Practical Advice for Building Machine Learning Applications

Machine Learning
Spring 2021

The slides are partly from Vivek Srikumar
This lecture: ML and the world

• Diagnostics of your learning algorithm

• Error analysis

• Injecting machine learning into *Your Favorite Task*
ML and the world

- Diagnostics of your learning algorithm
- Error analysis
- Injecting machine learning into *Your Favorite Task*
Debugging machine learning

Suppose you train an SVM or a logistic regression classifier for spam detection

You *obviously* follow best practices for finding hyper-parameters (such as cross-validation)

Your classifier is only 75% accurate

**What can you do to improve it?**

(assuming that there are no bugs in the code)
Different ways to improve your model

More training data

Features
1. Use more features
2. Use fewer features
3. Use other features

Better training
1. Run for more iterations
2. Use a different algorithm
3. Use a different classifier
4. Play with regularization
Different ways to improve your model

More training data

Features
1. Use more features
2. Use fewer features
3. Use other features

Better training
1. Run for more iterations
2. Use a different algorithm
3. Use a different classifier
4. Play with regularization

Tedious!
And prone to errors, dependence on luck
Let us try to make this process more methodical
First, diagnostics

Some possible problems:

1. Over-fitting (high variance)
2. Under-fitting (high bias)
3. Your learning does not converge
4. Are you measuring the right thing?

Easier to fix a problem if you know where it is
Detecting over or under fitting

**Over-fitting**: The training accuracy is much higher than the test accuracy
  – The model explains the training set very well, but poor generalization

**Under-fitting**: Both accuracies are unacceptably low
  – The model cannot represent the concept well enough
Detecting high *variance* using learning curves

![Graph showing training error vs size of training data](Image)

- **Error** axis
- **Size of training data** axis
- **Training error** line
Detecting high variance using learning curves

![Graph showing training error and generalization error versus size of training data.]

- **Error**
  - Generalization error/test error
  - Training error

Size of training data
Detecting high variance using learning curves

Test error keeps decreasing as training set increases; more data will help.

Large gap between train and test error:
Typically seen for more complex models.

Error vs. Size of training data:
- Training error (blue line, decreasing)
- Generalization error/test error (red line, increasing)

Error decreases with more data, but there's a gap.

Typically seen for more complex models.
Detecting high bias using learning curves

Both train and test error are unacceptable
(But the model seems to converge)

Typically seen for more simple models

Error

Generalization error/ test error

Training error

Size of training set
Different ways to improve your model

More training data

Features
1. Use more features
2. Use fewer features
3. Use other features

Better training
1. Run for more iterations
2. Use a different algorithm
3. Use a different classifier
4. Play with regularization
Different ways to improve your model

More training data

Helps with over-fitting

Features

1. Use more features
   Helps with under-fitting
2. Use fewer features
   Helps with over-fitting
3. Use other features
   Could help with over-fitting and under-fitting

Better training

1. Run for more iterations
2. Use a different algorithm
3. Use a different classifier
4. Play with regularization
   Could help with over-fitting
Diagnostics

Some possible problems:

✓ Over-fitting (high variance)

✓ Under-fitting (high bias)

3. Your learning does not converge

4. Are you measuring the right thing?
Does your learning algorithm converge?

If learning is framed as an optimization problem, track the objective.
Does your learning algorithm converge?

If learning is framed as an optimization problem, track the objective.

Not always easy to decide

Not yet converged here

How about here?
Does your learning algorithm converge?

If learning is framed as an optimization problem, track the objective.
Does your learning algorithm converge?

If learning is framed as an optimization problem, track the objective.

- Helps to debug
- If we are doing gradient descent on a *convex* function, the objective can’t increase.
- *Caveat:* For SGD, the objective will slightly increase occasionally, but not by much.

Something is wrong
Different ways to improve your model

More training data

- Helps with overfitting

Features

1. Use more features
   - Helps with under-fitting
2. Use fewer features
   - Helps with over-fitting
3. Use other features
   - Could help with over-fitting and under-fitting

Better training

1. Run for more iterations
2. Use a different algorithm
3. Use a different classifier
4. Play with regularization
   - Could help with over-fitting and under-fitting
Different ways to improve your model

More training data

- Helps with overfitting

Features

1. Use more features  - Helps with under-fitting
2. Use fewer features - Helps with over-fitting
3. Use other features - Could help with over-fitting and under-fitting

Better training

1. Run for more iterations
2. Use a different algorithm - Track the objective for convergence
3. Use a different classifier
4. Play with regularization - Could help with over-fitting and under-fitting
Diagnostics

Some possible problems:

✓ Over-fitting (high variance)

✓ Under-fitting (high bias)

✓ Your learning does not converge

4. Are you measuring the right thing?
What to measure

- Accuracy of prediction is the most common measurement

- But if your data set is unbalanced, accuracy may be misleading
  - 1000 positive examples, 1 negative example
  - A classifier that always predicts positive will get 99.9% accuracy. Has it really learned anything?

- Unbalanced labels → measure label specific precision, recall and F-measure
  - Precision for a label: Among examples that are predicted with label, what fraction are correct
  - Recall for a label: Among the examples with given ground truth label, what fraction are correct
  - F-measure: Harmonic mean of precision and recall
ML and the world

- Diagnostics of your learning algorithm
- Error analysis
- Injecting machine learning into *Your Favorite Task*
Machine Learning in this class

ML code
Machine Learning in context

Figure from [Sculley, et al NIPS 2015]
Error Analysis

Generally machine learning plays a small role in a larger application

• Pre-processing
• Feature extraction (possibly by other ML based methods)
• Data transformations

How much do each of these contribute to the error?

Error analysis tries to explain why a system is not performing perfectly
Example: A typical text processing pipeline
Example: A typical text processing pipeline
Example: A typical text processing pipeline
Example: A typical text processing pipeline
Example: A typical text processing pipeline

Text → Words → Parts-of-speech → Parse trees
Example: A typical text processing pipeline

Text → Words → Parts-of-speech → Parse trees → A ML-based application
Example: A typical text processing pipeline

Each of these could be ML driven

Or deterministic

But still error prone

Text

Words

Parts-of-speech

Parse trees

A ML-based application
Example: A typical text processing pipeline

Each of these could be ML driven

Or deterministic

But still error prone

How much do each of these contribute to the error of the final application?
Tracking errors in a complex system

Plug in the ground truth for the intermediate components and see how much the accuracy of the final system changes

<table>
<thead>
<tr>
<th>System</th>
<th>Accuracy</th>
</tr>
</thead>
<tbody>
<tr>
<td>End-to-end predicted</td>
<td>55%</td>
</tr>
<tr>
<td>With ground truth words</td>
<td>60%</td>
</tr>
<tr>
<td>+ ground truth parts-of-speech</td>
<td>84%</td>
</tr>
<tr>
<td>+ ground truth parse trees</td>
<td>89%</td>
</tr>
<tr>
<td>+ ground truth final output</td>
<td>100%</td>
</tr>
</tbody>
</table>
Tracking errors in a complex system

Plug in the ground truth for the intermediate components and see how much the accuracy of the final system changes

<table>
<thead>
<tr>
<th>System</th>
<th>Accuracy</th>
</tr>
</thead>
<tbody>
<tr>
<td>End-to-end predicted</td>
<td>55%</td>
</tr>
<tr>
<td>With ground truth words</td>
<td>60%</td>
</tr>
<tr>
<td>+ ground truth parts-of-speech</td>
<td>84%</td>
</tr>
<tr>
<td>+ ground truth parse trees</td>
<td>89%</td>
</tr>
<tr>
<td>+ ground truth final output</td>
<td>100%</td>
</tr>
</tbody>
</table>

Error in the part-of-speech component hurts the most
Ablative study

Usually seen with features

Suppose we have a collection of features and our system does well, but we don’t know which features are giving us the performance

Evaluate simpler systems that progressively use fewer and fewer features to see which features give the highest boost

It is not enough to have a classifier that works; it is useful to know why it works.

Helps interpret predictions, diagnose errors and can provide an audit trail
ML and the world

• Diagnostics of your learning algorithm

• Error analysis

• Injecting machine learning into *Your Favorite Task*
Classifying fish

Say you want to build a classifier that identifies whether a real physical fish is salmon or tuna
How do you go about this?
Classifying fish

Say you want to build a classifier that identifies whether a real physical fish is salmon or tuna. How do you go about this?

**The slow approach**

1. Carefully identify features, get the best data, the software architecture, maybe design a new learning algorithm
2. Implement it and hope it works

**Advantage:** Perhaps a better approach, maybe even a new learning algorithm. Research.
Classifying fish

Say you want to build a classifier that identifies whether a real physical fish is salmon or tuna. How do you go about this?

The slow approach

1. Carefully identify features, get the best data, the software architecture, maybe design a new learning algorithm
2. Implement it and hope it works

**Advantage:** Perhaps a better approach, maybe even a new learning algorithm. Research.

The hacker’s approach

1. First implement something
2. Use diagnostics to iteratively make it better

**Advantage:** Faster release, will have a solution for your problem quicker
What to watch out for

• **Do you have the right evaluation metric?**
  – And does your loss function reflect it?

• **Beware of contamination**: Ensure that your training data is not contaminated with the test set
  – Learning = generalization to new examples
  – Do not see your test set either. You may inadvertently contaminate the model
  – Beware of contaminating your features with the label!
  – (Be suspicious of perfect predictors)
What to watch out for

• **Be aware of bias vs. variance tradeoff** (or over-fitting vs. under-fitting)

• **Be aware that intuitions may not work in high dimensions**
  – No proof by picture
  – Curse of dimensionality

• **A theoretical guarantee may only be theoretical**
  – May make invalid assumptions (eg: if the data is separable)
  – May only be legitimate with infinite data (eg: estimating probabilities)
  – Experiments on real data are equally important
Big data is not enough

But more data is always better
  – Cleaner data is even better

Learning requires knowledge to guide the learner
  – Machine learning is not a magic wand
What knowledge?

• Which model is the right one for this task?
  – Linear models, decision trees, deep neural networks, etc

• Which learning algorithm?
  – Does the data violate any crucial assumptions that were used to define the learning algorithm or the model?
  – Does that matter?

• Feature engineering is crucial

• Implicitly, these are all claims about the nature of the problem
Miscellaneous advice

• Learn simpler models first
  – If nothing, at least they form a baseline that you can improve upon

• Ensembles seem to work better

• Think about whether your problem is learnable at all
  – Learning = generalization
Wish all of you great success in the future!