Machine Learning: Think Big and Parallel

Day 1

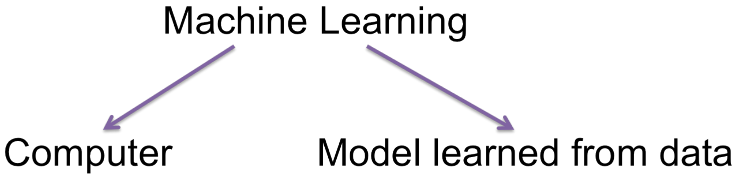
Inderjit S. Dhillon
Dept of Computer Science
UT Austin

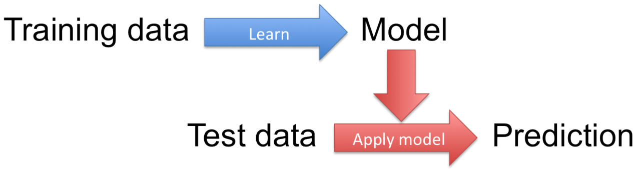
CS395T: Topics in Multicore Programming
Oct 1, 2013

Outline

- Scikit-learn: Machine Learning in Python
- Supervised Learning — day1
  - Regression: Least Squares, Lasso
  - Classification: kNN, SVM
- Unsupervised Learning — day2
  - Clustering: k-means, Spectral Clustering
  - Dimensionality Reduction: PCA, Matrix Factorization for Recommender Systems

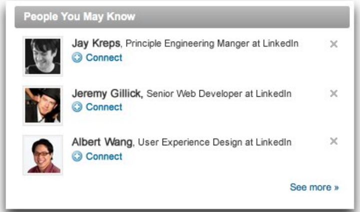
What is Machine Learning?





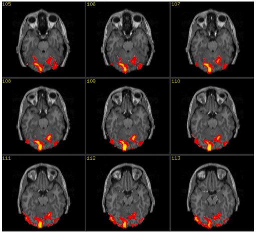
Machine Learning Applications

Link prediction

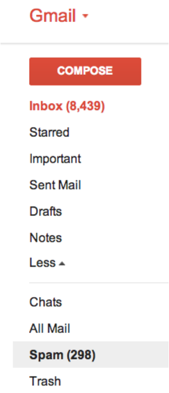


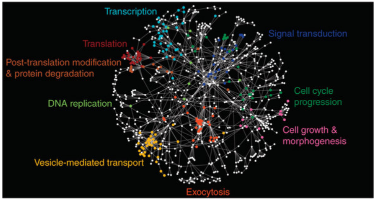
LinkedIn.

fMRI



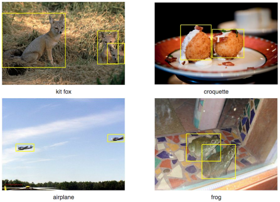
Spam classification





gene-gene network

Image classification



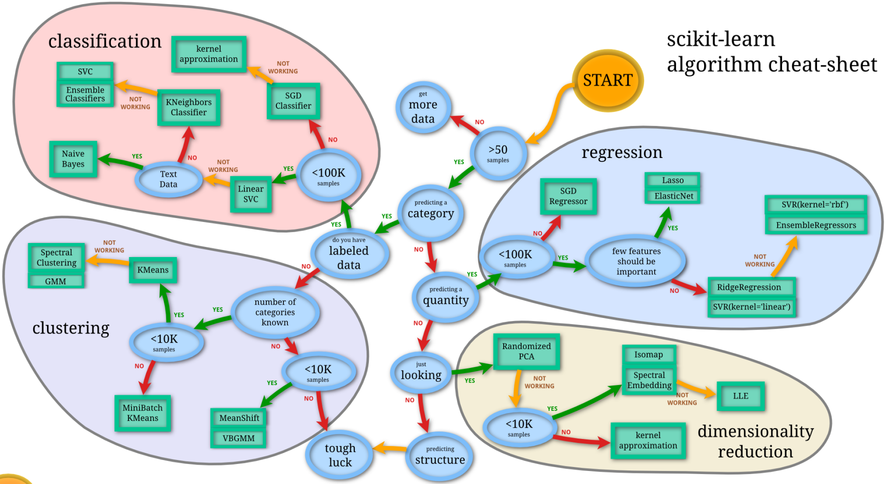
Scikit-learn: Machine Learning in Python

- Open Source with BSD Licence
  - `http://scikit-learn.org/`
  - `https://github.com/scikit-learn/scikit-learn`
- Built on efficient libraries
  - Python numerical library (numpy)
  - Python scientific library (scipy)
- Active development
  - A new release every 3 month
  - 183 contributors on the current release

Scikit-learn: What it includes

- Supervised Learning
  - Regression: Ridge Regression, Lasso, SVR, etc
  - Classification: kNN, SVM, Naive Bayes, Random Forest, etc
- Unsupervised Learning
  - Clustering: k-means, Spectral Clustering, Mean-Shift, etc
  - Dimension Reduction: (kernel/sparse) PCA, ICA, NMF, etc
- Model Selection
  - Cross-validation
  - Grid Search for parameters
  - Various metrics
- Preprocessing Tool
  - Feature extraction, such as TF-IDF
  - Feature standardization, such as mean removal and variance scaling
  - Feature binarization
  - Categorical feature encoding

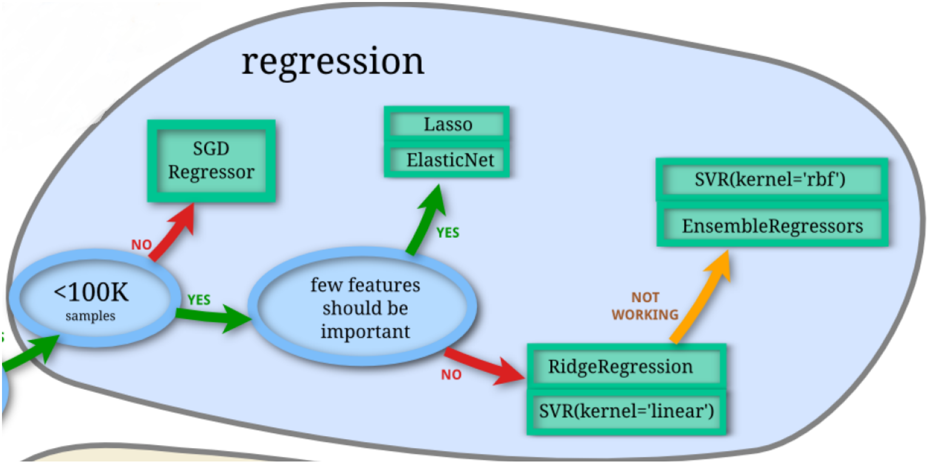
Scikit-learn Cheat Sheet



Back

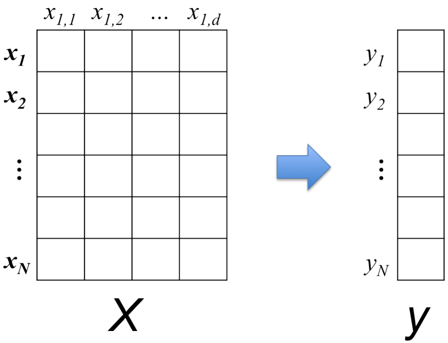
Regression

Regression



regression
SGD Regressor
Lasso
ElasticNet
SVR(kernel='rbf')
EnsembleRegressors
NO
YES
<100K samples
YES
few features should be important
NOT WORKING
NO
RidgeRegression
SVR(kernel='linear')

Regression



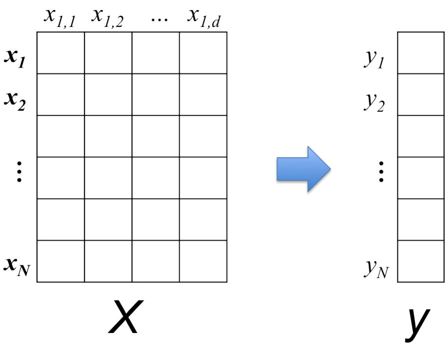
Types of data (X):

- Continuous: $\mathbb{R}^d$
- Discrete: $\{0, 1, \ldots, k\}$
- Structured (tree, string, ...)
- ...

Types of target ($\mathbf{y}$):

- Continuous: $\mathbb{R}$

Regression



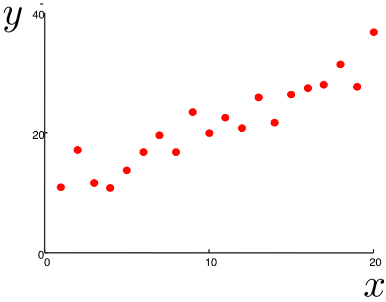
Examples:

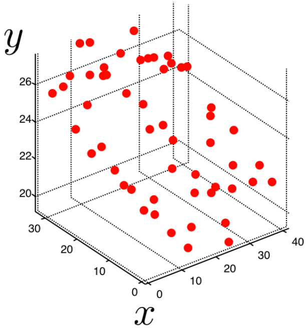
- Income, number of children $\Rightarrow$ Consumer spending
- Processes, memory $\Rightarrow$ Power consumption
- Financial reports $\Rightarrow$ Risk
- Atmospheric conditions $\Rightarrow$ Precipitation

Regression

Given examples $(\mathbf{x}_i, y_i)_{i=1,\dots,N}$

Predict y_t given a new test point $\mathbf{x}_t$

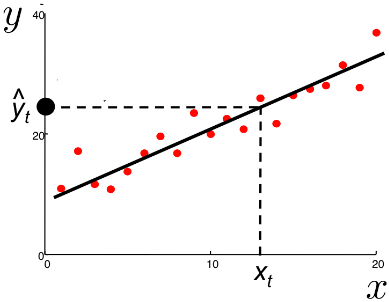


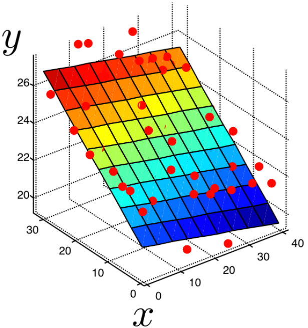


Goal is to estimate $\hat{y}_t$ by a linear function of given data $\mathbf{x}_t$:

$$\begin{aligned}\hat{y}_t &= w_0 + w_1 x_{t,1} + w_2 x_{t,2} + \cdots + w_d x_{t,d} \\ &= \mathbf{w}^T \mathbf{x}_t\end{aligned}$$

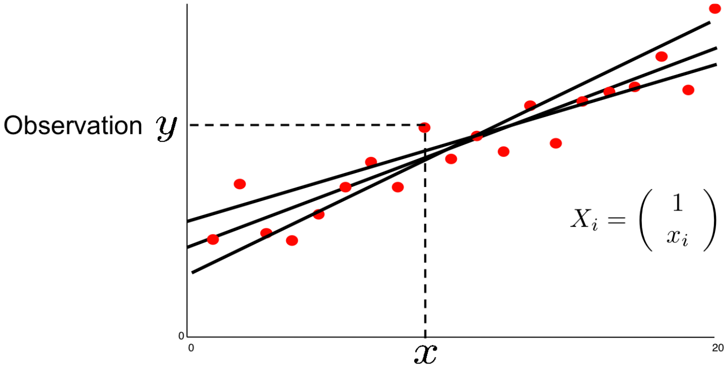
where $\mathbf{w}$ is the parameter to be estimated





Choosing the Regressor

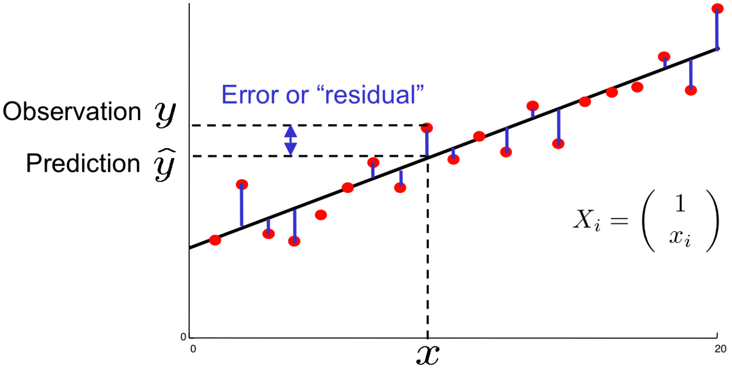
Of the many regression fits that approximate the data which one should we choose?



Least Squares

To clarify what we mean by a good choice of **w**
we define a cost function for how well we are doing on the training data

$$J_{\mathbf{w}} = \frac{1}{2}\sum_{i=1}^{N}(\mathbf{w}^T\mathbf{x}_i - y_i)^2$$



Normal Equations

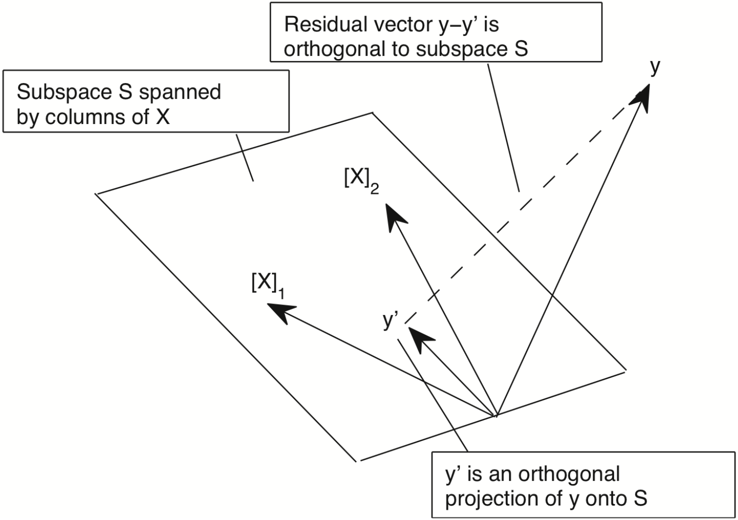
- Minimize the sum squared error $J_{\mathbf{w}}$

$$\begin{aligned} J_{\mathbf{w}} &= \frac{1}{2}\sum_{i=1}^{N}(\mathbf{w}^T\mathbf{x}_i - y_i)^2 \\ &= \frac{1}{2}(X\mathbf{w}-\mathbf{y})^T(X\mathbf{w}-\mathbf{y}) \\ &= \frac{1}{2}(\mathbf{w}^T X^T X\mathbf{w} - 2\mathbf{y}^T X\mathbf{w} + \mathbf{y}^T\mathbf{y}) \end{aligned}$$

- Derivative: $\dfrac{\partial}{\partial \mathbf{w}} J_{\mathbf{w}} = X^T X\mathbf{w} - X^T\mathbf{y}$
- Setting the derivative equal to zero gives the normal equations

$$\begin{aligned} X^T X\mathbf{w} &= X^T\mathbf{y} \\ \mathbf{w} &= (X^T X)^{-1}X^T\mathbf{y} \end{aligned}$$

Geometric Interpretation



Computing $\mathbf{w}$

Computing $\mathbf{w} = (X^T X)^{-1} X^T \mathbf{y}$

- If $X^T X$ is invertible
  - $(X^T X)^{-1} X^T$ coincides with the pseudoinverse $X^\dagger$ of X
  - Solution is unique
- If $X^T X$ is not invertible
  - There is no unique solution $\mathbf{w}$
  - $\mathbf{w} = X^\dagger \mathbf{y}$ chooses the solution with smallest Euclidean norm
  - Alternative way to deal with non-invertible $X^T X$ is to add a small multiple of the identity matrix (= Ridge regression)

Closed Form Solution for Linear Regression

$$\mathbf{w} = (X^T X)^{-1} X^T \mathbf{y}$$

On a machine with 8 cores, where X is a 20000×5000 matrix

```
>> % Matlab
>> tic; w=(X'*X)\(X'*y); toc
Elapsed time is 14.274773 seconds.

>> % Octave
>> tic; w=(X'*X)\(X'*y); toc
Elapsed time is 194.925 seconds.
```

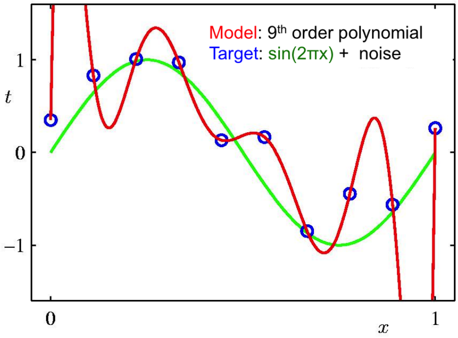
Huge difference, why?

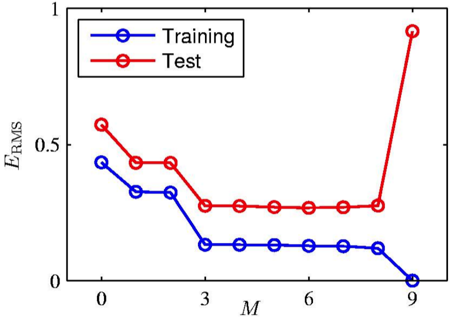
Closed Form Solution for Linear Regression

Different libraries for matrix computation and linear algebra operations

- Default BLAS and LAPACK, used by Octave
- Intel Math Kernel Library (Intel MKL), used by Matlab
- AMD Core Math Library (ACML)
- Automatically Tuned Linear Algebra Software (ATLAS)
- GoTo Blas, written by a former longhorn!

Overfitting





- Using too many features can lead to overfitting
- Least squares estimates often have low bias and large variance

Regularization

- Ridge Regression:
  - Objective:

$$J_{\mathbf{w}} = \underbrace{\frac{1}{2}\|X\mathbf{w} - \mathbf{y}\|_2^2}_{loss} + \underbrace{\lambda\|\mathbf{w}\|_2^2}_{L_2-regularization}$$

  - Setting the derivative equal to zero gives

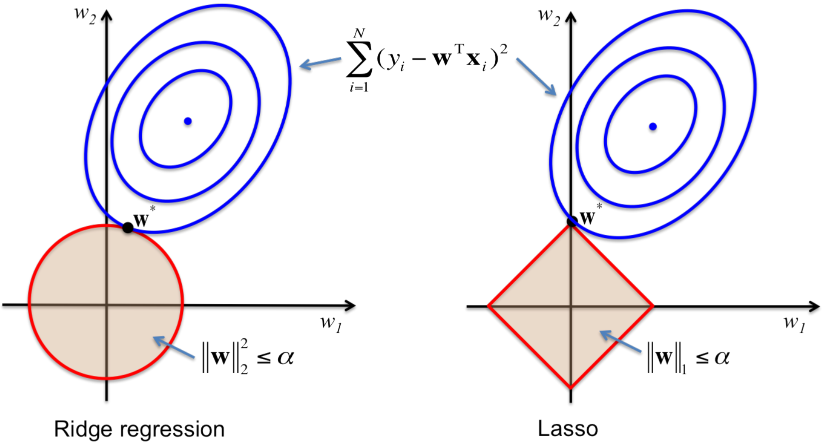
$$(X^T X + \lambda I)\mathbf{w} = X^T \mathbf{y}$$

- Lasso:
  - Objective:

$$J_{\mathbf{w}} = \underbrace{\frac{1}{2}\|X\mathbf{w} - \mathbf{y}\|_2^2}_{loss} + \underbrace{\lambda\|\mathbf{w}\|_1}_{L_1-regularization}$$

  - No closed form solution for $\mathbf{w}$ $\Rightarrow$ Iterative algorithms needed

Regularization



Regularized Risk Minimization

A general frame work for supervised learning

$$\min_{\mathbf{w}} \quad \textbf{Empirical loss} + \textbf{Regularization},$$

where

- $\mathbf{w}$: model parameter of the target function (e.g., coefficients of the hyperplane in linear regression)
- **Empirical loss**: performance of the current $\mathbf{w}$ estimated by the training data (e.g., $\sum_i (y_i - \mathbf{w}^T\mathbf{x}_i)^2$ is the square loss for linear regression)
- **Regularization**: a prior of the structure of the model. A common way to avoid overfitting (e.g., $\|\mathbf{w}\|_2^2$ and $\|\mathbf{w}\|_1$)

When it comes to large data

What we learned so far:

- Closed form solution:
  - $O(nd^2 + d^3)$ time and $O(d^2)$ space for linear regression
  - Not scalable for large d

Alternative methods:

- Stochastic Gradient Method:
  - One instance at a time
  - Obtain a model with reasonable performance for a few iterations
  - Online-fashion makes it also suitable for large-scale problems
- Coordinate Descent:
  - One variable at a time
  - Obtain a model with reasonable performance for a few iterations
  - Successfully applied in large-scale applications

Stochastic Gradient

Input: $X \in \mathbb{R}^{N \times d}$, $\mathbf{y} \in \mathbb{R}^N$, learning rate η, initial $\mathbf{w}^{(0)}$
Output: Solution $\mathbf{w}$

```
t = 0
while not converged do
    Choose a random training example x_i
    Compute gradient for x_i: ∇J_w(x_i)
    Update w: w^(t+1) ← w^(t) − η∇J_w(x_i)
    t ← t + 1
end while
```

Coordinate Descent for Lasso

Input: $X \in \mathbb{R}^{N \times d}$, $\mathbf{y} \in \mathbb{R}^N$, λ

Output: Solution $\mathbf{w}$

while not converged **do**

    for $j = 1$ **to** d **do**

        Compute partial residuals:

$$r_{ij} = y_i - \sum_{k \neq j} x_{ik} w_k$$

        Compute least squares coefficient of residuals on jth feature:

$$w_j^* = \frac{1}{\sum_{i=1}^N x_{ij}^2} \sum_{i=1}^N x_{ij} r_{ij}$$

        Update w_j by soft-thresholding:

$$w_j \leftarrow \text{sign}(w_j^*)(|w_j^*| - \lambda)_+$$

    end for

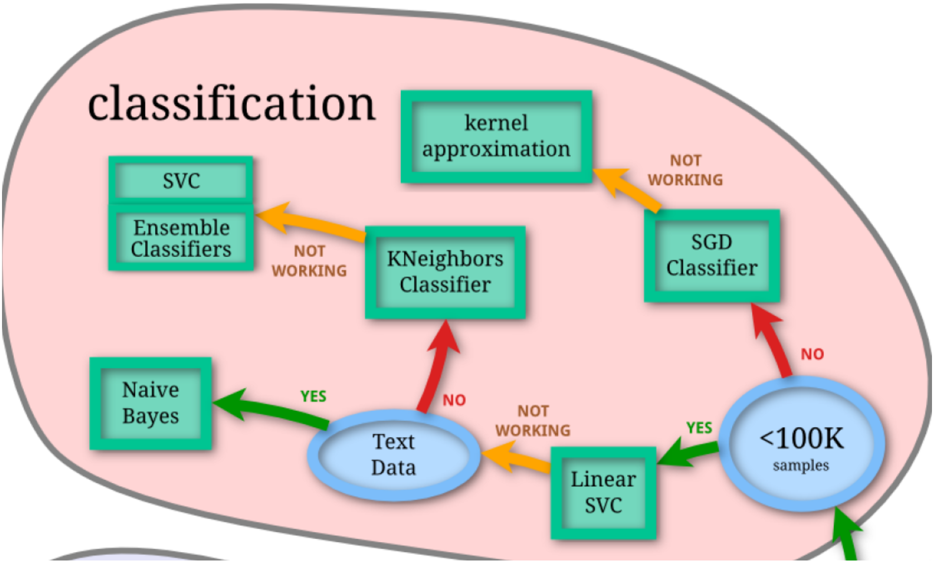
end while

Regression Solvers in Scikit-learn

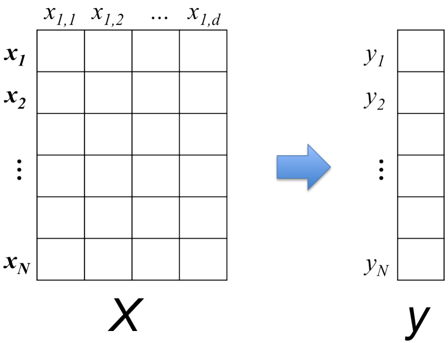
- Exact Solver for ordinary least square and Ridge Regression using LAPACK and BLAS
- Stochastic Gradient solvers for Ridge and Lasso
- Coordinate Descent solvers for Lasso and SVR

Classification

Scikit-learn: Classification



Classification



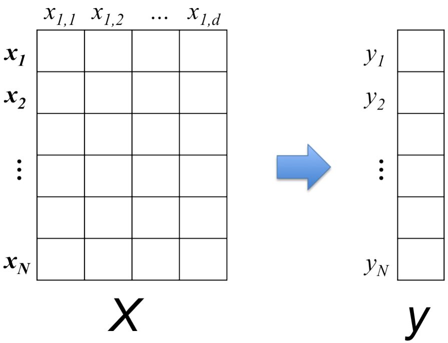
Types of data (X):

- Continuous: $\mathbb{R}^d$
- Discrete: $\{0, 1, \ldots, k\}$
- Structured (tree, string, ...)
- ...

Types of target ($\mathbf{y}$):

- Binary: $\{0, 1\}$
- Multi-class: $\{1, \ldots, k\}$
- Structured: tree, etc

Classification



Examples:

- Patients with and without disease $\Rightarrow$ Cancer or no-cancer
- Past movies you have watched $\Rightarrow$ Like or don't like
- Black-and-white pixel values $\Rightarrow$ Which digit is it?
- Past queries $\Rightarrow$ Whether the ad was clicked or not

Classification:
k-Nearest Neighbor

k-Nearest Neighbor

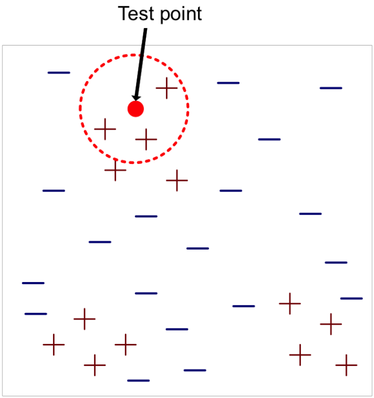
Majority vote within the k-nearest neighbors

- Set of training examples: $(\mathbf{x}_i, y_i)_{i=1,\ldots,N}$
- Define distance metric between two points $\mathbf{u}$ and $\mathbf{v}$

$$\text{e.g.} \quad d(\mathbf{u}, \mathbf{v}) = \|\mathbf{u} - \mathbf{v}\|_2$$

- Classify new test point $\mathbf{x}_t$ by looking at labels of k closest examples, $\mathcal{N}_k(x_t)$, in the training set

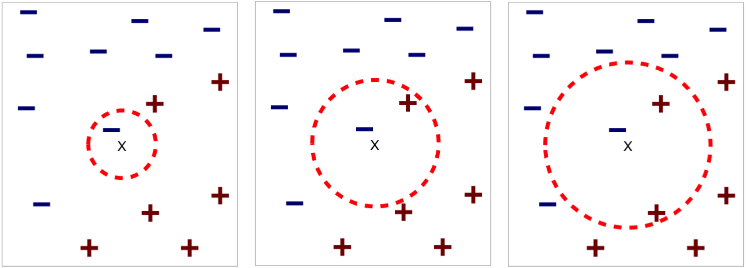
$$y_t = \frac{1}{k} \sum_{\mathbf{x}_i \in \mathcal{N}_k(\mathbf{x}_t)} y_i$$



k-Nearest Neighbor

Choosing k:

- If k is too small, sensitive to noise points
- If k is too large, neighborhood may include points from other class



(a) 1-nearest neighbor (b) 2-nearest neighbor (c) 3-nearest neighbor

Use "validation data": pick k with highest performance on validation set

k-Nearest Neighbor

Pros:

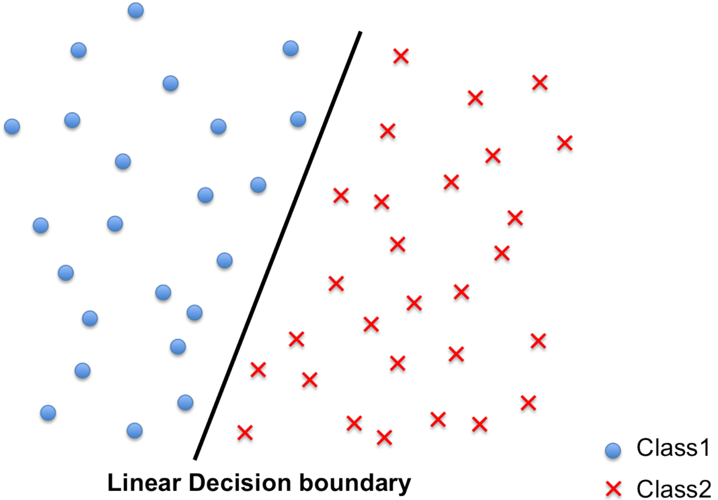
- Can express complex boundary — non-parametric
- Very fast training: need efficient data structure to look for closest point quickly (e.g. kd-trees, locality sensitive hashing)
- Simple, but still very good in practice
- Somewhat interpretable by looking at closest point

Cons:

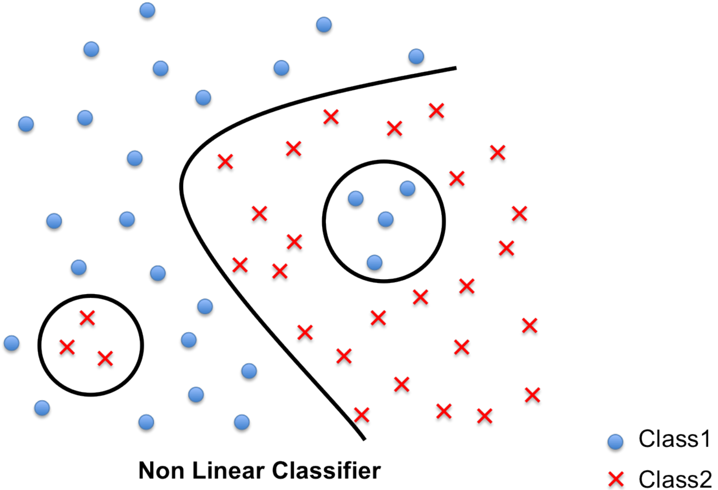
- Large memory requirement for prediction
- Not the best accuracy amongst classifiers

Classification: Support Vector Machine

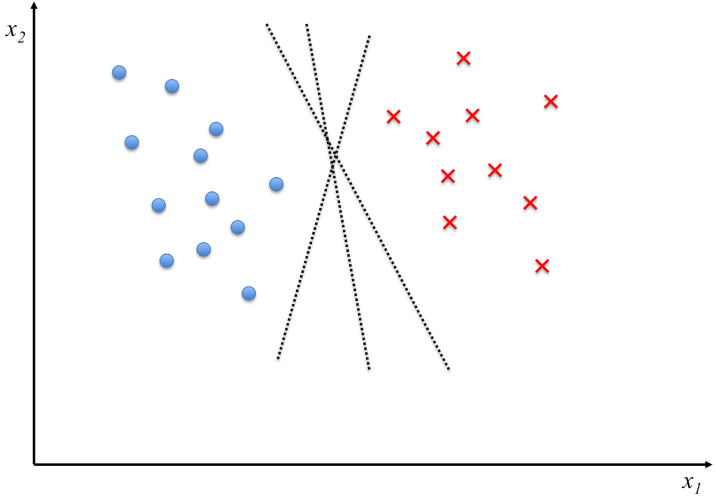
Linearly Separable Data



Nonlinearly Separable Data

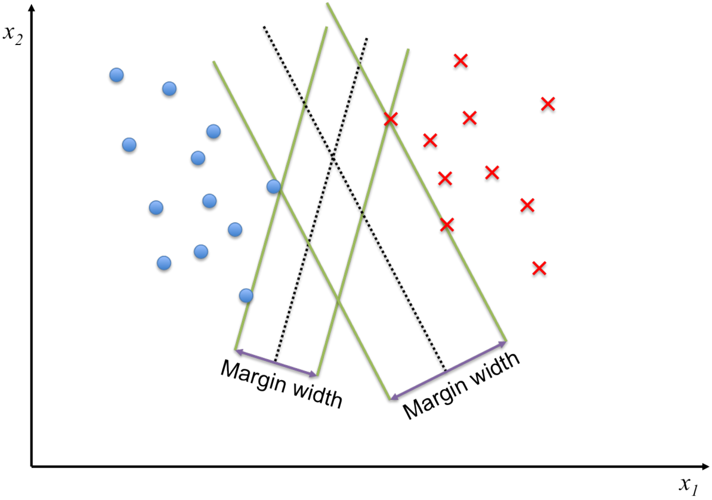


Which Separating Hyperplane to Use?

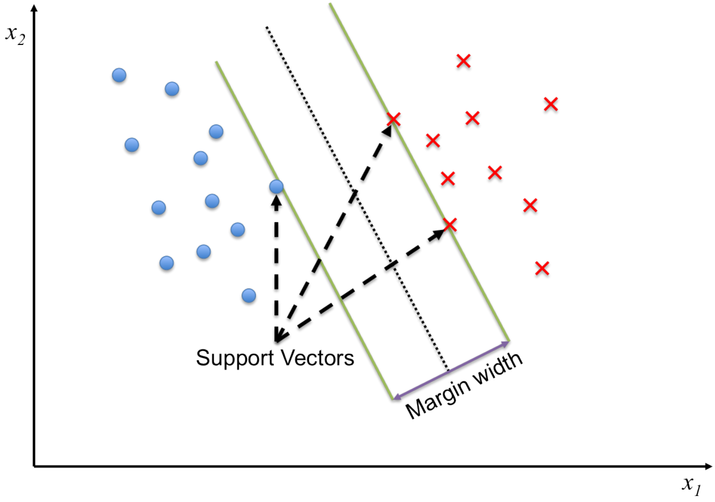


Maximizing the Margin

Select the separating hyperplane that maximizes the margin



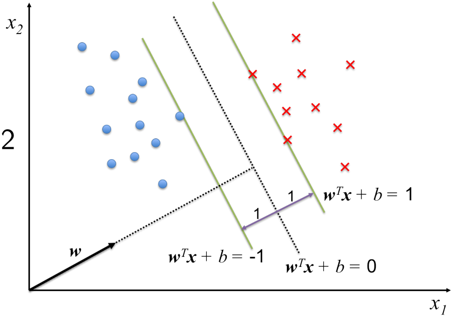
Support Vectors



Setting Up the Optimization Problem

The maximum margin can be characterized as a solution to an optimization problem:

$$\begin{aligned} \max \quad & \frac{2}{\|\mathbf{w}\|} \\ \text{s.t.} \quad & \mathbf{w}^T\mathbf{x}_i + b \geq 1, \quad \forall \mathbf{x}_i \text{ of class1} \\ & \mathbf{w}^T\mathbf{x}_i + b \leq -1, \quad \forall \mathbf{x}_i \text{ of class2} \end{aligned}$$



or equivalently

$$\begin{aligned} \min \quad & \frac{1}{2}\|\mathbf{w}\|^2 \\ \text{s.t.} \quad & y_i(\mathbf{w}^T\mathbf{x}_i + b) \geq 1, \quad \forall \mathbf{x}_i \end{aligned}$$

Linear, Hard-Margin SVM Formulation

Find $\mathbf{w}$ and b that solves

$$\begin{aligned} \min \quad & \frac{1}{2}\|\mathbf{w}\|^2 \\ \text{s.t.} \quad & y_i(\mathbf{w}^T\mathbf{x}_i + b) \geq 1, \quad \forall \mathbf{x}_i \end{aligned}$$

- Problem is convex, so there is a unique global minimum value (when feasible)
- There is also a unique minimizer, i.e. $\mathbf{w}$ and b that provides the minimum
- Quadratic Programming

Nonlinearly Separable Data

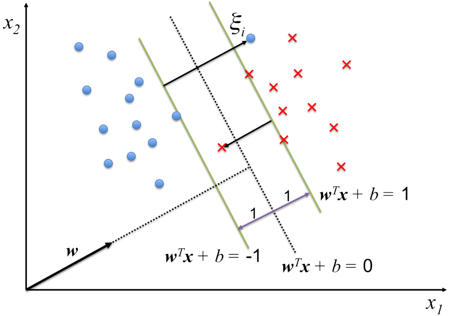
Introduce slack variables ξ_i

Allow some instances to fall within the margin, but penalize them

$$\min \quad \frac{1}{2}\|\mathbf{w}\|^2 + C\sum_i \xi_i$$

$$\text{s.t.} \quad y_i(\mathbf{w}^T\mathbf{x}_i + b) \geq 1 - \xi_i, \quad \forall \mathbf{x}_i$$

$$\xi_i \geq 0$$



C trades-off margin width and misclassifications

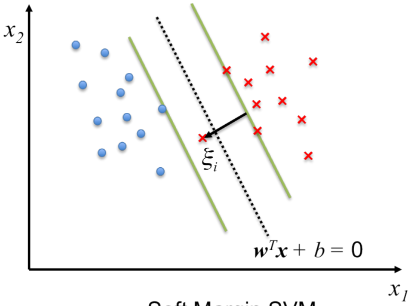
Linear, Soft-Margin SVM Formulation

Find $\mathbf{w}$ and b that solves

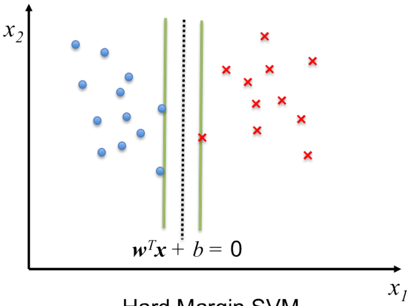
$$\begin{aligned} \min \quad & \frac{1}{2}\|\mathbf{w}\|^2 + C\sum_i \xi_i \\ \text{s.t.} \quad & y_i(\mathbf{w}^T\mathbf{x}_i + b) \geq 1 - \xi_i, \quad \forall \mathbf{x}_i \\ & \xi_i \geq 0 \end{aligned}$$

- Algorithm tries to maintain ξ_i to zero while maximizing margin
- Notice: algorithm does not minimize the number of misclassifications (NP-complete problem) but the sum of distances from the margin hyperplanes
- As $C \to 0$, we get the hard-margin solution

Robustness of Soft vs. Hard Margin SVMs



Soft Margin SVM



Hard Margin SVM

Regularized Risk Minimization

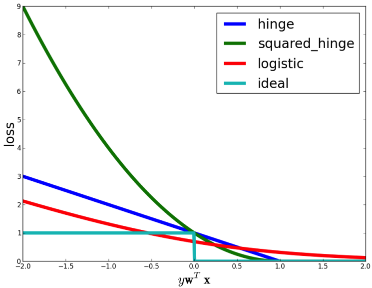
Soft margin SVM can be written as regularized risk minimization form:

$$\min_{\mathbf{w}} \quad \textbf{Empirical loss} + \textbf{Regularization}$$

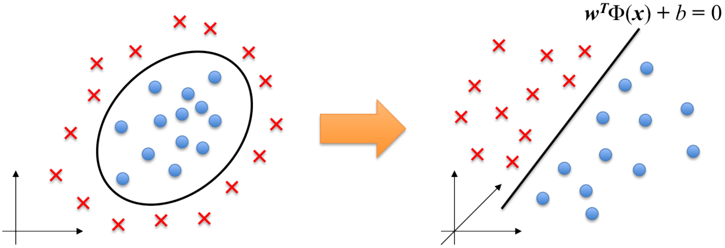
- Hinge loss: $\sum_i \max(0, 1 - y_i \mathbf{w}^T \mathbf{x}_i)$
- L2 regularization: $\|\mathbf{w}\|_2^2$

Other loss functions for classification:

- Ideal loss: $\sum_i I[y_i \mathbf{w}^T \mathbf{x}_i < 0]$
- Squared hinge loss: $\sum_i \max(0, 1 - y_i \mathbf{w}^T \mathbf{x}_i)^2$
- Logistic loss: $\sum_i \log\left(1 + \exp\left(-y_i \mathbf{w}^T \mathbf{x}_i\right)\right)$



Kernel Example



$\boldsymbol{x} = [x_1\ x_2]$ $\qquad$ $\Phi(\boldsymbol{x}) = [x_1{}^2\ x_2{}^2\ \sqrt{2}x_1x_2]$

$$
\begin{aligned}
\Phi(\mathbf{x}_i)^T\Phi(\mathbf{x}_j) &= \begin{bmatrix} x_{i1}^2 & x_{i2}^2 & \sqrt{2}x_{i1}x_{i2} \end{bmatrix} \begin{bmatrix} x_{j1}^2 & x_{j2}^2 & \sqrt{2}x_{j1}x_{j2} \end{bmatrix}^T \\
&= x_{i1}^2 x_{j1}^2 + x_{i2}^2 x_{j2}^2 + 2x_{i1}x_{i2}x_{j1}x_{j2} \\
&= (x_{i1}x_{j1} + x_{i2}x_{j2})^2 \\
&= (\mathbf{x}_i^T\mathbf{x}_j)^2
\end{aligned}
$$

The Primal of the SVM Formulation

- Original (Primal) SVM formulation:

$$\begin{aligned} \min \quad & \frac{1}{2}\|\mathbf{w}\|^2 + C\sum_i \xi_i \\ \text{s.t.} \quad & y_i(\mathbf{w}^T\Phi(\mathbf{x}_i) + b) \geq 1 - \xi_i, \quad \forall \mathbf{x}_i \\ & \xi_i \geq 0 \end{aligned}$$

The Dual of the SVM Formulation

- Dual SVM formulation:

$$
\begin{aligned}
\min \quad & \frac{1}{2}\sum_{i,j}\alpha_i\alpha_j y_i y_j \Phi(\mathbf{x}_i)^T\Phi(\mathbf{x}_j) - \sum_i \alpha_i \\
\text{s.t.} \quad & 0 \le \alpha_i \le C, \quad \forall \mathbf{x}_i \\
& \sum_i \alpha_i y_i = 0
\end{aligned}
$$

NOTE: Data only appear as $\Phi(\mathbf{x}_i)^T\Phi(\mathbf{x}_j)$

The Kernel Trick

- $\Phi(\mathbf{x}_i)^T\Phi(\mathbf{x}_j)$ means, map data into new space, then take the inner product of the new vectors
- We can find a function such that: $K(\mathbf{x}_i, \mathbf{x}_j) = \Phi(\mathbf{x}_i)^T\Phi(\mathbf{x}_j)$, i.e., the image of the inner product of the data is the inner product of the images of the data
- Then, we do not need to explicitly map the data into the high-dimensional space to solve the optimization problem
- Only inner products explicitly needed for training and evaluation

Beyond Binary Classification

Many applications have more than two classes.

- Character recognition (e.g., digits, letters)
- Face recognition

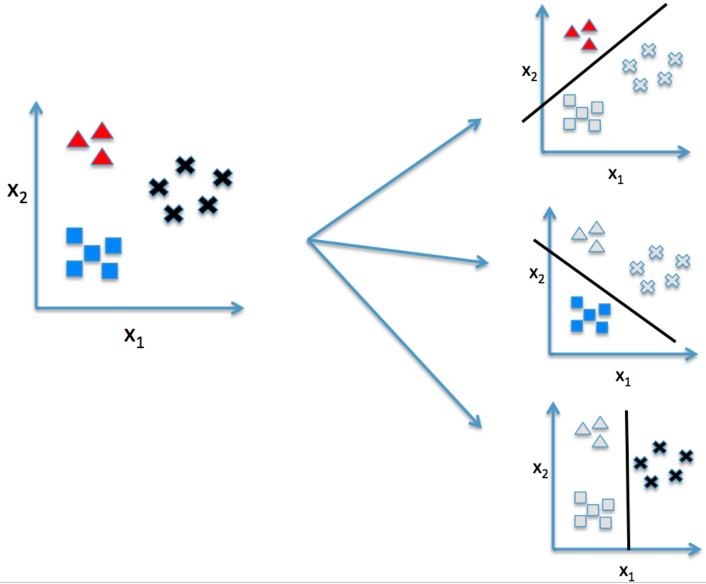
Approaches:

- Extend binary classifiers to handle multiple classes
  - One-versus-rest (OVR)
  - One-versus-One (OVO)
- A new model considers multiple classes together (e.g., Crammer & Singer 2001)

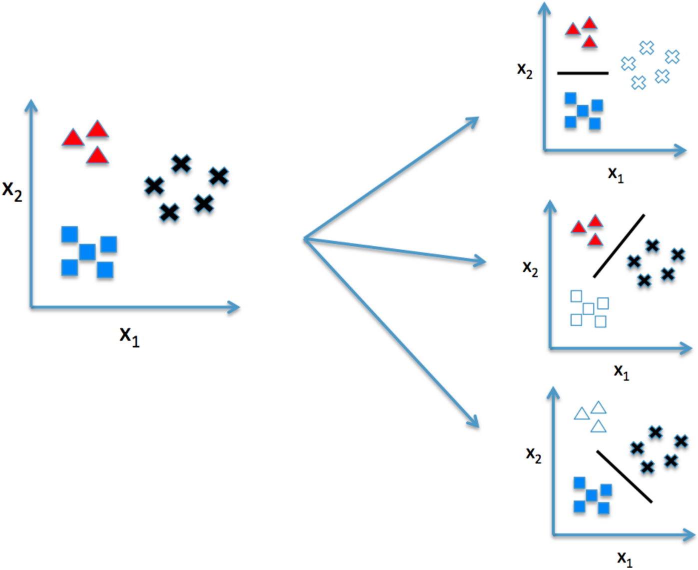
Multilabel Classification Problem

- An instance might belong to more than one class
- E.g., Automatic wikipage categorization/ Image tag generation

Multi-class Classification: One-versus-Rest



Multi-class Classification: One-versus-One



Classification Solvers in Scikit-learn

- Stochastic Gradient solvers for SVM/logistic regression with both L1/L2 regularization
- Coordinate Descent solvers for SVM/logistic regression with both L1/L2 regularization (LIBLINEAR/LIBSVM are used for SVM)
- Nearest Neighbors, Naive Bayes, Decision Trees, etc
- All classifiers support multiple classes

Think Parallel

Parallelization for Machine Learning

Designing parallel algorithms for existing models

- Not an easy task
- Usually model or problem specific
- Active research topic with many problems to explore

Some "easier" machine learning tasks which can be done in parallel:

- Prediction
- Multi-class classification (One-versus-rest, One-versus-one)
- Model Selection

Model Selection

Most machine learning models

- One or more parameters

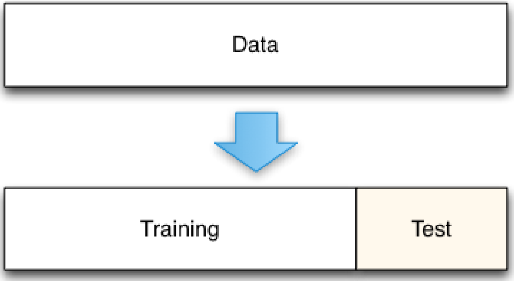
  E.g., λ in Ridge Regression and SVM

  E.g., k in k-Nearest Neighbor
- Parameter selection is crucial to achieve good performance in practice

How to evaluate the performance of a given set of parameters?

- Training error – risk to overfit
- Holdout validation
- Cross-validation

Holdout validation



5-fold Cross-validation