Class #6: Non-linear classification

ML4Bio 2012
February 17th, 2012
Quaid Morris
You are free:

- to Share — to copy, distribute and transmit the work
- to Remix — to adapt the work

Under the following conditions:

- Attribution. You must attribute the work in the manner specified by the author or licensor (but not in any way that suggests that they endorse you or your use of the work).
- Share Alike. If you alter, transform, or build upon this work, you may distribute the resulting work only under the same or similar licence to this one.
- For any reuse or redistribution, you must make clear to others the licence terms of this work.
- Any of the above conditions can be waived if you get permission from the copyright holder.
- The author’s moral rights are retained in this licence.

Your fair dealing and other rights are in no way affected by the above.
This is a human-readable summary of the Legal Code (the full licence) available in the following languages:
English French

Learn how to distribute your work using this licence
Overview

• Review
• Linear separability
• Non-linear classification
• Linear Support Vector Machines (SVMs)
  – Picking the best linear decision boundary
• Non-linear SVMs
• Decision Trees
• Ensemble methods: Bagging and Boosting
Objective functions

- $E(X, \Theta)$ -- function of the data ($X$) and the parameters ($\Theta$)

- They measure data fit between a model and the data, e.g.:
  - The K-means objective function is the sum of the squared Euclidean distance from each data point to the closest “mean” (or “centroid”), parameters are the mean

- When fitting a statistical model, we use:
  - $E(X, \Theta) = \log P(X | \Theta) = \Sigma_i \log P(X_i | \Theta)$
Examples of objective functions

• Clustering:
  – Mix of Gaussians: log likelihood of data
  – Affinity propagation: sum of similarities with exemplars

• Regression:
  – Linear: Sum of squared errors (aka residuals), aka log likelihood under Gaussian noise

• Classification:
  – Logistic regression: Sum of log probability of class under logistic function
  – Fisher’s: ratio of $\sigma^2_{\text{within class}}$ vs $\sigma^2_{\text{between class}}$
Overfitting

- More complex models better fit the training data.
- But after some point, more complex models generalize worse to new data.

![Graph showing the relationship between error and complexity](image)

- Error (i.e. 1 - model fit)
- Generalization (test set) error
- Training set error
- # of parameters or "complexity"
Dealing with overfitting by regularization

• Basic idea: add a term to the objective function that penalizes # of parameters or model complexity,
• Now: $E(X, \Theta) = \text{datafit}(X, \Theta) - \lambda \text{ complexity}(\Theta)$
• “Hyper-parameter” $\lambda$ controls the strength of regularization – could have a natural value, or need to set this by cross-validation,
• Increases in model complexity need to be balanced by improved model fit. (each unit of added complexity costs $\lambda$ units of data fit)
Examples of regularization

• Clustering: (N is # params, M is # datapoints)
  – Bayes IC: LL – 0.5 N log M
  – Akaike IC: LL – N

• Regression:
  – L1 (aka LASSO): LL – λ ∑ |b_i|
  – L2 (aka Ridge): LL – λ ∑ b_i^2
  – Elastic net: LL – λ_1 ∑ |b_i| - λ_2 ∑ b_i^2

• Classification:
  – Logistic regression: Same as linear regression
Linear classification

f(x) = x^Tb
“discriminant function (or value)"

Can define decision boundary with a vector of feature weights, \( b \), and a threshold, \( t \), if \( x^Tb > t \), predict + and otherwise predict -, can trade off false positives versus false negatives by changing \( t \).
Two classes are **linearly separable**, if you can perfectly classify them with a linear decision boundary.

Why is this important? Linear classifiers can only draw linear decision boundaries.
Non-linear classification

Non-linear classifiers have non-linear, and possibly discontinuous decision boundaries.
Non-linear decision boundaries

Set $x_3 = (x_1 - m_1)^2 + (x_2 - m_2)^2$

Decision rule:
Predict $+$ if $x_3 < 1$
Non-linear classification: Idea #1

• Fit a linear classifier to non-linear functions of the input features.
• E.g.: Transform features (quadratic):
  \[ \{1, x_1, x_2\} \rightarrow \{1, x_1, x_2, x_1x_2, x_1^2, x_2^2\} \]
  Now can fit arbitrary quadratic decision boundaries by fitting a linear classifier to transformed features.

• This is feasible for quadratic transforms, but what about other transforms (“power of 10”)?
Support Vector Machines

\[ R(T) \leq R_{emp}(T) + \frac{\ln N - h \eta}{\eta} \left( 1 + \sqrt{\frac{2R_{emp}(T)\eta}{\ln N - h\eta}} \right) \]

ALL YOUR BAYES ARE BELONG TO US

Vladimir Vapnik
Picking the best decision boundary

What’s the best decision boundary to choose when one? That is, what is the one most likely to generalize well to new datapoints?
LDA & Fisher sol’n (generative)

Best decision boundary (in 2-d): parallel to the direction of greatest variance.

This is the Bayes optimal classifier (if mean and covariance are correct).
Unregularized logistic regression (discriminative)

All solutions are equally good because each one perfectly discriminates the classes. (note: this changes if there’s regularization)
Linear SVM solution

Maximum margin boundary.

Vladimir Vapnik
Here the maximum margin boundary is specified by three points, called the support vectors.
**SVM objective function**

(Where targets: \( y_i = -1 \) or \( y_i = 1 \), \( x_i \) input features, \( b \) are feature weights)

- “Primal” objective function:
  
  \[
  E(b) = -C \sum_i \xi_i - \frac{b^T b}{2}
  \]

  where \( y_i (b^T x_i) > 1 - \xi_i \) for all \( i \)

  \( \xi_i \geq 0 \) is a degree of “mis-classification” under “hinge loss” for datapoint \( i \), \( C > 0 \) is the hyperparameter balancing regularization and data fit.
Visualization of $\xi_i$

If linearly separable, all values of $\xi_i = 0$
Another SVM objective function

- “Dual”: \[ E(\alpha) = \sum_i \alpha_i - \sum_i \sum_j \alpha_i \alpha_j y_i y_j x_i^T x_j / 2 \]

Under the constraints that:
\[ \sum_i \alpha_i y_i = 0 \quad \text{and} \quad 0 \leq \alpha_i \leq C \]

Then can define weights \( b = \sum_i \alpha_i y_i x_i \)
and *discriminant function*: \( b^T x = \sum_i \alpha_i y_i x_i^T x \)

- Data \( i \) such that \( \alpha_i > 0 \) are support vectors
Dot products of transformed vector as “kernel functions”

- Let $x = (x_1, x_2)$ and $\phi^2(x) = (\sqrt{2}x_1x_2, x_1^2, x_2^2)$
- Here $\phi_2(x)$ [or $\phi_p(x)$] is a vector valued function that returns all possible powers of two [or $p$] of $x_1$ and $x_2$
- Then $\phi_2(x)^T\phi_2(z) = (x^Tz)^2 = K(x,z)$ “kernel function”
- And in general, $\phi_p(x)^T\phi_p(z) = (x^Tz)^p$
- Every kernel function $K(x,z)$ that satisfies some simple conditions corresponds to a dot product of some transformation $\phi(x)$ (aka “projection function”)
Non-linear SVM objective function

- Objective: $E(\alpha) = \sum_i \alpha_i - \sum_i \sum_j \alpha_i \alpha_j y_i y_j K(x_i, x_j) / 2$

Under the constraints that:
$\sum_i \alpha_i y_i = 0$ and $0 \leq \alpha_i \leq C$

- Discriminant function: $f(x) = \sum_i \alpha_i y_i K(x_i, x)$

\[ w_i \text{ feature weight} \]
Kernelization (FYI)

• Often instead of explicitly writing out the non-linear feature set, one simply calculates a kernel function of the two input vectors, i.e., \( K(x^i, x^j) = \phi(x^i)^T \phi(x^j) \)

• Here’s why: any kernel function \( K(x^i, x^j) \) that satisfies certain conditions, e.g., \( K(x^i, x^j) \) is a symmetric positive semi-definite function (which implies that the matrix \( K \), where \( K_{ij} = K(x^i, x^j) \) is a symmetric positive semi-definite matrix), corresponds to a dot product \( K(x^i, x^j) = \phi(x^i)^T \phi(x^j) \) for some projection function \( \phi(x) \).

• Often it’s easy to think of defining a kernel function that captures some notation of “similarity”

• Non-linear SVMs use the discriminant function
  \[
  f(x; w) = \sum_i w_i K(x^i, x)
  \]
Some popular kernel functions

- \( K(x_i, x_j) = (x_i^T x_j + 1)^P \)
  - Inhomogeneous Polynomial kernel of degree \( P \)
- \( K(x_i, x_j) = (x_i^T x_j)^P \)
  - Homogeneous Polynomial kernel
- \( K(x_i, x_j) = \exp\left\{ -(x_i - x_j)^T (x_i - x_j) / s^2 \right\} \)
  - Gaussian “radial basis function” kernel
- \( K(x_i, x_j) = \text{Pearson}(x_i, x_j) + 1 \)
  - “Bioinformatics” kernel
- Also, can make a kernel out of an interaction network!
Example: radial basis function kernels

\[ \Phi(x) = \{N(x; x^1, \sigma^2I), N(x; x^2, \sigma^2I), \ldots, N(x; x^2, \sigma^2I)\} \]

Where \( N(x; m, \Sigma) \) is the probability density of \( x \) under a multivariate Gaussian with mean \( m \), and covariance \( \Sigma \).
SVM summary

A “sparse” linear classifier that uses as features “kernel functions” that measure similarity to each data point in the training.
Neural Networks
(aka multilayer perceptrons)

\[ f(x, \Theta) = \sum_k v_k h_k(x) \]

Where “hidden unit”:
\[ h_k(x) = \sigma(\sum_i w_{ik} x_i) \]

Often:
\[ \sigma(x) = \frac{1}{1 + e^{-x}} \]

\( v \) and \( W \) are parameters

Notes:
- Fitting the hidden units has become a lot easier using deep belief networks
- Could also fit means of RBF for hidden units
K-Nearest Neighbour

Classify $x$ in the same way that a majority of the $K$ nearest neighbours to $x$ are labeled.

Can also used # of nearest neighbors that are positive as a discriminant value.

Smaller values of $K$ lead to more complex classifiers, you can set $K$ using (nested) cross-validation.

Images from Elements of Stat Learning, Hastie, Tibshirani, Efron (available online)
Decision trees

For each leaf:
1) Choose a dimension to split along, and choose best split using a “split criterion” (see below)
2) Split along dimension, creating two new leaves, return to (1) until leaves are pure.

To reduce overfitting, it is advisable to prune the tree by removing leaves with small numbers of examples.

Split criterion:
- C4.5 (Quinlan 1993): Expected decrease in (binary) entropy
- CART (Breiman et al 1984): Expected decrease in Gini impurity (1-sum of squared class probabilities)
Decision trees

\[ x_2 > a? \]

- yes
- no
Decision trees

\[ x_2 > a? \]
- yes
- no

\[ x_2 > b? \]
- no
- yes

etc
Decision trees

\[
x_2 > a? \\
\text{yes} \\
\text{no}
\]

\[
x_2 > b? \\
\text{no} \\
\text{yes}
\]

\text{etc}
Decision tree summary

Decision trees learn a recursive splits of the data along individual features that partition the input space into “homogeneous” groups of data points with the same labels.
Ensemble classification

• Combining multiple classifiers together by training them separately and then averaging their predictions is a good way to avoid overfitting.

• **Bagging** (Breiman 1996):
  – Resample training set, train separate classifiers on each sample, have the classifiers vote for the classification

• **Boosting** (e.g. Adaboost, Freund and Schapire 1997):
  – Iteratively reweight training sets based on errors of a weighted average of classifiers:
    • Train classifier (“weak learner”) to minimize weighted error on training set
    • Weight new classifier according to prediction error, reweight training set according to prediction error
    • Repeat
  – Minimizes exponential loss on training set over a convex set of functions
Bagging

$(x_1, y_1), (x_2, y_2), \ldots, (x_N, y_N)$

Bootstrap samples

Train separate classifiers

$(x_1, y_1), (x_1, y_1), (x_3, y_3), \ldots, (x_{N-1}, y_{N-1}) \rightarrow f_1(x)$

$(x_1, y_1), (x_2, y_2), (x_4, y_4), \ldots, (x_N, y_N) \rightarrow f_2(x)$

etc (M samples in total)

$\text{bagged}(x) = \sum_j f_j(x) / M$
Boosting

$$(x_1, y_1), (x_2, y_2), \ldots, (x_N, y_N)$$

Train classifier

$$f_1(x)$$

Assess performance

$$(x_1, y_1), (x_2, y_2), \ldots, (x_N, y_N)$$

Resample “hard” cases, Set $w_t$

Assess performance

$$(x_2, y_2), (x_2, y_2), \ldots, (x_{N-1}, y_{N-1})$$

Train classifier

$$f_t(x)$$

Assess performance

$$(x_2, y_2), (x_2, y_2), \ldots, (x_{N-1}, y_{N-1})$$

Legend

$$(x_i, y_i)$$ -- correct

$$(x_i, y_i)$$ -- incorrect

$$\text{boost}(x) = \sum_j w_j f_j(x)$$
Ensemble learning summary

• **Bagging:** classifiers trained separately (in parallel) on different bootstrapped samples of the training set, and make equally weighted votes

• **Boosting:** classifiers trained sequentially, weighted by performance, on samples of the training set that focus on “hard examples”. Final classification is based on weighted votes.
Random Forests (Breiman 2001)

- Construct M bootstrapped samples of the training set (of size N)
- For each sample, build DT using CART (no pruning), but split optimally on randomly chosen features – random feature choice reduces correlation among trees, this is a good thing.
- Since bootstrap resamples training set with replacement, leaves out, on average, \((1-1/N)^N \times 100\%\) of the examples (~\(100/e\% = 36.7\%\)), can use these out-of-bag samples to estimate performance
- Bag predictions (i.e. average them)
- Can assess “importance” of features by evaluating performance of trees containing those features
Advantages of Decision Trees

- Relatively easy to combine continuous and discrete (ordinal or categorical) features within the same classifier, this is harder to do for other methods.
- Random Forests can be trained in parallel (unlike boosted decision trees), and relatively quickly.
- Random Forests tend to do quite well in empirical tests (but see “No Free Lunch” theorem).
Networks as kernels

- Can use matrix representations of graphs to generate kernels.
- One popular graph-based kernel is the diffusion kernel:
  - \( K = (\lambda I - L)^{-1} \)
  - Where \( L = D - W \), \( D \) is a diagonal matrix with the row sums, and \( W \) is the matrix representation of the graph.
- GeneMANIA label propagation:
  - \( f = (\lambda I - L)^{-1}y \)