Practical Issues:
Features, Evaluation, Debugging

CMSC 422
MARINE CARPUAT
marine@cs.umd.edu
Typical Design Process for an ML Application

1. Real world goal: revenue
2. R.W. mechanism: better ad display
3. Learning problem: classify click-through
4. Data collection: interact w/ current system
5. Collected data: query, ad, click
6. Data representation: bow², +/− click
7. Hypoth. class/ind. bias: dec. tree depth 20
8. Training data selection: subset from Apr '16
9. Model training (+hps): final decision tree
10. Predict on test data: subset from May '16
11. Evaluate error: AUC for +/− click predict'n

Deploy!
Practical Issues

• “garbage in, garbage out”
  – Learning algorithms can’t compensate for useless training examples
    • E.g., if all features are irrelevant
  – Feature design can have bigger impact on performance than tweaking the learning algorithm
Improving Input Representations

• Feature pruning
• Feature normalization

Centering: $x_{n,d} \leftarrow x_{n,d} - \mu_d$  \hfill (5.1)

Variance Scaling: $x_{n,d} \leftarrow x_{n,d} / \sigma_d$  \hfill (5.2)

Absolute Scaling: $x_{n,d} \leftarrow x_{n,d} / r_d$  \hfill (5.3)

where:

\[ \mu_d = \frac{1}{N} \sum_n x_{n,d} \]  \hfill (5.4)

\[ \sigma_d = \sqrt{\frac{1}{N-1} \sum_n (x_{n,d} - \mu_d)^2} \]  \hfill (5.5)

\[ r_d = \max_n |x_{n,d}| \]  \hfill (5.6)

• Example normalization

\[ x_n \leftarrow x_n / \|x_n\| \]

See CIML 5.3
Practical Issues: Evaluation

• So far we’ve measured classification performance using **accuracy**

• But this is not a good metric when some errors matter more than others
  – Given medical record, predict whether patient has cancer or not
  – Given a document collection and a query, find documents in collection that are relevant to query
The 2-by-2 contingency table

Imagine we are addressing a document retrieval task for a given query, where +1 means that the document is relevant and -1 means that the document is not relevant.

We can categorize predictions as:
- true/false positives
- true/false negatives

<table>
<thead>
<tr>
<th>Prediction</th>
<th>Gold label = +1</th>
<th>Gold label = -1</th>
</tr>
</thead>
<tbody>
<tr>
<td>+1</td>
<td>tp</td>
<td>fp</td>
</tr>
<tr>
<td>-1</td>
<td>fn</td>
<td>tn</td>
</tr>
</tbody>
</table>
Precision and recall

- **Precision**: % of positive predictions that are correct

- **Recall**: % of positive gold labels that are found

<table>
<thead>
<tr>
<th></th>
<th>Gold label = +1</th>
<th>Gold label = -1</th>
</tr>
</thead>
<tbody>
<tr>
<td>Prediction = +1</td>
<td>tp</td>
<td>fp</td>
</tr>
<tr>
<td>Prediction = -1</td>
<td>fn</td>
<td>tn</td>
</tr>
</tbody>
</table>
Practical Issues: hyperparameter tuning with dev set vs. cross-validation

**Algorithm 8 CrossValidate**(LearningAlgorithm, Data, K)

1. \( \hat{e} \leftarrow \infty \)  
   // store lowest error encountered so far  
2. \( \hat{\alpha} \leftarrow \) unknown  
   // store the hyperparameter setting that yielded it  
3. **for all** hyperparameter settings \( \alpha \) **do**  
4. \( \text{err} \leftarrow [ ] \)  
   // keep track of the \( K \)-many error estimates  
5. **for** \( k = 1 \) **to** \( K \) **do**  
6. \( \text{train} \leftarrow \{(x_n, y_n) \in \text{Data} : n \mod K \neq k - 1\} \)  
7. \( \text{test} \leftarrow \{(x_n, y_n) \in \text{Data} : n \mod K = k - 1\} \)  
   // test every \( K \)-th example  
8. \( \text{model} \leftarrow \text{Run LearningAlgorithm on train} \)  
9. \( \text{err} \leftarrow \text{err} \oplus \text{error of model on test} \)  
   // add current error to list of errors  
10. **end for**  
11. \( \text{avgErr} \leftarrow \text{mean of set err} \)  
12. **if** \( \text{avgErr} < \hat{e} \) **then**  
13. \( \hat{e} \leftarrow \text{avgErr} \)  
   // remember these settings  
14. \( \hat{\alpha} \leftarrow \alpha \)  
   // because they're the best so far  
15. **end if**  
16. **end for**
Practical Issues: Debugging!

• You’ve implemented a learning algorithm,  
• You try it on some train/dev/test data  
• You get really bad performance  

• What’s going on?  
  – Is the data too noisy?  
  – Is the learning problem too hard?  
  – Is the implementation of the learning algorithm buggy?
Strategies for Isolating Causes of Errors

• Is the problem with *generalization* to test data?
  – Can learner fit the training data?
  – Yes: problem is in generalization to test data
  – No: problem is in representation (need better features or better data)

• **Train/test mismatch**?
  – Try reselecting train/test by shuffling training data and test together
Strategies for Isolating Causes of Errors

• Is algorithm **implementation correct**?
  – Measure loss rather than accuracy
  – Hand-craft a toy dataset

• **Is representation adequate?**
  – Can you learn if you add a cheating feature that perfectly correlates with correct class?

• Do you have **enough data**?
  – Try training on 80% of the training set, how much does it hurt performance?
Formalizing Errors

The learned classifier

$\mathcal{F}$ set of all possible classifiers using a fixed representation

$$\text{error}(f) = \left[ \text{error}(f) - \min_{f^* \in \mathcal{F}} \text{error}(f^*) \right] + \left[ \min_{f^* \in \mathcal{F}} \text{error}(f) \right]$$

- **estimation error**
- **approximation error**

How far is the learned classifier $f$ from the optimal classifier $f^*$?

Quality of the model family aka hypothesis class
The bias/variance trade-off

• Trade-off between
  – approximation error (bias)
  – estimation error (variance)

• Example:
  – Consider the always positive classifier
    • Low variance as a function of a random draw of the training set
    • Strongly biased toward predicting +1 no matter what the input
Recap: practical issues

• Learning algorithm is only one of many steps in designing a ML application

• Many things can go wrong, but there are practical strategies for
  – Improving inputs
  – Evaluating
  – Tuning
  – Debugging

• Fundamental ML concepts: estimation vs. approximation error