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: $k$NN, SVM

- Unsupervised Learning — day2
  - Clustering: $k$-means, Spectral Clustering
  - Dimensionality Reduction: PCA, Matrix Factorization for Recommender Systems
What is Machine Learning?

Machine Learning

- Computer
- Model learned from data

Training data ➔ Model ➔ Test data ➔ Prediction

Learn ➔ Model ➔ Apply model
Machine Learning Applications

Link prediction

LinkedIn.

fMRI

Spam classification

Image classification

gene-gene network

Inderjit S. Dhillon Dept of Computer Science UT Austin

Machine Learning: Think Big and Parallel
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: $k$NN, 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
Regression
Regression

- SGD Regressor
- Lasso
- ElasticNet
- SVR(kernel='rbf')
- Ensemble Regressors
- Ridge Regression
- SVR(kernel='linear')

<100K samples

few features should be important

Machine Learning: Think Big and Parallel
Regression

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

Types of target ($y$):
- Continuous: $\mathbb{R}$
Regression

Examples:

- Income, number of children ⇒ Consumer spending
- Processes, memory ⇒ Power consumption
- Financial reports ⇒ Risk
- Atmospheric conditions ⇒ Precipitation
Regession

Given examples \((x_i, y_i)_{i=1,...,N}\)

Predict \(y_t\) given a new test point \(x_t\)
Regression

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

$$\hat{y}_t = w_0 + w_1 x_{t,1} + w_2 x_{t,2} + \cdots + w_d x_{t,d}$$

$$= w^T x_t$$

where $w$ is the parameter to be estimated
Choosing the Regressor

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

\[
X_i = \begin{pmatrix} 1 \\ x_i \end{pmatrix}
\]
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

\[
J_w = \frac{1}{2} \sum_{i=1}^{N} (w^T x_i - y_i)^2
\]

\[
X_i = \begin{pmatrix} 1 \\ x_i \end{pmatrix}
\]
Normal Equations

- Minimize the sum squared error $J_w$

$$J_w = \frac{1}{2} \sum_{i=1}^{N} (w^T x_i - y_i)^2$$

$$= \frac{1}{2} (Xw - y)^T (Xw - y)$$

$$= \frac{1}{2} (w^T X^T Xw - 2y^T Xw + y^T y)$$

- Derivative: $\frac{\partial}{\partial w} J_w = X^T Xw - X^T y$

- Setting the derivative equal to zero gives the normal equations

$$X^T Xw = X^T y$$

$$w = (X^T X)^{-1} X^T y$$
Geometric Interpretation

Subspace $S$ spanned by columns of $X$

Residual vector $y - y'$ is orthogonal to subspace $S$

$y'$ is an orthogonal projection of $y$ onto $S$
Computing **w**

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

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

\[ w = (X^T X)^{-1} X^T y \]

On a machine with 8 cores, where \( X \) is a \( 20000 \times 5000 \) 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:

\[ J_w = \frac{1}{2} \| Xw - y \|_2^2 + \lambda \| w \|_2^2 \]

• Setting the derivative equal to zero gives

\[ (X^TX + \lambda I)w = X^Ty \]

• Lasso:
  • Objective:

\[ J_w = \frac{1}{2} \| Xw - y \|_2^2 + \lambda \| w \|_1 \]

• No closed form solution for \( w \) ⇒ Iterative algorithms needed
Regularization

\[ \sum_{i=1}^{N} (y_i - \mathbf{w}^T \mathbf{x}_i)^2 \leq \alpha \]

Ridge regression

\[ \|\mathbf{w}\|_2 \leq \alpha \]

Lasso

\[ \|\mathbf{w}\|_1 \leq \alpha \]
A general framework for supervised learning

\[
\min_w \text{ Empirical loss} + \text{Regularization},
\]

where

- **\( w \):** model parameter of the target function (e.g., coefficients of the hyperplane in linear regression)

- **Empirical loss:** performance of the current \( w \) estimated by the training data (e.g., \( \sum_i(y_i - w^T x_i)^2 \) is the square loss for linear regression)

- **Regularization:** a prior of the structure of the model. A common way to avoid overfitting (e.g., \( \|w\|_2^2 \) and \( \|w\|_1 \))
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:** $X \in \mathbb{R}^{N \times d}$, $y \in \mathbb{R}^N$, learning rate $\eta$, initial $w^{(0)}$

**Output:** Solution $w$

1. $t = 0$
2. *while* not converged *do*
3. Choose a random training example $x_i$
4. Compute gradient for $x_i$: $\nabla J_w(x_i)$
5. Update $w$: $w^{(t+1)} \leftarrow w^{(t)} - \eta \nabla J_w(x_i)$
6. $t \leftarrow t + 1$
7. *end while*
Coordinate Descent for Lasso

**Input:** \( X \in \mathbb{R}^{N \times d} \), \( y \in \mathbb{R}^N \), \( \lambda \)

**Output:** Solution \( w \)

1. **while** not converged **do**
2. **for** \( j = 1 \) **to** \( d \) **do**
3. Compute partial residuals:
   \[ r_{ij} = y_i - \sum_{k \neq j} x_{ik} w_k \]
4. Compute least squares coefficient of residuals on \( j \)th feature:
   \[ w_j^* = \frac{1}{\sum_{i=1}^{N} x_{ij}^2} \sum_{i=1}^{N} x_{ij} r_{ij} \]
5. Update \( w_j \) by soft-thresholding:
   \[ w_j \leftarrow \text{sign}(w_j^*) (|w_j^*| - \lambda)_+ \]
6. **end for**
7. **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

SVC
Ensemble Classifiers

kernel approximation

KNeighbors Classifier

SGD Classifier

Naive Bayes

Text Data

Linear SVC

<100K samples

NOT WORKING

NOT WORKING

NOT WORKING

YES

NO

NO

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

Types of target ($y$):
- Binary: $\{0, 1\}$
- Multi-class: $\{1, \ldots, k\}$
- Structured: tree, etc
Classification

Examples:

- Patients with and without disease ⇒ Cancer or no-cancer
- Past movies you have watched ⇒ Like or don’t like
- Black-and-white pixel values ⇒ Which digit is it?
- Past queries ⇒ Whether the ad was clicked or not
Classification: 
\(k\)-Nearest Neighbor
Majority vote within the $k$-nearest neighbors

- Set of training examples: $(x_i, y_i)_{i=1,...,N}$
- Define distance metric between two points $u$ and $v$
  
  \[ d(u, v) = \| u - v \|_2 \]

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

\[
y_t = \frac{1}{k} \sum_{x_i \in \mathcal{N}_k(x_t)} y_i
\]
**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

Use “validation data”: pick $k$ with highest performance on validation set

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

Linear Decision boundary

Class1
Class2
Nonlinearly Separable Data

Non Linear Classifier

Class1

Class2
Which Separating Hyperplane to Use?

Inderjit S. Dhillon  Dept of Computer Science UT Austin

Machine Learning: Think Big and Parallel
Maximizing the Margin

Select the separating hyperplane that maximizes the margin

$x_2$

Margin width

Margin width

$x_1$
Support Vectors
Setting Up the Optimization Problem

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

\[
\begin{align*}
\text{max} & \quad \frac{2}{\|w\|} \\
\text{s.t.} & \quad w^T x_i + b \geq 1, \; \forall x_i \text{ of class 1} \\
& \quad w^T x_i + b \leq -1, \; \forall x_i \text{ of class 2}
\end{align*}
\]

or equivalently

\[
\begin{align*}
\text{min} & \quad \frac{1}{2} \|w\|^2 \\
\text{s.t.} & \quad y_i(w^T x_i + b) \geq 1, \; \forall x_i
\end{align*}
\]
Linear, Hard-Margin SVM Formulation

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

$$\min \frac{1}{2} \| \mathbf{w} \|^2$$

subject to

$$y_i (\mathbf{w}^T \mathbf{x}_i + b) \geq 1, \quad \forall \mathbf{x}_i$$

- 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 $\xi_i$

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

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

s.t. $y_i(w^T x_i + b) \geq 1 - \xi_i, \ \forall x_i$

$\xi_i \geq 0$

$C$ trades-off margin width and misclassifications
Linear, Soft-Margin SVM Formulation

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

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

s.t. \quad \begin{align*}
y_i (\mathbf{w}^T \mathbf{x}_i + b) & \geq 1 - \xi_i, \quad \forall \mathbf{x}_i \\
\xi_i & \geq 0
\end{align*}

- Algorithm tries to maintain $\xi_i$ 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

$\mathbf{w}^T \mathbf{x} + b = 0$

$x_1$

$x_2$

$\xi_i$
Regularized Risk Minimization

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

\[
\min_w \quad \text{Empirical loss + Regularization}
\]

- Hinge loss: \( \sum_i \max(0, 1 - y_i w^T x_i) \)
- L2 regularization: \( \|w\|_2^2 \)

Other loss functions for classification:

- Ideal loss: \( \sum_i I[y_i w^T x_i < 0] \)
- Squared hinge loss: \( \sum_i \max(0, 1 - y_i w^T x_i)^2 \)
- Logistic loss: \( \sum_i \log (1 + \exp(-y_i w^T x_i)) \)
Kernel Example

\[
\begin{align*}
\Phi(x_i)^T \Phi(x_j) & = \begin{bmatrix} x_{i1}^2 & x_{i2}^2 & \sqrt{2}x_{i1}x_{i2} \end{bmatrix} \begin{bmatrix} x_{j1}^2 & x_{j2}^2 & \sqrt{2}x_{j1}x_{j2} \end{bmatrix}^T \\
& = x_{i1}^2x_{j1}^2 + x_{i2}^2x_{j2}^2 + 2x_{i1}x_{i2}x_{j1}x_{j2} \\
& = (x_{i1}x_{j1} + x_{i2}x_{j2})^2 \\
& = (x_i^T x_j)^2
\end{align*}
\]

\(x = [x_1 \ x_2] \quad \Phi(x) = [x_1^2 \ x_2^2 \ \sqrt{2}x_1x_2] \quad \omega^T \Phi(x) + b = 0\)
Original (Primal) SVM formulation:

\[
\begin{align*}
\min & \quad \frac{1}{2} \|w\|^2 + C \sum_i \xi_i \\
\text{s.t.} & \quad y_i(w^T \Phi(x_i) + b) \geq 1 - \xi_i, \quad \forall x_i \\
& \quad \xi_i \geq 0
\end{align*}
\]
The Dual of the SVM Formulation

- Dual SVM formulation:

\[
\begin{align*}
\min & \quad \frac{1}{2} \sum_{i,j} \alpha_i \alpha_j y_i y_j \Phi(x_i)^T \Phi(x_j) - \sum_i \alpha_i \\
\text{s.t.} & \quad 0 \leq \alpha_i \leq C, \quad \forall x_i \\
& \quad \sum_i \alpha_i y_i = 0
\end{align*}
\]

NOTE: Data only appear as \( \Phi(x_i)^T \Phi(x_j) \)
The Kernel Trick

- $\Phi(x_i)^T \Phi(x_j)$ means, map data into new space, then take the inner product of the new vectors.

- We can find a function such that: $K(x_i, x_j) = \Phi(x_i)^T \Phi(x_j)$, 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

Inderjit S. Dhillon Dept of Computer Science UT Austin

Machine Learning: Think Big and Parallel
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., $\lambda$ 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

Data

Training  Test
5-fold Cross-validation