Machine Learning and Data Mining

Linear classification

Kalev Kask
Supervised learning

- Notation
  - Features $x$
  - Targets $y$
  - Predictions $\hat{y} = f(x; \theta)$
  - Parameters $\theta$

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

Learning algorithm
Change $\theta$
Improve performance

"train"

Training data (examples)
Features
Feedback / Target values

"predict"
Score performance ("cost function")
Linear regression

- Contrast with classification
  - Classify: predict discrete-valued target $y$
  - Initially: “classic” binary \{ -1, +1\} classes; generalize later

```
“Predictor”:
Evaluate line:
\[ r = \theta_0 + \theta_1 x_1 \]
return r
```
Perceptron Classifier (2 features)

Classifier $f(x; \theta)$

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

(weighted sum of the inputs)

Threshold Function

\[
    T(r) = \begin{cases} 
        -1, & r < 0 \\ 
        +1, & r > 0 
    \end{cases}
\]

output = class decision

$r = X\cdot\theta$; # compute linear response
$Y_{\text{hat}} = (r > 0)$ # predict class 1 vs 0
$Y_{\text{hat}} = 2*(r > 0) - 1$ # or ”sign”: predict +1 / -1

# Note: typically convert classes to ”canonical” values 0,1,…
# then convert back (“learner.classes[c]”) after prediction

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)
  - Input features $x_1 \ldots x_n$; 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 $T(\cdot)$ function

- Perceptron: a simple artificial model of human neurons
  - weights = “synapses”
  - threshold = “neuron firing”
The perceptron is defined by the decision algorithm:

\[ f(x; \theta) = \begin{cases} 
+1 & \text{if } \theta \cdot x^T > 0 \\
-1 & \text{otherwise}
\end{cases} \]

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

\[ \theta \cdot x^T = 0 \]

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

\[
\theta = (\theta_0, \theta_1, \theta_2) \\
= (1, .5, - .5)
\]
Example, Linear Decision Boundary

\[ \theta = (\theta_0, \theta_1, \theta_2) = (1, 0.5, -0.5) \]

\[ \theta \cdot x^T = 0 \]

\[ \Rightarrow 1 + 0.5 x_1 - 0.5 x_2 = 0 \]

\[ \Rightarrow -0.5 x_2 = -0.5 x_1 - 1 \]

\[ \Rightarrow x_2 = x_1 + 2 \]
Example, Linear Decision Boundary

\[ \theta = (\theta_0, \theta_1, \theta_2) = (1, 0.5, -0.5) \]

\[ \theta \cdot x^T = 0 \]

\[ \theta \cdot x^T < 0 \]

\[ \Rightarrow x_1 + 2 < x_2 \]

(this is the equation for decision region -1)

\[ \theta \cdot x^T > 0 \]

\[ \Rightarrow x_1 + 2 > x_2 \]

(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

(c) Alexander Ihler
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
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)
Adding features

- Linear classifier can’t learn some functions

1D example:

Not linearly separable

Add quadratic features

$x_2 = (x_1)^2$

Linearly separable in new features...
Adding features

- Linear classifier can’t learn some functions

1D example:

Not linearly separable

Quadratic features, visualized in original feature space:

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

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?
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…

![Predictive Error vs. Complexity Graph](chart.png)
Summary

• Linear classifier ⇔ 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

Kalev Kask
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 a loss function $J(\theta)$:
  - Classifier error rate (for a given set of weights $\theta$ and labeled data)

- Minimize this loss function (or, maximize accuracy)
  - 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)

\[
\text{err}(\theta) = \frac{1}{m} \sum_i 1[y^{(i)} \neq f(x^{(i)}; \theta)]
\]

where \[ 1[y \neq \hat{y}] = \begin{cases} 
1 & y \neq \hat{y} \\
0 & \text{o.w.} 
\end{cases} \]

\[
\text{Yhat} = \text{np.sign}(X\cdot\text{theta.T});
\]  
\[
\text{err} = \text{np.mean}(Y \neq \text{Yhat})
\]

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

1D example:

\[
T(f) = -1 \text{ if } f < 0
\]

\[
T(f) = +1 \text{ if } f > 0
\]
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 $j$:
          \[
          \hat{y}^{(j)} = \text{sign}(\theta \cdot x^{(j)})
          \]
          (predict output for 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 $\theta x^T$ 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
Perceptron algorithm

- Perceptron algorithm: an SGD-like algorithm

\[
\text{while } \neg \text{ done:} \\
\text{for each data point } j:\n\hat{y}(j) = \text{sign}(\theta \cdot x^{(j)}) \\
\theta \leftarrow \theta + \alpha(y^{(j)} - \hat{y}(j))x^{(j)}
\]

(predict output for point j)

("gradient-like" step)
Perceptron algorithm

• Perceptron algorithm: an SGD-like algorithm

while ¬ done:

    for each data point $j$:

        $\hat{y}(j) = \text{sign}(\theta \cdot x^{(j)})$
        (predict output for point $j$)

        $\theta \leftarrow \theta + \alpha(y^{(j)} - \hat{y}(j))x^{(j)}$
        ("gradient-like" step)
Perceptron algorithm

- Perceptron algorithm: an SGD-like algorithm

\[
\text{while } \neg \text{ done:} \\
\text{for each data point } j: \\
\hat{y}(j) = \text{sign}(\theta \cdot x^{(j)}) \\
\theta \leftarrow \theta + \alpha (y^{(j)} - \hat{y}(j)) x^{(j)}
\]

(predict output for point j)

(“gradient-like” step)

(Converges if data are linearly separable)
Machine Learning and Data Mining

Linear classification: Other Linear classifiers

Kalev Kask
Surrogate loss functions

- Another solution: use a “smooth” loss
  - e.g., approximate the threshold function
  - Usually some smooth function of distance
    - Example: logistic “sigmoid”, looks like an “S”
  - Now, measure e.g. MSE
    \[ J(\theta) = \frac{1}{m} \sum_j \left( \sigma( r(x^{(j)}) ) - y^{(j)} \right)^2 \]
  - Far from the decision boundary: \(|r(x)|\) large, small error
  - Nearby the boundary: \(|r(x)|\) near 1/2, larger error

\[
\text{Classification error} = 2/9
\]
\[
\text{MSE} = (0^2 + 1^2 + .2^2 + .25^2 + .05^2 + \ldots)/9
\]
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 many “smoothed” error functions
  – 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

\[ r(x_1, x_2) = a \cdot x_1 + b \cdot x_2 + c \]

- 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

\[ r(x_1, x_2) = a^* x_1 + b^* x_2 + c \]

- 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

\[ r(x_1, x_2) = a^* x_1 + b^* x_2 + c \]

- Example: 2D feature space \(\leftrightarrow\) parameter space

(c) Alexander Ihler
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

- **MSE** (note, depends on function \(\sigma(.)\))

\[
J(\theta = [a, b, c]) = \frac{1}{m} \sum_i (\sigma(ax^{(i)}_1 + bx^{(i)}_2 + c) - y^{(i)})^2
\]

- What's the derivative with respect to one of the parameters?
  - Recall the chain rule of calculus:

\[
\frac{\partial}{\partial a} f(g(h(a))) = f'(g(h(a))) \cdot g'(h(a)) \cdot h'(a)
\]

\[
f(g) = (g)^2 \quad \Rightarrow \quad f'(g) = 2(g)
\]

\[
g(h) = \sigma(h) - y \quad \Rightarrow \quad g'(h) = \sigma'(h)
\]

\[
h(a) = ax^{(i)}_1 + bx^{(i)}_2 + c \quad \Rightarrow \quad h'(a) = x^{(i)}_1
\]

\[
\frac{\partial J}{\partial a} = \frac{1}{m} \sum_i 2(\sigma(\theta \cdot x^{(i)}) - y^{(i)}) \cdot \partial \sigma(\theta \cdot x^{(i)}) \cdot x^{(i)}_1
\]

Error between class and prediction

Sensitivity of prediction to changes in parameter “a”

(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 (slope of the function at a point \( z \)) is
  \[ \partial \sigma(z) = \sigma(z) (1-\sigma(z)) \]

• Python Implementation:

```python
def sig(z):
    # logistic sigmoid
    return 1.0 / (1.0 + np.exp(-z))  # in [0,1]

def dsig(z):
    # its derivative at z
    return sig(z) * (1-sig(z))
```

(z = linear response, \( x^T \theta \))

(to predict: threshold \( z \) at 0 or threshold \( \sigma(z) \) at \( \frac{1}{2} \))

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 0
Machine Learning and Data Mining

Linear classification: Logistic Regression

Kalev Kask
**Logistic regression**

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

- Can write this succinctly:

$$J(\theta) = -\frac{1}{m} \left( \sum_i y^{(i)} \log \sigma(\theta \cdot x^{(i)}) + (1 - y^{(i)}) \log (1 - \sigma(\theta \cdot x^{(i)})) \right)$$

Nonzero only if $y=1$
Nonzero only if $y=0$
Logistic regression

• Interpret $\sigma(\theta x^T)$ as a probability that $y = 1$
• Use a negative log-likelihood loss function
  – If $y = 1$, cost is $-\log P(y=1) = -\log \sigma(\theta x^T)$
  – If $y = 0$, cost is $-\log P(y=0) = -\log (1 - \sigma(\theta x^T))$

• Can write this succinctly:

$$J(\theta) = -\frac{1}{m} \left( \sum_i y^{(i)} \log \sigma(\theta \cdot x^{(i)}) + (1 - y^{(i)}) \log (1 - \sigma(\theta \cdot x^{(i)})) \right)$$

• 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:

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

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

\[ \frac{\partial J}{\partial \alpha} = -\frac{1}{m} \left( \sum_i y^{(i)} \frac{1}{\sigma(\theta \cdot x^{(i)})} \cdot \partial \sigma(\theta \cdot x^{(i)}) \cdot x_1 \right) \]

\[ = -\frac{1}{m} \left( \sum_i y^{(i)} (1 - \sigma(\theta \cdot x^{(i)})) \cdot x_1 \right) \]
Surrogate loss functions

- Replace 0/1 loss \( \Delta_i(\theta) = 1 \left[ T(\theta x^{(i)}) \neq y^{(i)} \right] \) with something easier:

  - Logistic MSE
    \[
    J_i(\theta) = 4 \left( \sigma(\theta x^{(i)}) - y^{(i)} \right)^2
    \]

  - Logistic Neg Log Likelihood
    \[
    J_i(\theta) = -\frac{y^{(i)}}{\log 2} \log \sigma(\theta \cdot x^{(i)}) + \ldots
    \]
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

(c) Alexander Ihler
Multi-class linear models

- What about multiple classes? One option:
  - Define one linear response per class
  - Choose class with the largest response

\[ f(x; \theta) = \arg \max_c \theta_c \cdot x^T \]

- Boundary between two classes, \( c \) vs. \( c' \)?

\[
= \begin{cases} 
  c & \text{if } \theta_c \cdot x^T > \theta_{c'} \cdot x^T \iff (\theta_c - \theta_{c'})x^T > 0 \\
  c' & \text{otherwise}
\end{cases}
\]

- Linear boundary: \( (\theta_c - \theta_{c'})x^T = 0 \)
Multiclass linear models

- More generally, can define a generic linear classifier by

\[ f(x; \theta) = \arg \max_y \theta \cdot \Phi(x, y) \]

- Example: \( y = \{-1, +1\} \)

\[ \Phi(x, y) = y [1 \ x \ x^2 \ldots] \]

\[ f(x; \theta) = \begin{cases} +1 & \theta \cdot [1 \ x \ x^2 \ldots] > -\theta \cdot [1 \ x \ x^2 \ldots] \\ -1 & \text{o.w.} \end{cases} \]

(Standard perceptron rule)
Multiclass linear models

• More generally, can define a generic linear classifier by

$$f(x; \theta) = \arg\max_y \theta \cdot \Phi(x, y)$$

• Example: \( y = \{0, 1, 2, \ldots\} \)

\[
\Phi(x, y) = \begin{bmatrix}
\mathbbm{1}[y = 0][1 \ x \ x^2 \ \ldots] & \mathbbm{1}[y = 1][1 \ x \ x^2 \ \ldots] & \ldots
\end{bmatrix}
\]

\[
\theta = \begin{bmatrix}
[\theta_{00} \ \theta_{01} \ \theta_{02} \ \ldots] & [\theta_{10} \ \theta_{11} \ \theta_{12} \ \ldots] & \ldots
\end{bmatrix}
\]

(parameters for each class \( c \))

\[
f(x; \theta) = \arg\max_c \ \theta_c \cdot [1 \ x \ x^2 \ \ldots]
\]

(predict class with largest linear response)
Multiclass perceptron algorithm

- Perceptron algorithm:
  - Make prediction $f(x)$
  - Increase linear response of true target $y$; decrease for prediction $f$

While (~done)
  
  For each data point $j$:
  
  $f^{(j)} = \arg \max_c (\theta_c \cdot x^{(j)})$ : predict output for data point $j$

  $\theta_{f} \leftarrow \theta_{f} - \mathbb{1} x^{(j)}$ : decrease response of class $f^{(j)}$ to $x^{(j)}$

  $\theta_{y} \leftarrow \theta_{y} + \mathbb{1} x^{(j)}$ : increase response of true class $y^{(j)}$

- More general form update:

  $f(x; \theta) = \arg \max_y \theta \cdot \Phi(x, y)$

  $\theta \leftarrow \theta + \alpha \left( \Phi(x, y) - \Phi(x, f(x)) \right)$

(c) Alexander Ihler
Multilogit regression

- Define the probability of each class:

\[ p(Y = y | X = x) = \frac{\exp(\theta_y \cdot x^T)}{\sum_c \exp(\theta_c \cdot x^T)} \]

- Then, the NLL loss function is:

\[ J(\theta) = -\frac{1}{m} \sum_i \log p(y^{(i)} | x^{(i)}) = -\frac{1}{m} \sum_i \left[ \theta_{y^{(i)}} \cdot x^{(i)} - \log \sum_c \exp(\theta_c \cdot x^{(i)}) \right] \]

- P: “confidence” of each class
  - Soft decision value
- Decision: predict most probable
  - Linear decision boundary
- Convex loss function

(Y binary = logistic regression)