Lecture 8: Theoretical basics of machine learning

Machine Learning and Imaging

BME 590L
Roarke Horstmeyer
Last time: the linear classification model – what’s not to like?

1. Can only separate data with lines (hyper-planes)…
2. We only allowed for binary labels (y = +/- 1)
3. Error function $L_{in}$ inherently makes assumptions about statistical distribution of data
Cost functions matter: a simple example

What if you’re a CIA agent?

\[ L_{\text{in}} = 100,000 \text{ReLU}[f(x, W) - y] + \text{ReLU}[y - f(x, W)] \]

BIG penalty for intruder

Don’t mind about annoyance...

Letting an intruder in

Y

<table>
<thead>
<tr>
<th></th>
<th>+1</th>
<th>-1</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>No Error</td>
<td>False reject</td>
</tr>
<tr>
<td>-1</td>
<td>False accept</td>
<td>No Error</td>
</tr>
</tbody>
</table>

It’s you, but you can’t get in...

https://www.cnet.com/how-to/apple-face-id-everything-you-need-to-know/
Last time: the linear classification model – what’s not to like?

1. Can only separate data with lines (hyper-planes)...
2. We only allowed for binary labels ($y = +/- 1$)
3. Error function $L_{in}$ inherently makes assumptions about statistical distribution of data
Deriving cost function for logistic classification for probabilistic outputs

Maximize \( P(y_1, y_2, \ldots, y_N \mid x_1, x_2, \ldots, x_N) = \prod_{n=1}^{N} \theta(y_n w^T x_n) \)

Minimize \( -\frac{1}{N} \ln \left( \prod_{n=1}^{N} \theta(y_n w^T x) \right) \)

Minimize \( \frac{1}{N} \sum_{n=1}^{N} \ln \left( \frac{1}{\theta(y_n w^T x)} \right) \)

Use relationship \( \theta(a) = \frac{1}{1 + e^{-a}} \)

Minimize \( L_{in}(w) = \frac{1}{N} \sum_{n=1}^{N} \ln(1 + e^{-y_n w^T x}) \)

Cross entropy error for logistic classification
Requires iterative solution to minimize

Mean-square error for linear classification
Closed form solution available

Minimize \( L_{in}(w) = \frac{1}{N} \sum_{n=1}^{N} (y_n - w^T x)^2 \)
The linear classification model – what’s not to like?

\[ L_{\text{in}}(y, f(W,x)) = \text{cross_entropy}(y, f(W,x)) \]

\[ y^* = Wx \]

Probabilistic mapping to y
The linear classification model – what’s not to like?

1. Can only separate data with lines (hyper-planes)...
2. We only allowed for binary labels ($y = +/- 1$)
3. Error function $L_{in}$ inherently makes assumptions about statistical distribution of data
\[ f = W_1 x \]

Learned \( f \): not flexible
Can we add flexibility by multiplying with another weight matrix?

\[
\begin{align*}
  f_1 &= W_1 x + b_1 \\
  f_2 &= W_2 f_1 + b_2 \\
  f_2 &= W_2 (W_1 x + b_1) + b_2 \\
  f_2 &= W' x + b'
\end{align*}
\]

Unfortunately not…
Learned $f$: not flexible

$$f = W_1 x$$

Learned $f$: a bit flexible

$$f = W_2 \max(W_1 x, 0)$$
Deep Imaging

Training data

\[ f = W_1 x \]

Learned \( f \): not flexible

Training data

\[ f = W_2 \max(W_1 x, 0) \]

Learned \( f \): a bit flexible

Training data

\[ f = W_3 \max(0, W_2 \max(W_1 x, 0)) \]

Learned \( f \): more flexible

Does it generalize???

We can keep adding these “layers”…
Getting us to Convolutional Neural Networks

Each matrix $W$ is a convolution matrix

After, apply non-linearity and sub-sampling

Repeat a few times

At the end, use a full $W$ for a final matrix multiplication

Original Image published in [LeCun et al., 1998]
Getting us to Convolutional Neural Networks

In practice, this process is repeated many times:
Aside #1 before convolutional neural network details

Q: Can we try to avoid making these learning models too complicated?

A: Yes, by transforming the data coordinates before classification.
Image Features: Motivation

Cannot separate red and blue points with linear classifier

\[ f(x, y) = (r(x, y), \theta(x, y)) \]

After applying feature transform, points can be separated by linear classifier

From Stanford CS231: [http://cs231n.stanford.edu/](http://cs231n.stanford.edu/)
Example: Color Histogram

From Stanford CS231: http://cs231n.stanford.edu/
Example: Histogram of Oriented Gradients (HoG)

Divide image into 8x8 pixel regions
Within each region quantize edge
direction into 9 bins

Example: 320x240 image gets divided
into 40x30 bins; in each bin there are
9 numbers so feature vector has
30*40*9 = 10,800 numbers

From Stanford CS231: http://cs231n.stanford.edu/
Example: Bag of Words

Step 1: Build codebook
- Extract random patches
- Cluster patches to form "codebook" of "visual words"

Fei-Fei and Perona, "A bayesian hierarchical model for learning natural scene categories", CVPR 2005

Step 2: Encode images

From Stanford CS231: http://cs231n.stanford.edu/
Image features vs ConvNets

From Stanford CS231: [http://cs231n.stanford.edu/](http://cs231n.stanford.edu/)
Hand-crafted versus learned features also applies to imaging

Normal camera

Classification acc.: $\text{NN} \rightarrow 80\%$

Pre-designed mask

Classification acc.: $\text{NN} \rightarrow 90\%$

ML-optimized mask

Classification acc.: $\text{NN} \rightarrow 95\%$
Statistical Machine Learning in 30 minutes

Two competing goals in machine learning:

1. Can we make sure the in-sample error $L_{in}(y, f(x,W))$ is small enough?
   - Appropriate cost function
   - “Complex enough” model
Statistical Machine Learning in 30 minutes

Two competing goals in machine learning:

1. Can we make sure the in-sample error $L_{in}(y, f(x,W))$ is small enough?
   - Appropriate cost function
   - “complex enough” model

2. Can we make sure that $L_{out}(y, f(x,W))$ is close enough to $L_{in}(y, f(x,W))$?
   - Probabilistic analysis says yes!
   - $|L_{in} - L_{out}|$ bounded from above
   - Bound grows with model capacity (bad)
   - Bound shrinks with # of training examples (good)
Model overfitting versus underfitting – a thought exercise

Let’s fit these “training” data points:

And then here’s our testing dataset – good?

Perfect! But lucky….

10^{th} order Polynomial Fit
Model overfitting versus underfitting – a thought exercise

Let’s fit these “training” data points:

What if our test dataset was this:

Noisy, low complexity target

10th order Polynomial Fit
Let’s fit these “training” data points:

What if our test dataset was this:

If our data was noisy and the target followed a low-complexity model, we’d be better off with a second order fit!
Model overfitting versus underfitting – a thought exercise

If our data was noisy and the target followed a low-complexity model, we’d be better off with a second order fit!

**Model capacity:** ability to fit a wide range of functions

Control capacity through model’s hypothesis space (set of functions model can take)

Hard to know ahead of time!

*Deep Learning, I. Goodfellow et al., Fig. 5.3*
Low Capacity Model

Expected Error

- Test error $L_{\text{out}}$
- Training error $L_{\text{in}}$

Relatively high error

# of data points, J
Deep Imaging

Low Capacity Model

- Relatively high error
- Test error $L_{\text{out}}$
- Training error $L_{\text{in}}$

# of data points, $J$

High Capacity Model

- Relatively low error
- Test error $L_{\text{out}}$
- Training error $L_{\text{in}}$

But harder to get test error down...

# of data points, $J$
Take away concepts:

• Can’t ever really expect test error to be less than training error

• Complicated models tend to appear to “do better” during training, before trying test data

• When the model gets complicated and you don’t have enough data, challenging to get test error down
Model bias versus variance

“True” model

Bias = distance between true and learned model

Hypothesis space

True model

Learned $f$

Learned $f$

$x$

$y$
Model bias versus variance

Models that tend to be “a bit too simple” are biased away from “true” model.

Bias = distance between true and learned model

Bias = (g(x) - f(x))^2

Measures how far our learning model f is biased away from target function g (for perfect training data classification).
Model bias versus variance

Bias = distance between true and learned model

Variance = \text{Var}[g(x)]

More complicated datasets exhibit lots of variance between training and test set.
Model bias versus variance

Bias = distance between true and learned model

“True” model $g$

Hypothesis space

Learned $f$

Hypothesis space

Test Error is sum of model bias and variance!

Goal is to find a model $f$ that balances between these two quantities for a given dataset

Error vs. # data points

Expected Error

Variance

Test error $L_{\text{out}}$

Bias

Training error $L_{\text{in}}$

# of data points, $J$
How to formally define capacity and complexity?

• Short answer: it’s complicated…

• Related to something called the VC Dimension
  • Can provide theoretical bounds on performance
  • Dimensional bounds rather than scalar bounds…

• I decided not to go into it, but please let me know if you’d like me to!
Conclusions from statistical machine learning

- Conclusion: you want a model that is complex enough to capture variations within high-dimensional space, but not too complex such that it overfits the data

- Want a model with a high capacity, but can still generalize to data outside training set
  - More data -> less overfitting, complex target -> more overfitting

- For simple models, we can measure complexity via degrees of freedom, the VC bound and so-on to help us nail down ideal models that can generalize well
Conclusions from statistical machine learning

- Conclusion: you want a model that is complex enough to capture variations within high-dimensional space, but not too complex such that it overfits the data

- Want a model with a high capacity, but can still generalize to data outside training set
  • More data -> less overfitting, complex target -> more overfitting

- For simple models, we can measure complexity via degrees of freedom, the VC bound and so-on to help us nail down ideal models that can generalize well

- **For DL models:** this will get too hard...here's a few counter-intuitive properties:

  1. A fixed DL architecture exhibits data-dependent complexities
     • e.g., “good” DL networks achieve 0 training error on images with random labels, so cannot generalize at all in this case, and are too complex

  2. DL networks with more hidden units leads to *better* generalization (the main finding of the last few years). So deeper models tend to be less complex, actually...

  3. Complexity depends upon loss function and optimization method...
Important to remember: “No Free Lunch Theorem”

- “Averaged over all possible data-generating distributions, every classification algorithm has the same error rate when classifying previously unobserved points.”

- The most sophisticated DL algorithm has save average performance (averaged over all possible tasks) as the simplest.
Important to remember: “No Free Lunch Theorem”

• “Averaged over all possible data-generating distributions, every classification algorithm has the same error rate when classifying previously unobserved points.”

• The most sophisticated DL algorithm has save average performance (averaged over all possible tasks) as the simplest.

• Must make assumptions about probability distributions of inputs we’ll encounter in real-world

Set of 20 “images”, random Gaussian distribution

Face at different orientations = manifold n-D space

Deep Learning, I. Goodfellow et al., Fig. 5.12-13
Important to remember: “No Free Lunch Theorem”

- “Averaged over all possible data-generating distributions, every classification algorithm has the same error rate when classifying previously unobserved points.”

- The most sophisticated DL algorithm has save average performance (averaged over all possible tasks) as the simplest.

- Must make assumptions about probability distributions of inputs we’ll encounter in real-world

CT reconstructions of every brain in the world = kD manifold in nD space?

1D Manifold in 2D space