Neural networks

- Another classifier (or regression technique) for 2-class problems.
- Like logistic regression, it tries to predict labels $y$ from $x$.
- Tries to find
  \[
  \hat{f} = \arg\min_{f \in C} \sum_{i=1}^{n} (Y_i - f(X_i))^2
  \]

where $C = \{ "neural\ net\ functions" \}$. 

Neural networks: single layer

- Black box model with input nodes $X_1, X_2, X_3$ and an output node $Y$.
- Connections with weights $0.3$.
Neural networks

**Single layer neural nets**

- With $p$ inputs, there are $p + 1$ weights
  \[ f(x_1, \ldots, x_p) = S(\alpha + x^T \beta) \]
- 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.

In the previous figure, there are 15 weights and 3 intercept terms (\(\alpha\)'s) for input $\rightarrow$ hidden layer. There are 3 weights and one intercept term for hidden $\rightarrow$ output layer. The outputs from the hidden layer have the form

\[ S(\alpha_j + \sum_{i=1}^{5} w_{ij}x_i) \quad 1 \leq j \leq 3. \]

So, the final output is of the form

\[ S(\alpha_0 + \sum_{j=1}^{3} v_j S(\alpha_j + \sum_{i=1}^{5} w_{ij}x_i)). \]
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

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

- 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 Machines

- 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 machines

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

\[
\min_{\beta, \alpha, \xi} \|\beta\|_2^2
\]

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

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

subject to

\[
y_i (x_i^T \beta + \alpha) \geq 1 - \xi_i, \xi_i \geq 0.
\]

The \( \xi_i \)'s can be removed from this problem, yielding

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

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

- Or,

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

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

or

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

• The 	exttt{svm} function in R allows some such "kernels" for SVM.
Iris data: sepal.length, sepal.width

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.

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

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 \( w = (w_1, \ldots, w_n) \)

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

and returns labels of \{+1, −1\}.

**Ensemble methods**

**Boosting algorithm (AdaBoost.M1)**

1. **Step 0** Initialize weights \( w_i^{(0)} = \frac{1}{n}, 1 \leq i \leq n. \)
2. **Step 1** For \( 1 \leq t \leq T \), fit a classifier with weights \( w^{(t-1)} \) yielding

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

3. **Step 2** Compute the total error rate

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

4. **Step 3** Produce new weights

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

5. **Step 4** Repeat steps 1-3 until termination.
6. **Step 5** Output the classifier

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

Illustrating AdaBoost

Data points for training

Initial weights for each data point

Original Data

Boosting Round 1

0.1 0.1 0.1

0.0094 0.0094 0.4623

α = 1.9459

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.

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 \texttt{gbm} package.
Ensemble methods

**Boosting as gradient descent**

- Consider the problem

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

- 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 \( \alpha \)

\[
\arg\min_{g \in \mathcal{F}_0} L(f + \alpha \cdot g)
\]

Ensemble methods

**Boosting as gradient descent**

- In this context, the *forward stagewise* algorithm for steepest descent is:
  1. Initialize: \( \hat{f}^{(0)}(x) = 0 \);
  2. For \( i = 1, \ldots, T \): solve

\[
\alpha^{(t)}, \hat{g}(t) = \arg\min_{\alpha, \hat{g} \in \mathcal{F}_0} L(f^{(t-1)} + \alpha \cdot \hat{g})
\]

and update \( f^{(t)} = f^{(t-1)} + \alpha^{(t)} \cdot \hat{g}(t) \).

  - 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 gradient boosting algorithms.

**Iris data:** sepal.length, sepal.width, 1000 trees

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

Boosting fits an additive model (by default)

Iris data: sepal.length, sepal.width, 10000 trees

Boosting fits an additive model (by default)
Out-of-bag error improvement