Linear Classifiers
Reading

- http://cs231n.github.io/classification/
Announcements

• P4 out today. For real.
Goals

• Understand the benefits and limitations of linear classifiers over KNN.
• Understand the tradeoff between complexity in the feature extractor vs. complexity in the classifier.
• Understand the mathematical formulation of a binary and multiclass linear classifier.
• Know the definition and purpose of a loss function
• Understand the intuition behind linear classifiers with:
  – SVM loss
  – Softmax loss
Image classification - Multiclass classification

Which of these is it: dog, cat or zebra?

Dog
KNN: Bottom Line

• Fast to train but slow to predict
• Distance metrics don’t behave well for high-dimensional image vectors
Classifying Images

- Nearest Neighbor Classifier
  - the data
  - NN classifier $h$

- Linear Classifier
Linear classifiers

• Finding nearest neighbor is slow.

• Basic idea:
  – Training time: find a line that separates the data
  – Testing time: which side of the line is $\phi(x)$ on?

+ Fast to compute
- Restrictive – data must be linearly separable
Linear classifiers

• A linear classifier corresponds to a hyperplane
  – Equivalent of a line in high-dimensional space
  – Equation: \( w^T x + b = 0 \)

• Points on the same side are the same class
Does this ever work?

• It’s easier to be linearly separable in high-dimensional space.

• But simple linear classifiers still don’t work on most interesting data.
Some history from the Antedeepluvian Era

• Example pipeline from days of yore:
  – Detect corners and extract SIFT features
  – Collect features into a “bag of features”
  – (if you’re feeling fancy) maintain some spatial information
  – Somehow convert feature bag to fixed size
  – Apply linear classifier

• Key idea: $\phi$ is designed by hand, while $h$ is learned from data.
Some history of the Antedeleepluvian Era

• Key idea: $\phi$ is designed by hand, while $h$ is learned from data.

• Nowadays: learn both from data - “end-to-end”: image goes in, label comes out.
  – Enabled only recently by bigger
    • labeled datasets
    • compute power (GPUs)
Linear classifiers

- Equation: $w^T x + b = 0$
- Points on the same side are the same class
We have a classifier

• $h(x) = w^T x + b$ gives a score

• Score negative: red
• Score positive: blue

• Does it solve the runtime issues of KNN?
We have a classifier

- $h(x) = w^T x + b$ gives a score
- Score negative: red
- Score positive: blue
- Where do $W$ and $b$ come from?
How do we find a good $W, b$?

- Step 1: For a given $W, b$, decide on a **Loss Function**: a measure of how much we dislike the line.
- Step 2: use **optimization** to find the $W, b$ that *minimize* the loss function.
Questions?
Multiclass Linear Classifiers: Stack multiple $w^\top$ into a matrix.
Multiclass Linear Classifier: Geometric Interpretation
Interpreting a Linear Classifier: Visual Viewpoint

- airplane
- automobile
- bird
- cat
- deer
- dog
- frog
- horse
- ship
- truck

Input image:

Score:

W:

b:

0.2 -0.5
0.1 2.0
1.5 1.3
2.1 0.0
0.2 -0.3

1.1
3.2
-1.2

-96.8
437.9
61.95

Slide: Fei-Fei Li, Justin Johnson, & Serena Yeung
Loss Functions

• Step 1: For a given $W$, $b$, decide on a **Loss Function**: a measure of how much we dislike this classifier.

• Step 2: use **optimization** to find the $W$, $b$ that **minimize** the loss function.
Loss Functions

• Step 1: For a given $W$, $b$, decide on a **Loss Function**: a measure of how much we dislike this classifier.

• Loss Function intuition:
  – loss should be large if many data points are misclassified
  – loss should be small (0?) if all data is classified correctly.
Loss Functions – SVM Loss

• SVM Loss:
  – Insists that data points are not just correctly classified, but a certain distance from the hyperplane:
  – \( L_i = \max(0 \cdot x_i, 1 - y_i(w^T x_i + b)) \)

\( x_i = \) i’th data point
\( y_i = \) i’th data point’s true label:
  -1 if red
  +1 if green
Loss Functions – SVM Loss

• SVM Loss:
  – Insists that data points are not just correctly classified, but a certain distance from the hyperplane:
  
  \[ L_i = \max(0, 1 - y_i(w^Tx_i + b)) \]

  \[ x_i = \text{i’th data point} \]
  \[ y_i = \text{i’th data point’s true label:} \]
  -1 if red
  +1 if green

  \[ L(w, b) = \sum_i L_i \]

  – Loss for a given line is the sum of the loss for all datapoints
The Bias Trick
The Bias Trick

• Fold b into an additional dimension of w
• Add a fixed 1 to all feature vectors.

• Now, $h(x) = w^T x$
Softmax Classifier / Cross-Entropy Loss: Intuition

$W^T x + b$ gives us a vector of scores, one per class (each row of $W$ is a classifier)

Wouldn’t it be nice to interpret these as probabilities?
Binary Equivalent:
Logistic Regression Loss

If \( y = 1 \)

If \( y = 0 \)
Softmax Classifier / Cross-Entropy Loss: Intuition

$W^T x + b$ gives us a vector of scores, one per class (each row of $W$ is a classifier)

Wouldn’t it be nice to interpret these as probabilities?

They’re not:

- not always nonnegative
- don’t sum to 1

But we can treat them as unnormalized log probabilities.
Softmax Classifier / Cross-Entropy Loss: Intuition

But we can treat scores as *unnormalized log probabilities*. 
Cross-Entropy Loss: Intuition

$W^T x + b$ gives us a vector of scores, one per class (each row of $W$ is a classifier)

Cross-Entropy loss: Apply a sigmoid to get values between 0 and 1, then normalize them to sum to 1.

Then, compute the difference between the predicted distribution and the true distribution.