 2d projection of MNIST dataset



Stochastic Neighbor Embedding (SNE)

- SNE basic idea:
  - ”Encode” high-dimensional neighborhood information as a distribution
  - Intuition: Random walk between data points.
    - High probability of jumping to a close point
- Find low dimensional points such that their neighborhood distribution is similar.
- How do you measure the distance between distributions?
- Most common measure: KL divergence

Neighborhood Distributions

- Consider the neighborhood around an input data point $\mathbf{x}_i \in R^d$
- Imagine that we have a Gaussian distribution centered around $\mathbf{x}_i$
- Then the probability that $\mathbf{x}_i$ chooses some other data point $\mathbf{x}_j$ as its neighbor is in proportion with the density under this Gaussian
- A point closer to $\mathbf{x}_i$ will be more likely than one further away

SNE objective

- Given $\mathbf{x_1}, .., \mathbf{x_n} \in R^D$ we define the distribution of distances between points P_{ij}
- Goal: Find good embedding $\mathbf{y_1}, .., \mathbf{y_n} \in R^d$ for some $d < D$ (normally 2 or 3)
- How do we measure an embedding quality?
- For points $\mathbf{y_1}, .., \mathbf{y_n} \in R^d$ we can define distribution Q_{ij} similarly to P_{ij}

$$Q_{ij} = \frac{\exp\left(-||\mathbf{y}^{(i)} - \mathbf{y}^{(j)}||^2\right)}{\sum_k \sum_{l \neq k} \exp\left(-||\mathbf{y}^{(l)} - \mathbf{y}^{(k)}||^2\right)}$$

- Optimize Q to be close to P
  - Minimize KL-divergence
- The embeddings $\mathbf{y_1}, .., \mathbf{y_n} \in R^d$ are the parameters we are optimizing.
  - How do you embed a new point? No embedding function, but there are ways.

Kullback–Leibler divergence

- KL divergence measures distance between two distributions, P and Q:

$$KL(Q||P) = \sum_{ij} Q_{ij} \log \left(\frac{Q_{ij}}{P_{ij}} \right)$$

- Not a metric function - not symmetric!
- Code theory intuition: If we are transmitting information that is distributed according to P, then the optimal (lossless) compression will need to send on average H(P) bits.
- What happens if you expect P (and design your compression accordingly), but the actual distribution is Q?
  - will send on average H(Q) + KL(Q||P)
  - KL(Q||P) is the ”penalty” for using the wrong distribution

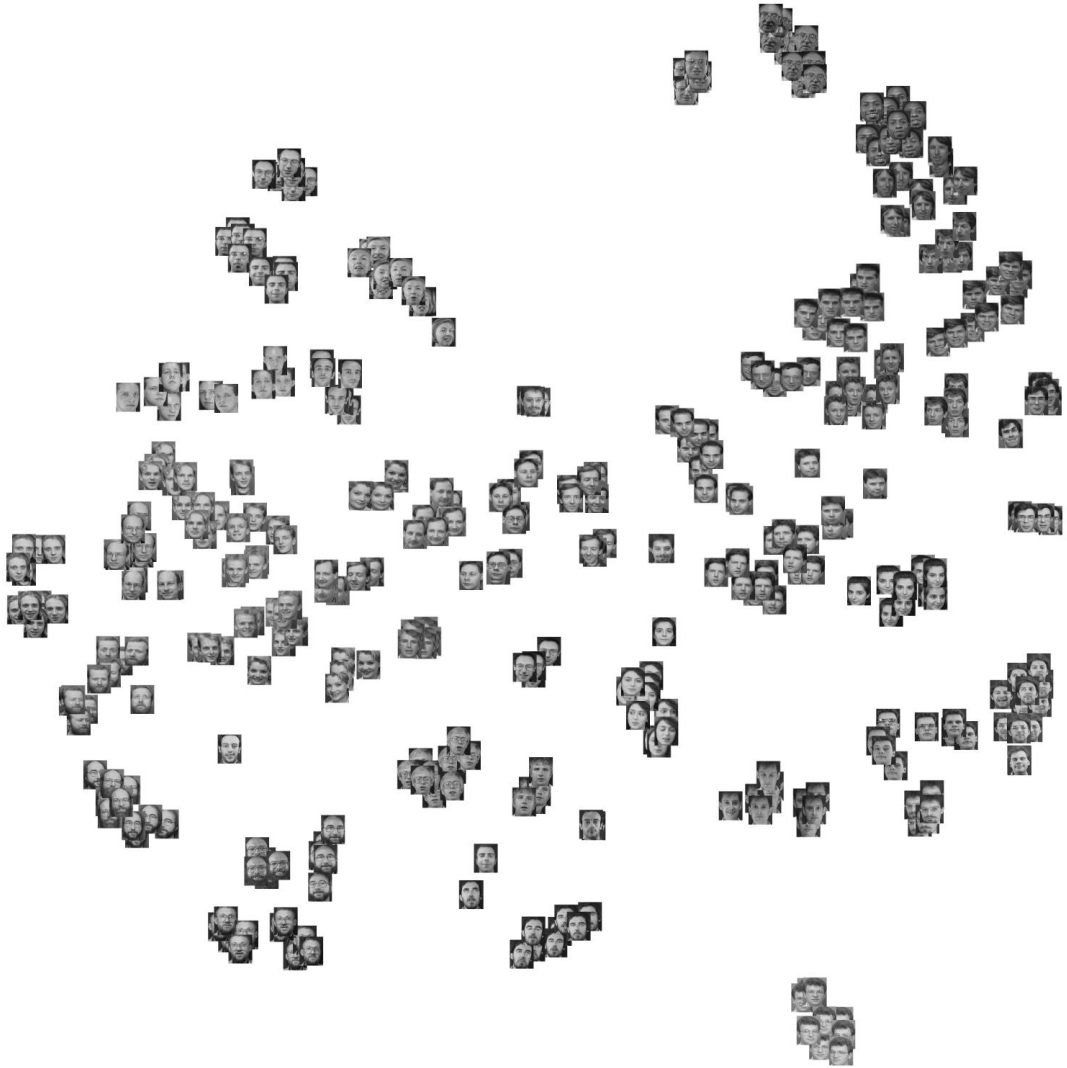
Crowding Problem

- In high dimension we have more room, points can have a lot of different neighbors
- In 2D a point can have a few neighbors at distance one all far from each other - what happens when we embed in 1D?
- This is the ”crowding problem” - we don’t have enough room to accommodate all neighbors.
- This is one of the biggest problems with SNE.
- t-SNE solution: Change the Gaussian in Q to a heavy-tailed distribution.
  - if Q changes slower, we have more ”wiggle room” to place points at.

CNN features t-SNE 2d embedding

http://cs.stanford.edu/people/karpathy/cnnembed

CNN features t-SNE 2d embedding



https://lvdmaaten.github.io/tsne

t-SNE summary

- t-SNE is a great way to visualize high-dimensional data
- Helps understand ”black-box” algorithms like DNN.
- Reduced ”crowding problem” with heavy-tailed distribution.
- Non-convex optimization - solved by gradient descent (GD) with momentum.

- Maaten, L.v.d. and Hinton, G., (2008) Visualizing data using t-SNE. Journal of Machine Learning Research, Vol 9(Nov), pp. 2579—2605, [PDF]
- Wattenberg, M., Viégas, F. and Johnson, I. (2016) How to Use t-SNE Effectively, Distil https://distill.pub/2016/misread-tsne/
- Poličar, P. OpenTSNE, https://github.com/pavlin-policar/openTSNE