Statistics 202: Data Mining

Support Vector Machine, Random Forests, Boosting
Based in part on slides from textbook, slides of Susan Holmes

©Jonathan Taylor

December 2, 2012
Neural networks

Neural network

- Another classifier (or regression technique) for 2-class problems.
- Like logistic regression, it tries to predict labels $y$ from $x$.
- Tries to find

$$
\hat{f} = \arg\min_{f \in \mathcal{C}} \sum_{i=1}^{n} (Y_i - f(X_i))^2
$$

where

$$
\mathcal{C} = \{ \text{“neural net functions”} \}.
$$
Neural networks: single layer

Artificial Neural Networks (ANN)

Input nodes

Output node

$X_1$, $X_2$, $X_3$  

Black box

$Y$

$t=0.4$

0.3

$\Sigma$

© Tan, Steinbach, Kumar
Introduction to Data Mining
4/18/2004
Neural networks

Single layer neural nets

- With $p$ inputs, there are $p + 1$ weights

$$f(x_1, \ldots, x_p) = S \left( \alpha + x^T \beta \right)$$

- Here, $S$ is a sigmoid function like the logistic curve.

$$S(\eta) = \frac{e^\eta}{1 + e^\eta}.$$  

- The algorithm must estimate $(\alpha, \beta)$ from data.
- Classifier would assign $+1$ to anything with $S(\hat{\alpha} + x^T \hat{\beta}) > t$ for some threshold like 0.5.
Neural networks: double layer
Neural networks

Single layer neural nets

- In the previous figure, there are 15 weights and 3 intercept terms (α’s) for input → hidden layer.
- There are 3 weights and one intercept term for hidden → output layer.
- The outputs from the hidden layer have the form
  \[ S \left( \alpha_j + \sum_{i=1}^{5} w_{ij} x_i \right) \quad 1 \leq j \leq 3. \]

- So, the final output is of the form
  \[ S \left( \alpha_0 + \sum_{j=1}^{3} v_j S \left( \alpha_j + \sum_{i=1}^{5} w_{ij} x_i \right) \right). \]
Neural networks

Single layer neural nets

- Fitting this model is done via nonlinear least squares.
- Computationally difficult due to the highly nonlinear form of the regression function.
Support vector machines

Support vector machine

- Another classifier (or regression technique) for 2-class problems.
- Like logistic regression, it tries to predict labels $y$ from $x$ using a linear function.
- A linear function with an intercept $\alpha$ determines an affine function $f_{\alpha,\beta}(x) = x^T \beta + \alpha$ and a hyperplane

$$H_{\alpha,\beta} = \left\{ x : x^T \beta + \alpha = 0 \right\}.$$ 

- A good classifier will “separate the classes” into the sides where $f_{\alpha,\beta}(x) > 0$, $f_{\alpha,\beta}(x) < 0$.
- It seeks to find the hyperplane that “separates the classes the most.”
Support vector machine

Find a linear hyperplane (decision boundary) that will separate the data.
Support vector machine
Support vector machine
Support vector machine
Support vector machine

Which one is better? B1 or B2?

How do you define better?
Support vector machine

B_1

B_2

margin

b_{11}
b_{12}
b_{21}
b_{22}
Support vector machines

Support vector machine

- The reciprocal maximum margin can be written as

\[ \inf_{\beta, \alpha} \| \beta \|_2 \]

subject to \( y_i(x_i^T \beta + \alpha) \geq 1 \).

- Therefore, the support vector machine problem is

\[ \text{minimize}_{\beta, \alpha} \| \beta \|_2 \]

subject to \( y_i(x_i^T \beta + \alpha) \geq 1 \).
Support vector machine

What if the problem is not linearly separable?
Support vector machines

### Non-separable problems

- If we cannot completely separate the classes it is common to modify the problem to

\[
\text{minimize}_{\beta, \alpha, \xi} \|\beta\|_2
\]

subject to \( y_i(x_i^T \beta + \alpha) \geq 1 - \xi_i, \xi_i \geq 0, \sum_{i=1}^{n} \xi_i \leq C \)

- Or the Lagrange version

\[
\text{minimize}_{\beta, \alpha, \xi} \|\beta\|_2^2 + \gamma \sum_{i=1}^{n} \xi_i
\]

subject to \( y_i(x_i^T \beta + \alpha) \geq 1 - \xi_i, \xi_i \geq 0. \)
Non-separable problems

- The $\xi_i$’s can be removed from this problem, yielding

\[
\min_{\beta, \alpha} \|\beta\|_2^2 + \gamma \sum_{i=1}^{n} (1 - y_i f_{\alpha, \beta}(x_i))^+ + \lambda \|\beta\|_2^2
\]

where $(z)_+ = \max(z, 0)$ is the positive part function.

- Or,

\[
\min_{\beta, \alpha} \sum_{i=1}^{n} (1 - y_i f_{\alpha, \beta}(x_i))^+ + \lambda \|\beta\|_2^2
\]
Logistic vs. SVM

![Logistic vs. SVM graph](image_url)
Iris data: sepal.length, sepal.width
Iris data: sepal.length, sepal.width
Iris data: sepal.length, sepal.width
Support vector machines

Kernel trick

- The term $\|\beta\|^2_2$ can be thought of as a complexity penalty on the function $f_{\alpha, \beta}$ so we could write

$$\min_{f_{\beta, \alpha}} \sum_{i=1}^{n} (1 - y_i (\alpha + f_{\beta}(x_i)))_+ + \lambda \|f_{\beta}\|^2_2$$

- We could also add such a complexity penalty to logistic regression

$$\min_{f_{\beta, \alpha}} \sum_{i=1}^{n} \text{DEV}(\alpha + f_{\beta}(x_i), y_i) + \lambda \|f_{\beta}\|^2_2$$

- So, in both SVM and penalized logistic, we are looking for the “best” linear function where “best” is determined by either logistic loss (deviance) or the hinge loss (SVM).
Support vector machines

Kernel trick

- If the decision boundary is nonlinear, one might look for functions other than linear.
- There are classes of functions called *Reproducing Kernel Hilbert Spaces* (RKHS) for which it is theoretically (and computationally) possible to solve

\[
\minimize_{f \in \mathcal{H}} \sum_{i=1}^{n} (1 - y_i(\alpha + f(x_i)))_+ + \lambda \|f\|_{\mathcal{H}}^2.
\]

or

\[
\minimize_{f \in \mathcal{H}} \sum_{i=1}^{n} \text{DEV}(\alpha + f(x_i), y_i) + \lambda \|f\|_{\mathcal{H}}^2.
\]

- The `svm` function in R allows some such “kernels” for SVM.
Iris data: sepal.length, sepal.width
Iris data: sepal.length, sepal.width
Iris data: sepal.length, sepal.width
Iris data: sepal.length, sepal.width
Iris data: sepal.length, sepal.width
Iris data: sepal.length, sepal.width

SVM classification plot
Ensemble methods

General Idea

Step 1: Create Multiple Data Sets

Step 2: Build Multiple Classifiers

Step 3: Combine Classifiers

Original Training data

D

D_1

D_2

... 

D_{t-1}

D_t

C_1

C_2

C_{t-1}

C_t

C^*
Ensemble methods

**Why use ensemble methods?**

- Basic idea: weak learners / classifiers are *noisy learners*.
- Aggregating over a large number of weak learners should improve the output.
- Assumes some “independence” across weak learners, otherwise aggregation does nothing.
- That is, if every learner is the same, then any sensible aggregation technique should return this classifier.
- These are often some of the best classifiers in terms of performance.
- Downside: they have lost interpretability in the aggregation stage.
Ensemble methods

Bagging

- In this method, one takes several bootstrap samples (samples with replacement) of the data.
- For each bootstrap sample $S_b, 1 \leq b \leq B$, fit a model, retaining the classifier $f^{*,b}$.
- After all models have been fit, use majority vote

$$\hat{f}(x) = \text{majority vote of } (f^{*,b}(x))_{1 \leq i \leq B}.$$
Ensemble methods

### Bagging

- Approximately 1/3 of data is not selected in each bootstrap sample (recall 0.632 rule).
- Using this observation, we can define the *Out-of-bag (OOB)* estimate of accuracy based on OOB prediction

\[
\hat{f}_{OOB}(x_i) = \text{majority vote of } (f^b(x_i))_{1 \leq b \leq B(i)}
\]

where \(B(i) = \{\text{all trees not built with } x_i\}\).
- Total OOB error is

\[
OOB = \frac{1}{n} \sum_{i=1}^{n} (\hat{f}_{OOB}(x_i) \neq y_i)
\]
Ensemble methods

Random forests: bagging trees

- If the weak learners are trees, this method is called a Random Forest.
- Some variations on random forests inject more randomness than just bagging.
- For example:
  - Taking a random subset of features.
  - Random linear combinations of features.
Iris data: sepal.length, sepal.width using randomForest
Ensemble methods

Boosting

- Unlike bagging, boosting does not insert randomization into the algorithm.
- Boosting works by iteratively reweighting each observation and refitting a given classifier.
- To begin, suppose we have a classification algorithm that accepts case weights $w = (w_1, \ldots, w_n)$

$$\hat{f}_w = \text{argmin}_f \sum_{i=1}^{n} w_i L(y_i, f(x_i)).$$

and returns labels of $\{+1, -1\}$. 

```
**Ensemble methods**

**Boosting algorithm (AdaBoost.M1)**

**Step 0** Initialize weights $w_i^{(0)} = \frac{1}{n}, 1 \leq i \leq n$.

**Step 1** For $1 \leq t \leq T$, fit a classifier with weights $w^{(t-1)}$ yielding

$$
\hat{g}^{(t)} = \hat{f}_{w^{(t-1)}} = \arg\min_f \sum_{i=1}^n w_i^{(t-1)} L(y_i, f(x_i)).
$$

**Step 2** Compute the total error rate

$$
\text{err}^{(t)} = \frac{\sum_{i=1}^n w_i^{(t-1)} (y_i \neq \hat{g}^{(t)}(x_i))}{\sum_{i=1}^n w_i^{(t-1)}}
$$

$$
\alpha^{(t)} = \log \left( \frac{1 - \text{err}^{(t)}}{\text{err}^{(t)}} \right)
$$
Ensemble methods

Boosting algorithm (AdaBoost.M1)

Step 3 Produce new weights

\[ w_i^{(t)} = w_i^{(t-1)} \cdot \exp(\alpha^{(t)} \cdot (y_i \neq \hat{g}^{(t)}(x_i))) \]

Step 4 Repeat steps 1-3 until termination.

Step 5 Output the classifier

\[ \hat{f}(x) = \text{sign} \left( \sum_{t=1}^{T} \alpha^{(t)} \hat{g}^{(t)}(x) \right) \]
Ensemble methods

Initial weights for each data point

Data points for training

Original Data

Boosting Round 1

$\alpha = 1.9459$

Illustrating AdaBoost

Data points for training

Initial weights for each data point

$\alpha = 1.9459$

Data points for training
Ensemble methods

Illustrating AdaBoost

Boosting
Round 1

0.0094

+ + +

0.0094

+ + +

0.4623

- - -

\(\alpha = 1.9459\)

Boosting
Round 2

0.3037

- - -

0.0009

- - -

0.0422

+ +

\(\alpha = 2.9323\)

Boosting
Round 3

0.0276

+ + +

0.1819

+ + + +

0.0038

+ +

\(\alpha = 3.8744\)

Overall

+ + +

- - -

+ +
Ensemble methods

Boosting as gradient descent

- It turns out that boosting can be thought of as something like gradient descent.

- Above, we assumed we were solving

\[ \arg\min_f \sum_{i=1}^{n} w_i^{(t-1)} L(y_i, f(x_i)) \]

but we didn’t say what type of \( f \) we considered.
Ensemble methods

Boosting as gradient descent

- Call the space of all possible classifiers above $\mathcal{F}_0$, and all linear combinations of such classifiers

$$\mathcal{F} = \left\{ \sum_j \alpha_j f_j, f_j \in \mathcal{F}_0 \right\}$$

- In some sense, the boosting algorithm is a “steepest descent” algorithm to find

$$\arg\min_{f \in \mathcal{F}} \sum_{i=1}^{n} L(y_i, f(x_i))$$

- For more details, see ESL, but we will discuss some of these ideas. In R, this idea is implemented in the gbm package.
Boosting as gradient descent

- Consider the problem

\[
\text{minimize}_{f \in \mathcal{F}} = \sum_{i=1}^{n} L(y_i, f(x_i)) = L(f)
\]

- This can be solved by “gradient descent” where the “gradient” \(\nabla L(f)\) is in \(\mathcal{F}_0\) and is the direction of steepest descent for some stepsize \(\alpha\)

\[
\text{argmin}_{g \in \mathcal{F}_0} L(f + \alpha \cdot g)
\]
Ensemble methods

Boosting as gradient descent

- We might also optimize the stepsize

$$\arg\min_{g \in \mathcal{F}_0, \alpha} \mathcal{L}(f + \alpha \cdot g)$$
Ensemble methods

Boosting as gradient descent

- In this context, the \textit{forward stagewise} algorithm for steepest descent is:
  1. Initialize: $\hat{f}^{(0)}(x) = 0$;
  2. For $i = 1, \ldots, T$: solve
    \[
    \alpha^{(t)}, \hat{g}^{(t)} = \arg\min_{\alpha, g \in \mathcal{F}_0} \mathcal{L}(f^{(t-1)} + \alpha \cdot g)
    \]
    and update $f^{(t)} = f^{(t-1)} + \alpha^{(t)} \cdot \hat{g}^{(t)}$.
  3. Return the final classifier $f^{(T)}$.

- When $L(y, f(x)) = e^{-y \cdot f(x)}$ this corresponds to AdaBoost.M1.
- Changing the loss function yields new \textit{gradient boosting} algorithms.
Iris data: sepal.length, sepal.width, 1000 trees

{Versicolor} vs. {Setosa, Virginica}
Iris data: sepal.length, sepal.width, 5000 trees

{Versicolor} vs. {Setosa, Virginica}
Iris data: sepal.length, sepal.width, 10000 trees
Boosting fits an additive model (by default)
Boosting fits an additive model (by default)
Out-of-bag error improvement