Linear Classifiers and the Linear SVM

D.A. Forsyth, UIUC

Linear Classifiers

- Key ideas
  - data (x_i, y_i)
    - x_i are feature vectors, dimension d
    - y_i are labels, and y_i is either 1 or -1
  - query is x
  - predicted label is:

$$\mathrm{sign}(\mathbf{a}^T\mathbf{x} + b)$$

Bias, variance, irreducible error

- Assume we choose a classifier from some family
  - eg, a particular linear classifier from the family of linear classifiers
- The error on future data decomposes into three terms:

Bias + Variance + Irreducible error

The error caused by the fact that even the best classifier in the family makes errors

The error caused by the fact that we didn't get the best classifier in the family, just nearly

Unavoidable error (recall alien example)

Question: how to choose a and b?

- Rough answer:
  - split dataset into train/test
  - choose a and b to get good behavior on train
    - how? details?
  - evaluate on test

Question: how to choose a and b?

- Choose a and b to get good behavior on train
    - write a cost function C(a, b) and find best
- The cost function:
    - generally, each training example should contribute evenly
        - average of per example costs

$$C(\mathbf{a}^T, b) = \left(\frac{1}{N}\right) \sum_{i=1}^{N} c(\mathbf{a}, b, \mathbf{x}_i)$$

Desirable properties c(a, b, x_i)

- Write: $\gamma_i = \mathbf{a}^T \mathbf{x}_i + b$
  - sign is the predicted label
  - magnitude is prop to distance to line
- We want:
  - Prediction <> true label:
    - c(a, b, x_i) should be big. Should get bigger when magnitude is bigger, but not too fast
  - Prediction == label, mag small:
    - c(a, b, x_i) should be non-zero, getting smaller as magnitude gets bigger
  - Prediction==label, mag big:
    - c(a, b, x_i) should be zero

The hinge loss

$$c(\mathbf{a}, b, \mathbf{x}_i) = \max(1 - \left(\mathbf{a}^T \mathbf{x}_i + b\right) y_i, 0) = \max(1 - \gamma_i y_i, 0)$$

Feature vector for example

True label for example

$$\gamma_i = \mathbf{a}^T \mathbf{x}_i + b$$

The hinge loss

$$c(\mathbf{a}, b, \mathbf{x}_i) = \max(1 - \left(\mathbf{a}^T \mathbf{x}_i + b\right) y_i, 0) = \max(1 - \gamma_i y_i, 0)$$

- Notice:
  - Prediction <> true label:
    - c(a, b, x_i) is big. Gets bigger when magnitude grows, but not too fast
  - Prediction == label, mag small:
    - c(a, b, x_i) is non-zero, getting smaller as magnitude gets bigger
  - Prediction==label, mag big:
    - c(a, b, x_i) is zero

Regularization

- The cost function drives us towards a classifier that does well on training data
  - but what about future data?
- Cases:
  - future data item is correctly classified, large mag
    - nothing to do
  - future data item is correctly classified, small mag
    - OR is incorrectly classified
    - Notice:
      - hinge loss on this data item (which we don't know) can be scaled
      - and this doesn't change the classification of the training data

$$\text{if } (\mathbf{a}^T\mathbf{x} + b)y < 1 \text{ and } s > 0, \text{ we have } c(s\mathbf{a}, sb, \mathbf{x}_i) = (1 - s\left(\mathbf{a}^T\mathbf{x}_i + b\right) y_i$$

Regularization

- IDEA:
  - Hinge loss on future data items can be reduced by scaling a, b
  - Equiv:
    - find “small” a, b that produce low cost on training
  - Equiv:
    - penalize large a during training
- Which gets us

Regularization weight

$$S(\mathbf{a}, b; \lambda) = \underbrace{\left[(1/N)\sum_{i=1}^{N}\max(0, 1 - y_i\left(\mathbf{a}^T\mathbf{x}_i + b\right))\right]}_{\text{Data term; empirical risk}} + \overset{\downarrow}{\lambda}\underbrace{\left(\frac{\mathbf{a}^T\mathbf{a}}{2}\right)}_{\text{Regularization term}}.$$

Data term; empirical risk

Regularization term

Choosing a classifier

- Minimize this expression with respect a, b

$$S(\mathbf{a}, b; \lambda) = \underbrace{\left[(1/N) \sum_{i=1}^{N} \max(0, 1 - y_i \left(\mathbf{a}^T \mathbf{x}_i + b \right)) \right]}_{\text{Data term; empirical risk}} + \lambda \underbrace{\left(\frac{\mathbf{a}^T \mathbf{a}}{2} \right)}_{\text{Regularization term}}.$$

Choosing a classifier, II

- Write $\mathbf{u} = \begin{bmatrix} \mathbf{a} \\ b \end{bmatrix}$ $g(\mathbf{u})$ for cost function

- Usual procedure:
  - start at $\mathbf{u}^{(0)}$
  - repeat:
    - find a descent direction $\mathbf{p}^{(i)}$
    - choose a steplength η
    - update $\mathbf{u}^{(i+1)} = \mathbf{u}^{(i)} + \eta \mathbf{p}^{(i)}$

Choosing a classifier, III

- What is descent direction?
  - gradient descent:

$$-\nabla_{\mathbf{u}} g = -\left[\begin{array}{c} \frac{\partial g}{\partial u_1} \\ \cdots \end{array}\right]$$

  - Newton's method:
    - second derivatives help choose

- What is steplength?
  - small number
  - search for a good one

$$S(\mathbf{a}, b; \lambda) = \left[(1/N) \sum_{i=1}^{N} \max(0, 1 - y_i \left(\mathbf{a}^T \mathbf{x}_i + b \right)) \right] + \lambda \left(\frac{\mathbf{a}^T \mathbf{a}}{2} \right).$$

Stochastic gradient descent

- Assume we choose k data items uniformly at random
  - compute

$$C_k = \frac{1}{k}\sum_{i=1}^{k} c(\mathbf{a}, b, \mathbf{x}_i)$$

- We have

$$\mathbb{E}\left[C_k\right] = \frac{1}{N}\sum_{i=1}^{N} c(\mathbf{a}, b, \mathbf{x}_i)$$

What is the gradient?

Stochastic gradient descent, II

- IDEA:
    - repeat:
        - pick one or few data items uniformly at random
        - compute gradient using those
            - it isn't right, but
                - expected value is right
                - it's quick
        - take step backward down this gradient

$\uparrow$ How far? it's too hard to search because it's too hard to evaluate g(u) so fixed, but may change over time (eg. 1000 steps at 1e-3; 1000 steps at 1e-4; etc).

Convergence: too hard to test; instead,
do this a fixed large number of steps, and monitor error rate

Stochastic gradient descent, III

- IDEA:
    - repeat:
        - pick one or few data items uniformly at random
        - compute gradient using those
            - it isn't right, but
                - expected value is right
                - it's quick
        - take step backward down this gradient

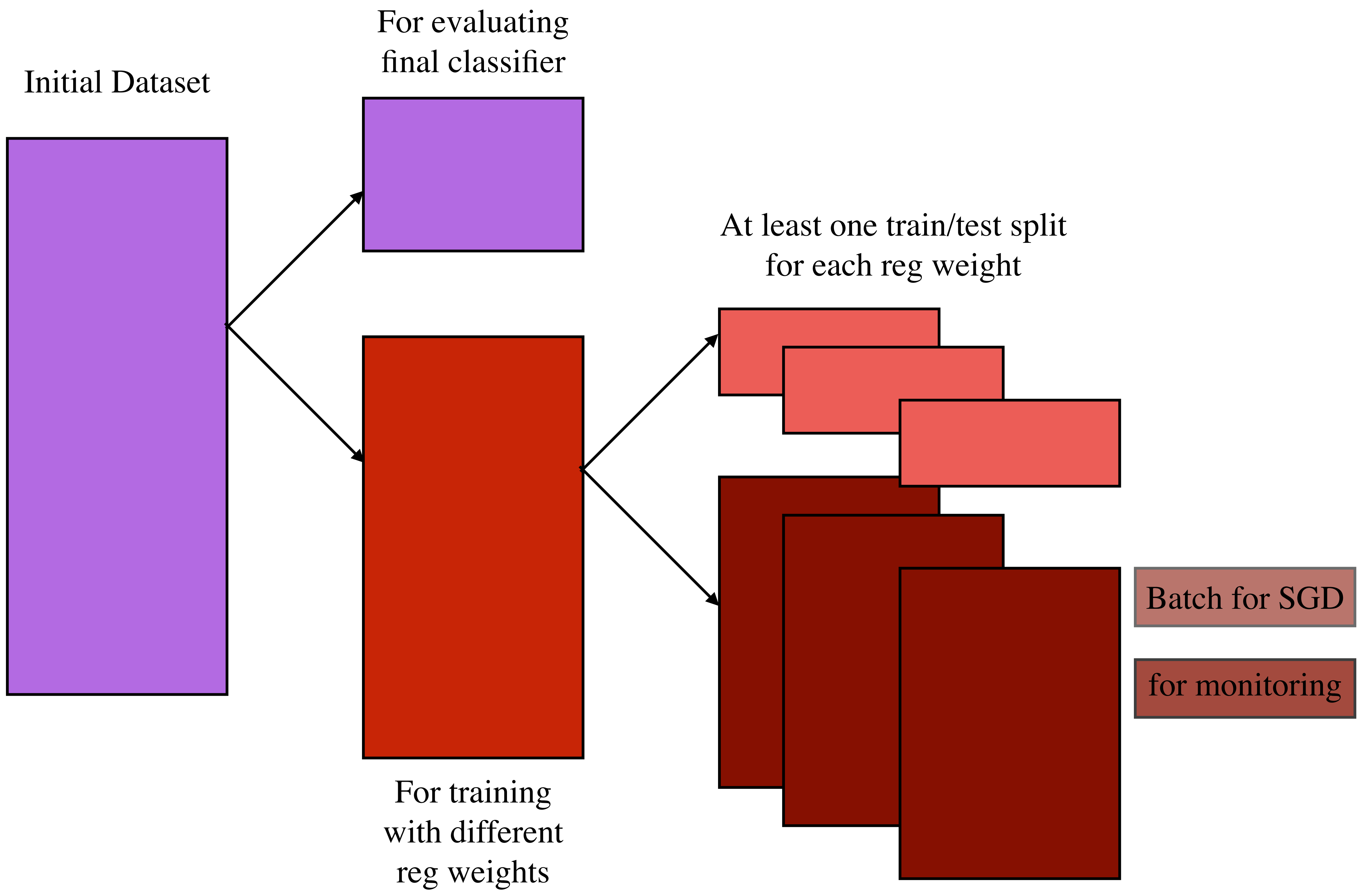
Convergence: too hard to test;
instead, do this a fixed large
number of steps, and monitor error rate

General points

- Support vector machine (SVM):
  - A linear classifier trained with hinge loss
- SVM’s:
  - Impressively reliable if you have good features
  - Easy and effective
  - Good evidence you don’t need to see all the training data
    - ie SGD might get you to about the right classifier without seeing all training data

Getting the regularization weight

- Strategy:
  - try various values, choose the one that yields best classifier
  - not super sensitive - search by factors of 10
- Remember:
  - we can’t evaluate a classifier on data used to train it!
  - this makes training and evaluating an SVM slightly elaborate
- Constraints:
  - we need to know how well the final classifier works - split off some data
  - for each value of reg. weight, we must evaluate - split off some data
  - we may need to monitor - split off some data


Initial Dataset
For evaluating final classifier
For training with different reg weights
At least one train/test split for each reg weight
Batch for SGD
for monitoring

Behavior

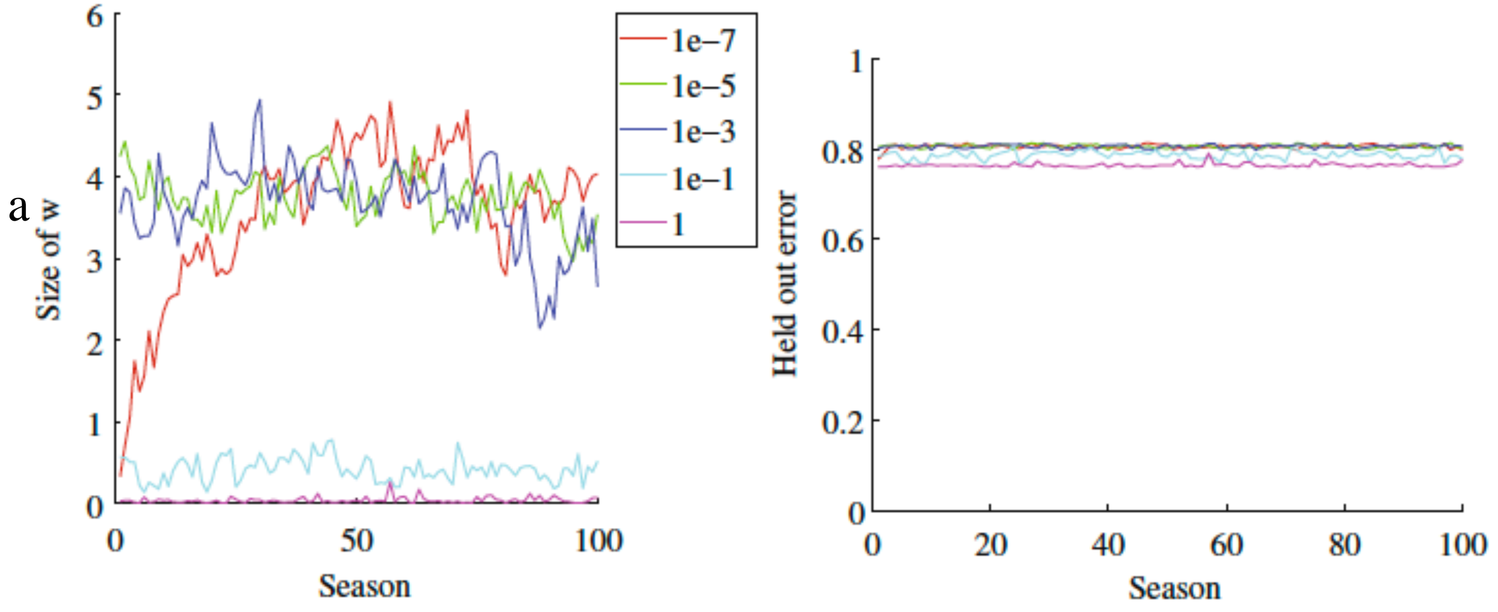


Fig. 11.3 On the *left*, the magnitude of the weight vector a at the end of each season for the first training regime described in the text. On the *right*, the accuracy on held out data at the end of each season. Notice how different choices of regularization parameter lead to different magnitudes of a; how the method isn't particularly sensitive to choice of regularization parameter (they change by factors of 100); how the accuracy settles down fairly quickly; and how overlarge values of the regularization parameter do lead to a loss of accuracy

Stochastic gradient descent

- IDEA:
    - repeat:
        - pick one or few data items uniformly at random
        - compute gradient using those
        - take step backward down this gradient

Convergence: too hard to test;
instead, do this a fixed large
number of steps, and monitor error rate

Gradient

$$\nabla_{\mathbf{a}} S' = \begin{cases} \lambda \mathbf{a} & \text{if } \gamma_i y_i > 1 \\ -y_i \mathbf{x}_i + \lambda \mathbf{a} & \text{otherwise} \end{cases}$$

$$\nabla_b S' = \begin{cases} 0 & \text{if } \gamma_i y_i > 1 \\ -y_i & \text{otherwise} \end{cases}$$

- Notice:
  - this “makes sense”

A little line geometry…

- All this applies to planes, hyperplanes, etc.
    - easier to draw for lines

$\mathbf{x}$	a vector representing point on x, y plane
$\mathbf{a}^T\mathbf{x} + b = 0$	equation satisfied by all points on the line
$\mathbf{a}$	vector normal to the line
$\dfrac{\lvert \mathbf{a}^T\mathbf{x} + b \rvert}{\sqrt{(\mathbf{a}^T\mathbf{a})}}$	distance from point x to the line

$\gamma_i y_i > 1$

In this case, push line away from example

$$\nabla_{\mathbf{a}} S = \lambda \mathbf{a}$$

$$\nabla_b S = 0$$

$$\gamma_i y_i \leq 1$$

In this case, turn line and push towards example

$$\nabla_{\mathbf{a}} S = -y_i \mathbf{x}_i + \lambda \mathbf{a}$$

$$\nabla_b S = -y_i$$

Some practical examples

- Whitening features is a very good idea
- important to translate as well - why?
- Notice one interesting point
  - the gradient is not zero, even if you get every training example right
  - why this is a “good thing”
- Notice shrinking effect of lamda

Why this is a support vector machine

- Simple geometric argument suggests
  - very few examples are important in linearly separable case
    - where zero error is possible for a linear classifier
- This extends to other cases
  - the hyperplane is determined by a very small set of examples
    - these are sometimes known as support vectors

Notice we have a recipe…

- We could apply to other cost functions
- We could apply SGD to other predictors
  - neural networks, etc.
- Procedure for selecting reg. weight is general
  - we'll see other cases

Logistic regression

$$c(\mathbf{a}, b, \mathbf{x}_i) = \log\left(1 + e^{-\left[(\mathbf{a}^T\mathbf{x}_i + b)y_i\right]}\right)$$

- This is very like the hinge loss when plotted
  - but smooth

$$S(\mathbf{a}, b; \lambda) = \frac{1}{N}\sum_i \log\left(1 + e^{-\left[(\mathbf{a}^T\mathbf{x}_i + b)y_i\right]}\right) + (\frac{\lambda}{2})\mathbf{a}^T\mathbf{a}$$

Logistic regression - gradient

- Pick a batch of one example

$$S(\mathbf{a}, b; \lambda) = \log\left(1 + e^{-\left[(\mathbf{a}^T\mathbf{x}_i + b)y_i\right]}\right) + (\frac{\lambda}{2})\mathbf{a}^T\mathbf{a}$$

$$\gamma_i = \mathbf{a}^T\mathbf{x}_i + b$$

$$\nabla_{\mathbf{a}} S = \frac{-e^{-\gamma_i y_i}}{1 + e^{-\gamma_i y_i}} y_i \mathbf{x_i} + \lambda\mathbf{a}$$

$$\nabla_b S = \frac{-e^{-\gamma_i y_i}}{1 + e^{-\gamma_i y_i}} y_i$$

Logistic regression - gradient

- Notice:
  - this “makes sense”
  - is very like linear SVM gradient

$$\nabla_{\mathbf{a}} S = \frac{-e^{-\gamma_i y_i}}{1+e^{-\gamma_i y_i}} y_i \mathbf{x_i} + \lambda \mathbf{a}$$

$$\nabla_{\mathbf{a}} S' = \begin{cases} \lambda \mathbf{a} & \text{if } \gamma_i y_i > 1 \\ -y_i \mathbf{x}_i + \lambda \mathbf{a} & \text{otherwise} \end{cases}$$

$$\nabla_b S = \frac{-e^{-\gamma_i y_i}}{1+e^{-\gamma_i y_i}} y_i$$

$$\nabla_b S' = \begin{cases} 0 & \text{if } \gamma_i y_i > 1 \\ -y_i & \text{otherwise} \end{cases}$$

Some practical examples

- Whitening features is a very good idea
- important to translate as well - why?
- Notice one interesting point
  - the gradient is not zero, even if you get every training example right
  - why this is a “good thing”
- Notice shrinking effect of lamda

Multi class classification with an SVM

- SVM is naturally a binary classifier
- Bad option:
  - expand class labels as a binary vector
  - use one SVM to predict each bit
  - this doesn’t work - if you get bit wrong, you’re in trouble
- Better options:
  - One v one
  - One v all
  - neither is perfect