• **Regression**
  • Lasso L1 regularization
  • SVM regression and epsilon-insensitive loss

• **More loss functions**

• **Multi-class Classification**
  • using binary classifiers
  • random forests
  • neural networks
Regression cost functions

Minimize with respect to $w$

$$
\sum_{i=1}^{N} l \left( f(x_i, w), y_i \right) + \lambda R(w)
$$

- loss function
- regularization

• There is a choice of both loss functions and regularization

• So far we have seen – “ridge” regression

  • squared loss: $\sum_{i=1}^{N} (y_i - f(x_i, w))^2$

  • squared regularizer: $\lambda \|w\|^2$

• Now, consider other losses and regularizers
The “Lasso” or $L_1$ norm regularization

• LASSO = Least Absolute Shrinkage and Selection

Minimize with respect to $w \in \mathbb{R}^d$

$$\sum_{i=1}^{N} (y_i - f(x_i, w))^2 + \lambda \sum_{j} |w_j|$$

- loss function
- regularization

• This is a quadratic optimization problem
• There is a unique solution
• $p$-Norm definition: $||w||_p = \left( \sum_{j=1}^{d} |w_i|^p \right)^{\frac{1}{p}}$
Sparsity property of the Lasso

- contour plots for $d = 2$

- Minimum where loss contours tangent to regularizer’s
- For the lasso case, minima occur at “corners”
- Consequently one of the weights is zero
- In high dimensions many weights can be zero
Comparison of learnt weight vectors

Linear regression

$N = 100$

$d = 10$

$y = w \cdot x$

**Least Squares**

**Ridge Regression**

**LASSO**

lambda = 1

Lasso: number of non-zero coefficients = 5
Lasso in action

Ridge Regression

percent of lambdaMax

coefficient values

L1-Regularized Least Squares

percent of lambdaMax

coefficient values

regularization parameter lambda
Sparse weight vectors

- Weights being zero is a method of “feature selection” – zeroing out the unimportant features

- The SVM classifier also has this property (sparse alpha in the dual representation)

- Ridge regression does not
Other regularizers

\[ \sum_{i=1}^{N} (y_i - f(x_i, w))^2 + \lambda \sum_{j} |w_j|^q \]

- For \( q \geq 1 \), the cost function is convex and has a unique minimum. The solution can be obtained by quadratic optimization.

- For \( q < 1 \), the problem is not convex, and obtaining the global minimum is more difficult.
SVMs for Regression

Use $\varepsilon$-insensitive error measure

$$V_\varepsilon(r) = \begin{cases} 
0 & \text{if } |r| \leq \varepsilon \\
|r| - \varepsilon & \text{otherwise.}
\end{cases}$$

This can also be written as

$$V_\varepsilon(r) = (|r| - \varepsilon)_+$$

where $(\cdot)_+$ indicates the positive part of $(\cdot)$.

Or equivalently as

$$V_\varepsilon(r) = \max((|r| - \varepsilon), 0)$$

The loss function regularization

$$\tilde{E}(w) = \sum_{i=1}^{N} V_\varepsilon(f(x_i, w) - y_i) + \frac{\lambda}{2} \|w\|^2$$

loss function  regularization

cost is zero inside epsilon “tube”

square loss
As before, introduce slack variables for points that violate \( \varepsilon \)-insensitive error.

For each data point, \( x_i \), two slack variables, \( \xi_i, \hat{\xi}_i \), are required (depending on whether \( f(x_i) \) is above or below the tube).

Learning is by the optimization

\[
\min_{w \in \mathbb{R}^d, \xi_i, \hat{\xi}_i} C \sum_{i=1}^{N} \left( \xi_i + \hat{\xi}_i \right) + \frac{1}{2} ||w||^2
\]

subject to

\[
y_i \leq f(x_i, w) + \varepsilon + \xi_i, \quad y_i \geq f(x_i, w) - \varepsilon - \hat{\xi}_i, \quad \xi_i \geq 0, \quad \hat{\xi}_i \geq 0 \text{ for } i = 1 \ldots N
\]

Again, this is a quadratic programming problem

It can be dualized

Some of the data points will become support vectors

It can be kernelized
Example: SV regression with Gaussian basis functions

- Regression function – Gaussians centred on data points
- Parameters are: C, epsilon, sigma

\[ f(x, \mathbf{a}) = \sum_{i=1}^{N} a_i e^{-\frac{(x-x_i)^2}{2\sigma^2}} \]

N = 50
C = 1000,  epsilon = 0.2, sigma = 0.5

As epsilon increases:

• fit becomes looser
• less data points are support vectors
Loss functions for regression

- **quadratic (square) loss** \( \ell(y, f(x)) = \frac{1}{2}(y - f(x))^2 \)

- **\(\varepsilon\)-insensitive loss** \( \ell(y, f(x)) = \max((|r| - \varepsilon), 0) \)

- **H"uber loss** (mixed quadratic/linear): robustness to outliers: 
  \[ \ell(y, f(x)) = h(y - f(x)) \]
  \[ h(r) = \begin{cases} 
  r^2 & \text{if } |r| \leq c \\
  2c|r| - c^2 & \text{otherwise.} 
  \end{cases} \]

- all of these are convex
Final notes on cost functions

Regressors and classifiers can be constructed by a “mix ‘n’ match” of loss functions and regularizers to obtain a learning machine suited to a particular application. e.g. for a classifier $f(x) = w^\top x + b$

- **L1–SVM**

$$\min_{w \in \mathbb{R}^d} \sum_{i}^{N} \max (0, 1 - y_i f(x_i)) + \lambda ||w||_1$$

- **Least squares SVM**

$$\min_{w \in \mathbb{R}^d} \sum_{i}^{N} [\max (0, 1 - y_i f(x_i))]^2 + \lambda ||w||^2$$
Multi-class Classification
Multi-Class Classification – what we would like

Assign input vector $\mathbf{x}$ to one of $K$ classes $C'_k$.

Goal: a decision rule that divides input space into $K$ decision regions separated by decision boundaries.
Reminder: K Nearest Neighbour (K-NN) Classifier

Algorithm

• For each test point, $x$, to be classified, find the K nearest samples in the training data
• Classify the point, $x$, according to the majority vote of their class labels

e.g. $K = 3$

- naturally applicable to multi-class case
Build from binary classifiers

- **Learn:** $K$ two-class 1-vs-the-rest classifiers $f_k(x)$
Build from binary classifiers continued

• **Learn:** K two-class 1 vs the rest classifiers $f_k(x)$
• **Classification:** choose class with most positive score

$$\max_k f_k(x)$$
Build from binary classifiers continued

- **Learn**: $K$ two-class 1 vs the rest classifiers $f_k(x)$
- **Classification**: choose class with most positive score

$$\max_k f_k(x)$$
Application: hand written digit recognition

- **Feature vectors:** each image is 28 x 28 pixels. Rearrange as a 784-vector $\mathbf{x}$

- **Training:** learn $k=10$ two-class 1-vs-the-rest SVM classifiers $f_k(\mathbf{x})$

- **Classification:** choose class with most positive score

$$ f(\mathbf{x}) = \max_k f_k(\mathbf{x}) $$
Example

<table>
<thead>
<tr>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>4</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

hand drawn

classification
Why not learn a multi-class SVM directly?

For example for three classes

- Learn \( \mathbf{w} = (\mathbf{w}_1, \mathbf{w}_2, \mathbf{w}_3)^\top \) using the cost function

\[
\min_{\mathbf{w}} ||\mathbf{w}||^2 \quad \text{subject to}
\]

\[
\mathbf{w}_1^\top \mathbf{x}_i \geq \mathbf{w}_2^\top \mathbf{x}_i \quad \& \quad \mathbf{w}_1^\top \mathbf{x}_i \geq \mathbf{w}_3^\top \mathbf{x}_i \quad \text{for } i \in \text{class 1}
\]

\[
\mathbf{w}_2^\top \mathbf{x}_i \geq \mathbf{w}_3^\top \mathbf{x}_i \quad \& \quad \mathbf{w}_2^\top \mathbf{x}_i \geq \mathbf{w}_1^\top \mathbf{x}_i \quad \text{for } i \in \text{class 2}
\]

\[
\mathbf{w}_3^\top \mathbf{x}_i \geq \mathbf{w}_1^\top \mathbf{x}_i \quad \& \quad \mathbf{w}_3^\top \mathbf{x}_i \geq \mathbf{w}_2^\top \mathbf{x}_i \quad \text{for } i \in \text{class 3}
\]

- This is a quadratic optimization problem subject to linear constraints and there is a unique minimum

- Note, a margin can also be included in the constraints

In practice there is a little or no improvement over the binary case
Random Forests
Random Forest Overview

• A natural multiple class classifier

• Fast at test time

• Start with single tree, and then move onto forest
Generic trees and decision trees

A general tree structure

A decision tree

Classify as green class

Classify as blue class

Classify as red class
• choose class with max posterior at leaf node
**Decision forest model:** training and information gain
(for discrete, non-parametric distributions)

Information gain

\[ I = H(S) - \sum_{i \in \{1,2\}} \frac{|S^i|}{|S|} H(S^i) \]

Shannon’s entropy

\[ H(S) = -\sum_{c \in C} p(c) \log(p(c)) \]

Node training

\[ \theta^* = \arg \max_{\theta \in \mathcal{T}} I \]
Trees vs Forests

- A single tree may over fit to the training data
- Instead train multiple trees and combine their predictions
- Each tree can differ in both training data and node tests
- Achieve this by injecting randomness into training algorithm
Forests and trees

A forest is an ensemble of trees. The trees are all slightly different from one another.
(1) Bagging (randomizing the training set)

$S_0$  The full training set

$S_0^t \subseteq S_0$  The randomly sampled subset of training data made available for the tree $t$
Classification forest - injecting randomness II

(2) Node training - random subsets of node tests available at each node

Node weak learner

\[ h(v, \theta_j) \]

Node test params

\[ \theta_j \in \mathcal{T}_j \]

Forest training

\[ T_1 \]

\[ T_2 \]

\[ T_3 \]
From a single tree classification ....

What do we do at the leaf?

Prediction model: probabilistic

\[ p(c|v) \]
How to combine the predictions of the trees?

The ensemble model

Forest output probability

\[ p(c|v) = \frac{1}{T} \sum_{t=1}^{T} p_t(c|v) \]
Random Forests - summary

- Node tests are usually chosen to be cheap to evaluate (weak classifiers) e.g. comparing pairs of feature components
- At run time, classification is very fast (like AdaBoost)
- Classifier is non-linear in feature space
- Training can be quite fast as well if node test chosen completely randomly
- Many parameters: depth of tree, number of trees, type of node tests, random sampling
- Requires a lot of training data
- Large memory footprint (cf. k-NN)
Application:

Body tracking in Microsoft Kinect for XBox 360
Kinect – random forest classifier

Input data for each frame

RGB image

Depth image

Output

Multi-class classification

inferred body parts
Goal: train a random forest classifier to predict body parts

fit stickman model and track skeleton
Training/test Data

From motion capture system

e.g. 1 million training examples
Input data point \( v = (x, y) \) (pixel position in 2D image)

Output \( c \in \{ \text{left-hand, right-hand, head, r.-foot, r.-shoulder...} \} \)

Feature response \( f(v, \theta_j) = I(v) - I(p_j) \) (depth difference between two points)
Example result

Input depth image (bg removed)  Inferred body parts (posterior)

body parts in 3D
Neural Networks
Neural Network (NN) Overview

• A natural multiple class classifier

• Start with single neuron, and then move onto a network
Perceptron steps:

1. Map a vector $x$ to a scalar score by an affine projection $(w, b)$
2. Transform the score monotonically but non-linearly by the sigmoid $S()$

$P(y = 1 | x, w, b) = \frac{1}{1 + e^{-\sum_{i=1}^{D} w_i x_i + b}}$

$S(z) = \frac{1}{1 + e^{-z}}$
Training as a binary classifier – Logistic Regression

\[ f(x) = w^\top x + b \]

Learning is formulated as the optimization problem

\[
\min_{w \in \mathbb{R}^d} \sum_{i=1}^{N} \log (1 + e^{-y_if(x_i)}) + \lambda \|w\|^2
\]

Comparison with SVM:
- both approximate 0-1 loss
- both convex optimizations
- very similar asymptotic behaviour
- main difference is smoothness of LR, and non-zero outside SVM margin
Neural Networks

Assemble neurons into feed forward networks

- More parameters and non-linearities
- More capacity for learning – can train multi-way classifier
- No longer a convex optimization problem – but optimize using steepest descent on w

Deep architectures – multi-layer perceptron
Example: Convolutional Neural Networks (CNN)

- Choice of network architecture and loss function
- "Feed forward" network – multiple layers from input to output
- In a CNN, the architecture is convolutional
- In this example, consider a loss function for multi-way classification of images

\[
\sum_{i=1}^{N} l(f(x_i, w), y_i) + \lambda R(w)
\]

Minimize with respect to \( w \)

LeCun
Hinton
Krizhevsky
Convolutional layers

- Linear filters
- Down-sampling
- Non-linear gating
- Spatial pooling
- Channel normalisation

Slide credit: Andrea Vedaldi
Convolution

input features  a bank of 2 filters  2-dimensional output features
Filter bank example

A bank of 256 filters (learned from data)

Each filter is 1D (it applies to a grayscale image)

Each filter is $16 \times 16$ pixels
Filtering

Each filter generates a feature map.
CNN components

- Linear 3D filters: $\mathbf{x} \rightarrow (F, b) \rightarrow \mathbf{y} = F \ast \mathbf{x} + b$
- Downsampling: $\mathbf{x} \rightarrow \downarrow \rightarrow \mathbf{y}$
- ReLU: $\mathbf{x} \rightarrow \mathbf{y} = \max\{0, \mathbf{x}\}$
- Spatial pooling: $\mathbf{x} \rightarrow \text{max} \rightarrow y_{ijk} = \max_{pq \in \Omega_{ij}} x_{pqk}$
- Normalization: $\mathbf{x} \rightarrow \text{sliding l}^2 \rightarrow \mathbf{y}$

slide credit: Andrea Vedaldi
Convolutional layers

(F, b) → \( \max(0, z) \) → max pooling → sliding \( l^2 \) normalisation → code
Learning a CNN

Stochastic gradient descent
(with momentum, dropout, ...)

argmin $E(w_1, w_2, \ldots, w_8)$

slide credit: Andrea Vedaldi
Application: recognizing object categories

ImageNet Competition
– 1000 categories

- Flute
- Strawberry
- Traffic light
- Backpack
- Matchstick
- Sea lion
- Bathing cap
- Racket
ImageNet Competition

- 1K classes
- ~ 1K training images per class
- ~ 1M training images
- 100k test images

ImageNet challenge (ILSVRC), 1000 categories
Top-5 classification accuracy

Supervised CNN – summary

• Learn everything for the task at hand (including feature vectors)

• All parameters are learnt by back-propagation

• Excellent performance

• Quite fast at test time

• Many parameters: e.g. 500M weights

• Requires a lot of training data, and “tricks” in training such as data augmentation and drop-out

• Not a convex optimization problem

• Training can be slow – days or weeks on a GPU

• Large memory footprint at run time, e.g. more than 1GB
More …

- **Other regressors**
  - e.g. Gaussian process regression, random forests

- **Different lassos**
  - e.g. group lasso

- **Other loss functions**
  - ranking loss
  - dimensionality reduction

- **Other NN architectures**
  - e.g. RNN, LSTM

More on web page: [http://www.robots.ox.ac.uk/~az/lectures/ml](http://www.robots.ox.ac.uk/~az/lectures/ml)