Machine Learning and Data Mining

Linear classification

Prof. Alexander Ihler
Supervised learning

- Notation
  - Features $x$
  - Targets $y$
  - Predictions $\hat{y}$
  - Parameters $\theta$

Program ("Learner")
Characterized by some "parameters" $\theta$
Procedure (using $\theta$) that outputs a prediction

Learning algorithm
Change $\theta$
Improve performance

Feedback / Target values
Score performance ("cost function")

Training data (examples)
Features
Linear regression

- Contrast with classification
  - Classify: predict discrete-valued target $y$

"Predictor":
Evaluate line:
$$r = \theta_0 + \theta_1 x_1$$

return $r$
Perceptron Classifier (2 features)

\[ f = \theta_1 x_1 + \theta_2 x_2 + \theta_0 \]

Visualizing for one feature “x”:

(c) Alexander Ihler
Perceptrons

- Perceptron = a linear classifier
  - The parameters $\theta$ are sometimes called weights ("w")
    - real-valued constants (can be positive or negative)
  - Define an additional constant input "1"

- A perceptron calculates 2 quantities:
  - 1. A weighted sum of the input features
  - 2. This sum is then thresholded by the $T(.)$ function

- Perceptron: a simple artificial model of human neurons
  - weights = “synapses”
  - threshold = “neuron firing”

(c) Alexander Ihler
Notation

• Inputs:
  - $x_0, x_1, x_2, \ldots, x_d,$
  - $x_1, x_2, \ldots, x_{d-1}, x_d$ are the values of the $d$ features
  - $x_0 = 1$ (a constant input)
  - $\mathbf{x} = (x_0, x_1, x_2, \ldots, x_d)$: feature vector (row vector)

• Weights (parameters):
  - $\theta_0, \theta_1, \theta_2, \ldots, \theta_d,$
  - we have $d+1$ weights
  - one for each feature + one for the constant
  - $\underline{\theta} = (\theta_0, \theta_1, \theta_2, \ldots, \theta_d)$: parameter vector (row vector)

• Linear response
  - $\theta_0 x_0 + \theta_1 x_1 + \ldots + \theta_d x_d = \underline{\theta} \cdot \mathbf{x}^\prime$ then threshold

(Matlab) \[ \text{>> f = th*x'}; \quad f = \text{sum(th.*x)}; \quad \text{yhat = sign(f);} \]
Perceptron Decision Boundary

- The perceptron is defined by the decision algorithm:

\[
o(x_1, x_2, \ldots, x_d, x_{d+1}) = \begin{cases} 
1 & \text{if } \theta \cdot x' > 0 \\
-1 & \text{otherwise}
\end{cases}
\]

- The perceptron represents a hyperplane decision surface in d-dimensional space
  - A line in 2D, a plane in 3D, etc.

- The equation of the hyperplane is given by

\[
\theta \cdot x' = 0
\]

This defines the set of points that are on the boundary.
Example, Linear Decision Boundary

\[ \theta = (\theta_1, \theta_2, \theta_0) = (.5, -.5, 0) \]
Example, Linear Decision Boundary

\[ \theta = (\theta_1, \theta_2, \theta_0) \]
\[ = (.5, -.5, 0) \]

\[ \theta \cdot x' = 0 \]
\[ \implies .5 \cdot x_1 - .5 \cdot x_2 + 0 \cdot 1 = 0 \]
\[ \implies -.5 x_2 = -.5 x_1 + 0 \]
\[ \implies x_2 = x_1 \]

From P. Smyth
Example, Linear Decision Boundary

\[ \theta = (\theta_1, \theta_2, \theta_0) \]
\[ = (0.5, -0.5, 0) \]

\[ \theta \cdot x' < 0 \]
\[ \Rightarrow x_1 < x_2 \] (this is the equation for decision region -1)

\[ \theta \cdot x' > 0 \]
\[ \Rightarrow x_1 > x_2 \] (decision region +1)

From P. Smyth
Separability

• A data set is separable by a learner if
  – There is some instance of that learner that correctly predicts all the data points
• Linearly separable data
  – Can separate the two classes using a straight line in feature space
  – In 2 dimensions the decision boundary is a straight line
Class overlap

- Classes may not be well-separated
- Same observation values possible under both classes
  - High vs low risk; features {age, income}
  - Benign/malignant cells look similar
  - ...
- Common in practice
- May not be able to perfectly distinguish between classes
  - Maybe with more features?
  - Maybe with more complex classifier?
- Otherwise, may have to accept some errors

(c) Alexander Ihler
Another example
Non-linear decision boundary

(c) Alexander Ihler
Representational Power of Perceptrons

- What mappings can a perceptron represent perfectly?
  - A perceptron is a linear classifier
  - thus it can represent any mapping that is linearly separable
  - some Boolean functions like AND (on left)
  - but not Boolean functions like XOR (on right)

(c) Alexander Ihler
Adding features

- Linear classifier can’t learn some functions

**1D example:**

$$y = T(b \cdot x + c)$$

Not linearly separable

Add quadratic features

$$y = T(a \cdot x^2 + b \cdot x + c)$$

Linearly separable in new features...
Adding features

- Linear classifier can’t learn some functions

1D example:

\[ y = T(b \cdot x + c) \]

Not linearly separable

Quadratic features, visualized in original feature space:

\[ y = T(a \cdot x^2 + b \cdot x + c) \]

More complex decision boundary: \( ax^2 + bx + c = 0 \)
Representational Power of Perceptrons

- What mappings can a perceptron represent perfectly?
  - A perceptron is a linear classifier
  - thus it can represent any mapping that is linearly separable
  - some Boolean functions like AND (on left)
  - but not Boolean functions like XOR (on right)

What kinds of functions would we need to learn the data on the right?

(c) Alexander Ihler
Representational Power of Perceptrons

- What mappings can a perceptron represent perfectly?
  - A perceptron is a linear classifier
  - thus it can represent any mapping that is linearly separable
  - some Boolean functions like AND (on left)
  - but not Boolean functions like XOR (on right)

What kinds of functions would we need to learn the data on the right?
Ellipsoidal decision boundary: \[ a x_1^2 + b x_1 + c x_2^2 + d x_2 + e x_1 x_2 + f = 0 \]
Feature representations

- Features are used in a linear way
- Learner is dependent on representation

Ex: discrete features
- Mushroom surface: \{fibrous, grooves, scaly, smooth\}
- Probably not useful to use \(x = \{1, 2, 3, 4\}\)
- Better: 1-of-K, \(x = \{[1000], [0100], [0010], [0001]\}\)
- Introduces more parameters, but a more flexible relationship
Effect of dimensionality

- Data are increasingly separable in high dimension – is this a good thing?

  - “Good”
    - Separation is easier in higher dimensions (for fixed # of data m)
    - Increase the number of features, and even a linear classifier will eventually be able to separate all the training examples!

  - “Bad”
    - Remember training vs. test error? Remember overfitting?
    - Increasingly complex decision boundaries can eventually get all the training data right, but it doesn’t necessarily bode well for test data…

### Graph:

- **Predictive Error**
  - Underfitting
  - Ideal Range
  - Overfitting

- **Complexity**
  - Error on Training Data
  - Error on Test Data

Ideal Range

Underfitting

Overfitting

Error on Training Data

Error on Test Data
Summary

• Linear classifier $\Leftrightarrow$ perceptron

• Linear decision boundary
  – Computing and visualizing

• Separability
  – Limits of the representational power of a perceptron

• Adding features
  – Interpretations
  – Effect on separability
  – Potential for overfitting
Machine Learning and Data Mining

Linear classification: Learning

Prof. Alexander Ihler
Learning the Classifier Parameters

- Learning from Training Data:
  - training data = labeled feature vectors
  - Find parameter values that predict well (low error)
    - error is estimated on the training data
    - “true” error will be on future test data

- Define an objective function $J(\theta)$:
  - Classifier accuracy (for a given set of weights $\theta$ and labeled data)

- Maximize this objective function (or, minimize error)
  - An optimization or search problem over the vector $(\theta_1, \theta_2, \theta_0)$
Training a linear classifier

• How should we measure error?
  – Natural measure = “fraction we get wrong” (error rate)
    $$\text{err}(@) = \frac{1}{m} \sum \delta(\hat{y}(i) \neq y(i))$$
    where $$\delta(\hat{y}(i) \neq y(i)) = 0$$ if $$\hat{y}(i) = y(i)$$, and 1 otherwise

(Matlab)
    $$>> yh = \text{sign}(th*X'); \text{err} = \text{mean}(y \neq yh);$$

• But, hard to train via gradient descent
  – Not continuous
  – As decision boundary moves, errors change abruptly

1D example:

\[ T(f) = \begin{cases} 
  -1 & \text{if } f < 0 \\
  +1 & \text{if } f > 0 
\end{cases} \]

(c) Alexander Ihler
Linear regression?

• Simple option: set $\theta$ using linear regression

• In practice, this often doesn’t work so well…
  – Consider adding a distant but “easy” point
  – MSE distorts the solution

(c) Alexander Ihler
Perceptron algorithm

- Perceptron algorithm: an SGD-like algorithm
  While (~done)
    For each data point j:
      \( \hat{y}(j) = T( \theta * x(j) ) \) : predict output for data point j
      \( \theta \leftarrow \theta + \alpha ( y(j) - \hat{y}(j) ) x(j) \) : “gradient-like” step

- Compare to linear regression + MSE cost
  - Identical update to SGD for MSE except error uses thresholded \( \hat{y}(j) \) instead of linear response \( \theta x' \) so:
    - (1) For correct predictions, \( y(j) - \hat{y}(j) = 0 \)
    - (2) For incorrect predictions, \( y(j) - \hat{y}(j) = \pm 2 \)

“adaptive” linear regression: correct predictions stop contributing

(c) Alexander Ihler
Perceptron algorithm

- Perceptron algorithm: an SGD-like algorithm

\[
\text{While (~done)}
\]

For each data point \( j \):

\[
\hat{y}(j) = T(\theta \ast x(j)) \quad \text{: predict output for data point } j
\]

\[
\theta \leftarrow \theta + \alpha (y(j) - \hat{y}(j)) x(j) \quad \text{:\text{ "gradient-like" step}}
\]

(c) Alexander Ihler
Perceptron algorithm

- Perceptron algorithm: an SGD-like algorithm

While (∼done)
  For each data point \( j \):
    \[
    \hat{y}(j) = T( \theta \cdot x(j) ) \\
    \theta \leftarrow \theta + \alpha \left( y(j) - \hat{y}(j) \right) x(j)
    \]
    : predict output for data point \( j \)
    : “gradient-like” step

\( y(j) \) predicted correctly: no update

(c) Alexander Ihler
Perceptron algorithm

- Perceptron algorithm: an SGD-like algorithm
  While (~done)
    For each data point \( j \):
      \[
      \hat{y}(j) = T( \theta \ast x(j) ) : \text{predict output for data point } j
      \]
      \[
      \theta \leftarrow \theta + \alpha ( y(j) - \hat{y}(j) ) x(j) : \text{“gradient-like” step}
      \]
      (Converges if data are linearly separable)
Surrogate loss functions

- Another solution: use a “smooth” loss
  - e.g., approximate the threshold function

- Usually some smooth function of distance
  - Example: “sigmoid”, looks like an “S”

- Now, measure e.g. MSE

\[
J(\theta) = \frac{1}{m} \sum_{j} \left( \sigma(f(x^{(i)})) - y^{(i)} \right)^2
\]

- Far from the decision boundary: |f(.)| large, small error
- Nearby the boundary: |f(.)| near 1/2, larger error

1D example:

Classification error = 2/9
MSE = \( (0^2 + 1^2 + .2^2 + .25^2 + .05^2 + \ldots)/9 \)
Beyond misclassification rate

- Which decision boundary is “better”?  
  - Both have zero training error (perfect training accuracy)  
  - But, one of them seems intuitively better…

- Side benefit of “smoothed” error function  
  - Encourages data to be far from the decision boundary  
  - See more examples of this principle later…

(c) Alexander Ihler
Training the Classifier

- Once we have a smooth measure of quality, we can find the “best” settings for the parameters of $f(X_1, X_2) = aX_1 + bX_2 + c$

- Example: 2D feature space $\leftrightarrow$ parameter space

(c) Alexander Ihler
Training the Classifier

• Once we have a smooth measure of quality, we can find the “best” settings for the parameters of
  \[ f(X_1, X_2) = aX_1 + bX_2 + c \]

• Example: 2D feature space \( \Leftrightarrow \) parameter space

(c) Alexander Ihler
Training the Classifier

- Once we have a smooth measure of quality, we can find the “best” settings for the parameters of
  \[ f(X_1,X_2) = aX_1 + bX_2 + c \]
- Finding the minimum loss \( J(.) \) in parameter space…
Finding the Best MSE

• As in linear regression, this is now just optimization

• Methods:
  – Gradient descent
    • Improve loss by small changes in parameters (“small” = learning rate)
  – Or, substitute your favorite optimization algorithm…
    • Coordinate descent
    • Stochastic search
    • Genetic algorithms
Gradient Equations

- \( J(\theta = [a, b, c]) = \frac{1}{m} \sum_{i} (\sigma(ax_1^{(i)} + bx_2^{(i)} + c) - y^{(i)})^2 \)

- What’s the derivative with respect to one of the parameters?

\[
\frac{\partial J}{\partial a} = \frac{1}{m} \sum_{i} 2(\sigma(\theta \cdot x^{(i)}) - y^{(i)}) \partial \sigma(\theta \cdot x^{(i)}) x_1^{(i)}
\]

Error between class and prediction

Sensitivity of prediction to changes in parameter “a”

- Similar for parameters b, c [replace \( x_1 \) with \( x_2 \) or 1 (constant)]

(c) Alexander Ihler
Saturating Functions

- Many possible “saturating” functions

- “Logistic” sigmoid (scaled for range \([0,1]\)) is
  \[
  \sigma(z) = \frac{1}{1 + \exp(-z)}
  \]

- Derivative is
  \[
  \partial \sigma(z) = \sigma(z) (1-\sigma(z))
  \]

- Matlab Implementation:

  ```matlab
  function s = sig(z)
  % value of [0,1] sigmoid
  s = 1 ./ (1+exp(-z));
  end
  
  function ds = dsig(x)
  % derivative of (scaled) sigmoid
  ds = sig(z) .* (1-sig(z));
  end
  ```

  For range \([-1 , +1]\):\
  \[
  \rho(z) = 2 \sigma(z) -1
  \]
  \[
  \partial \rho(z) = 2 \sigma(z) (1-\sigma(z))
  \]

  Predict: threshold \(z\) or \(\rho\) at zero

(c) Alexander Ihler
Logistic regression

- Interpret $\sigma(\theta x')$ as a probability that $y = 1$
- Use a negative log-likelihood loss function
  - If $y = 1$, cost is $-\log \Pr[y=1] = -\log \sigma(\theta x')$
  - If $y = 0$, cost is $-\log \Pr[y=0] = -\log (1 - \sigma(\theta x'))$

- Can write this succinctly:
  $$J(\theta) = -\frac{1}{m} \sum_i y^{(i)} \log \sigma(\theta \cdot x^{(i)}) + (1 - y^{(i)}) \log (1 - \sigma(\theta \cdot x^{(i)}))$$

(c) Alexander Ihler
Logistic regression

- Interpret \( \sigma(\theta x') \) as a probability that \( y = 1 \)
- Use a negative log-likelihood loss function
  - If \( y = 1 \), cost is \(-\log \Pr[y=1] = -\log \sigma(\theta x')\)
  - If \( y = 0 \), cost is \(-\log \Pr[y=0] = -\log (1 - \sigma(\theta x'))\)

- Can write this succinctly:

\[
J(\theta) = -\frac{1}{m} \sum_{i} y(i) \log \sigma(\theta \cdot x(i)) + (1 - y(i)) \log (1 - \sigma(\theta \cdot x(i)))
\]

- Convex! Otherwise similar: optimize \( J(\theta) \) via …

1D example:

Classification error = MSE = 2/9

NLL = - (log(.99) + log(.97) + …)/9
Gradient Equations

• Logistic neg-log likelihood loss:

\[ J(\theta) = -\frac{1}{m} \sum_{i} y^{(i)} \log \sigma(\theta \cdot x^{(i)}) + (1 - y^{(i)}) \log (1 - \sigma(\theta \cdot x^{(i)})) \]

• What’s the derivative with respect to one of the parameters?

\[ \frac{\partial J}{\partial a} = -\frac{1}{m} \sum_{i} y^{(i)} \frac{1}{\sigma(\theta \cdot x^{(i)})} \frac{\partial \sigma(\theta \cdot x^{(i)})}{\partial \theta} x^{(i)} + (1 - y^{(i)}) \ldots \]

\[ = -\frac{1}{m} \sum_{i} y^{(i)} (1 - \sigma(\theta \cdot x^{(i)})) x^{(i)}_{1} - (1 - y^{(i)}) \ldots \]
Surrogate loss functions

- Replace 0/1 loss $\Delta_i(\theta) = \delta(T(\theta x^{(i)}) \neq y^{(i)})$ with something easier:
  - Logistic MSE
    $$J_i(\theta) = 4(\sigma(\theta x^{(i)}) - y^{(i)})^2$$
  - Logistic Neg Log Likelihood
    $$J_i(\theta) = -\frac{y^{(i)}}{\log 2} \log \sigma(\theta \cdot x^{(i)}) + \ldots$$

(c) Alexander Ihler
Summary

• Linear classifier $\Leftrightarrow$ perceptron

• Measuring quality of a decision boundary
  – Error rate (0/1 loss)
  – Logistic sigmoid + MSE criterion
  – Logistic Regression

• Learning the weights of a linear classifier from data
  – Reduces to an optimization problem
  – Perceptron algorithm
  – For MSE or Logistic NLL, we can do gradient descent
  – Gradient equations & update rules