Model Selection, Evaluation, Diagnosis

INFO-4604, Applied Machine Learning

University of Colorado Boulder

November 8, 2020

Prof. Abe Handler

A little bit of review

Evaluation

In homework, you’ve seen that:

- training data is usually separate from **test data**
- training accuracy is often much higher than test accuracy
  - Training accuracy is what your classifier is optimizing for (plus regularization), but not a good indicator of how it will perform

Evaluation

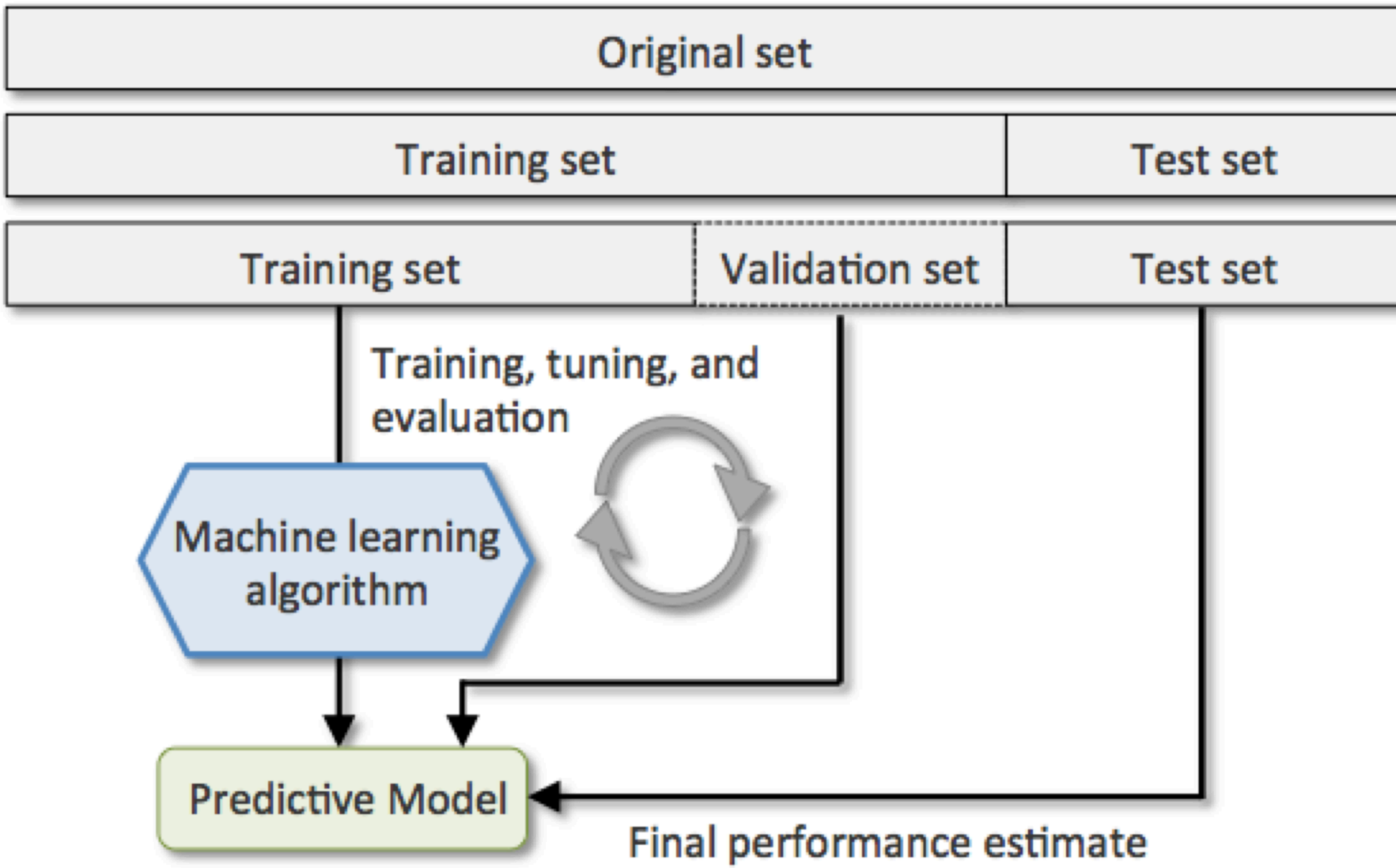
Distinction between:

- **in-sample** data
  - The data that is available when building your model
  - “Training” data in machine learning terminology
- **out-of-sample** data
  - Data that was not seen during training
  - Also called **held-out data** or a **holdout set**
  - Useful to see what your classifier will do on data it hasn’t seen before
  - Usually assumed to be from the same distribution as in-sample data

Evaluation

- Ideally, you should be “blind” to the test data until you are ready to evaluate your final model
- Often you need to evaluate a model repeatedly (e.g., you’re trying to pick the best regularization strength, and you want to see how different values affect the performance)
- So you set aside part of training data as a validation set, or perform cross-validation

Evaluation



Original set
Training set
Test set
Training set
Validation set
Test set
Training, tuning, and evaluation
Machine learning algorithm
Predictive Model
Final performance estimate

Cross validation

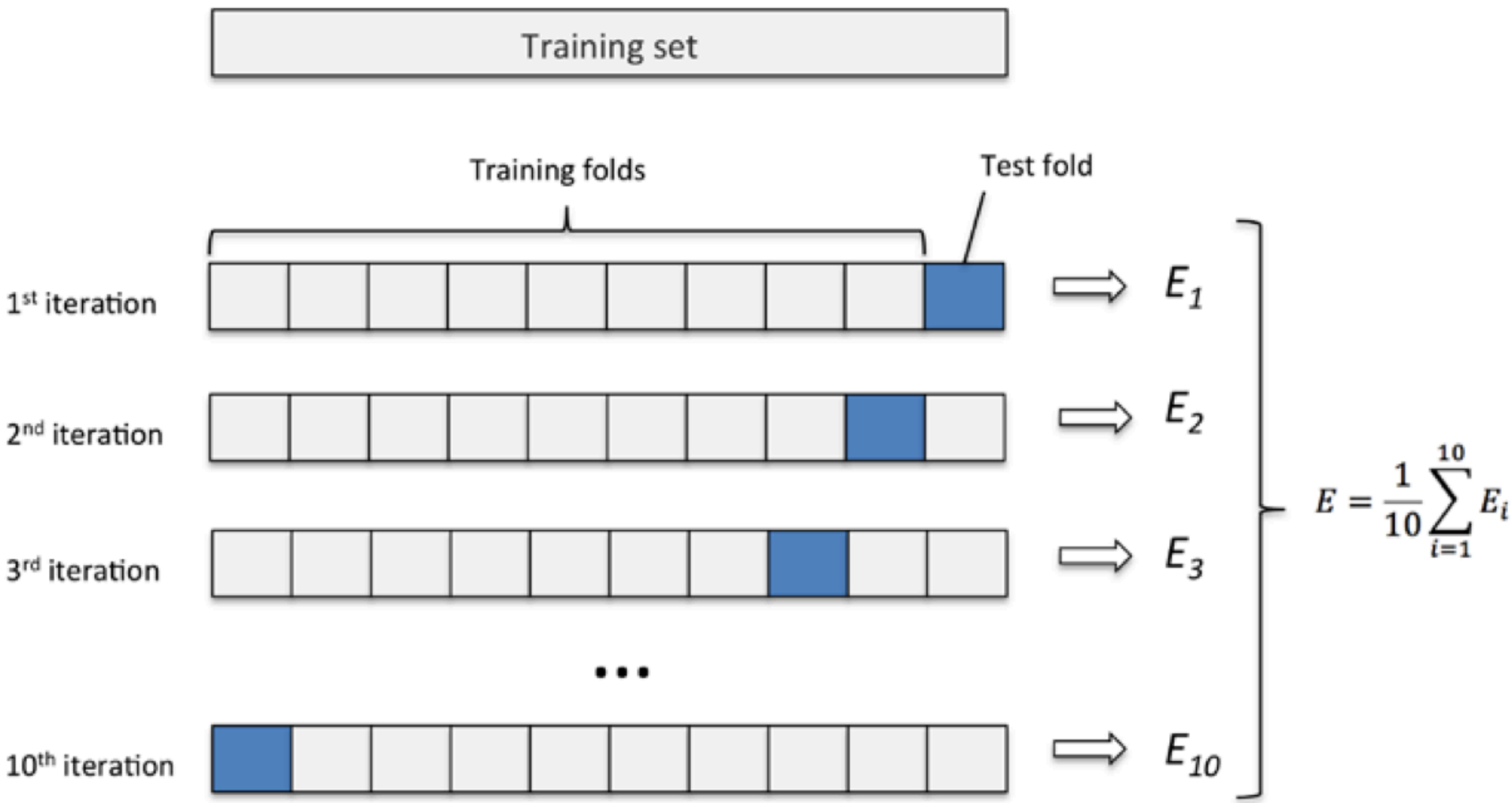


Illustration of 10-fold cross-validation

Splitting your dataset

- Remember to watch for leaks between train in and test. If you already know that person A likes bananas from train, you might be able to infer that person A also likes apples in test.

- This takes experience and thought. Start practicing!

Other considerations

- If test performance looks too good to be true, it probably is.
- Watch out for bugs and dumb errors.
- This is very common everywhere, including in fancy CS research.
- Because ML algorithms are often stochastic, debugging can be trickier

Evaluation Metrics

How do we measure performance?

What *metrics* should be used?

Evaluation Metrics

So far, we have mostly talked about **accuracy** in this class

- The number of correctly classified instances divided by the total number of instances

Error is the complement of accuracy

- Accuracy = 1 – Error
- Error = 1 – Accuracy

Evaluation Metrics

Accuracy/error give an overall summary of model performance, though sometimes hard to interpret

Example: fraud detection in bank transactions

- 99.9% of instances are legitimate
- A classifier that never predicts fraud would have an accuracy of 99.9%
- Need a better way to understand performance

Evaluation Metrics

Some metrics measure performance with respect to a particular class

With respect to a class *c*, we define a prediction as:

- True positive: the label is c and the classifier predicted c
- False positive: the label is not c but the classifier predicted c
- True negative: the label is not c and the classifier did not predict c
- False negative: the label is c but the classifier did not predict c

Evaluation Metrics

Two different types of errors:

- False positive (“type I” error)
- False negative (“type II” error)

Usually there is a tradeoff between these two

- Can optimize for one at the expense of the other
- Which one to favor? Depends on task

Evaluation Metrics

Precision is the percentage of instances predicted to be positive that were actually positive

$$PRE = \frac{TP}{TP + FP}$$

Fraud example:

- Low precision means you are classifying legitimate transactions as fraudulent

Evaluation Metrics

Recall is the percentage of positive instances that were predicted to be positive

$$REC = TPR = \frac{TP}{P} = \frac{TP}{FN + TP}$$

Fraud example:

- Low recall means there are fraudulent transactions that you aren't detecting

Precision and recall

- Today, just focus on understanding what they are
- Later this week we will do some whiteboard exercises to build intuition for these numbers
- There is a deep, deep tradeoff between making wrong predictions and missing things.
- Ideally, you want as many of both as possible. But in reality you have to choose
- Good practitioners should be able to help non-technical people understand and make this choice

Evaluation Metrics

Similar to optimizing for false positives vs false negatives, there is usually a tradeoff between prediction and recall

- Can increase one at the expense of the other
- One might be more important than the other, or they might be equally important; depends on task

Fraud example:

- If a human is reviewing the transactions flagged as fraudulent, probably optimize for recall
- If the classifications are taken as-is (no human review), probably optimize for precision

Evaluation Metrics

Can modify prediction rule to adjust tradeoff

- Increased prediction threshold (i.e., score or probability of an instance belonging to a class) → increased precision
  - Fewer instances will be predicted positive
  - But the ones that are classified positive are more likely to be classified correctly (more confidence classifier)
- Decreased threshold → increased recall
  - More instances will get classified as positive (the bar has been lowered)
  - But this will make your classifications less accurate, lower precision

Evaluation Metrics

The **F1 score** is an average of precision and recall

- Used as a summary of performance, still with respect to a particular class *c*
- Defined using *harmonic* mean, affected more by lower number
  - Both numbers have to be high for F1 to be high
  - F1 is therefore useful when both are important

$$F1 = 2\frac{PRE \times REC}{PRE + REC}$$

Evaluation Metrics

Which metrics to use?

- Accuracy easier to understand/communicate than precision/recall/F1, but harder to interpret correctly
- Precision/recall/F1 generally more informative
  - If you have a small number of classes, just show P/R/F for all classes
  - If you have a large number of classes, then probably should do macro/micro averaging
- F1 better if precision/recall both important, but sometimes you might highlight one over the other

Evaluation Metrics

It is often a good idea to contextualize your results by comparing to a *baseline* level of performance

- A “dummy” baseline, like always outputting the majority class, can be useful to understand if your data is imbalanced (like in fraud example)
- A simple, “default” classifier for your task, like using standard 1-gram features for text, can help you understand if your modifications resulted in an improvement

Model Selection

Evaluation can help you choose (or *select*) between competing models

- Which classification algorithm to use?
- Best preprocessing steps?
- Best feature set?
- Best hyperparameters?

Usually these are all decided *empirically* based on testing different possibilities on your data

Model Selection

Selecting your model to get optimal test performance is risky

- What works best for the particular test set might not actually be the best in general
- Overfitting to the test data

Validation (or **development**) data refers to data that is held-out for measuring performance, but is separate from the final test set

Model Selection

What settings should you test? Lots of possibilities.

Different decisions/hyperparameters depend on each other; not independent

- e.g., optimal regularization strength depends on what kind of regularizer (L2 vs L1), what kind of feature selection, etc

Best to optimize *combinations* of settings, rather than optimizing individually

Model Selection

A **grid search** is the process of evaluating every combination of settings (from a specified set of potential values) on validation data

Quickly becomes expensive… example:

5 regularization values
2 regularizer types
3 kernel settings
5 feature selection settings
2 preprocessing options
= 300 combinations

Model Selection

Might only perform a grid search on a subset of combinations initially

One option: start by searching a small number of very different values (e.g., C={0.1, 1.0, 10.0, 100.0}), then fine-tune with more values close to the optimal one

(e.g. find that C=1.0 and C=10.0 are the best of the above options, so now try C={1.0,2.0,4.0,6.0,8,0,…})

Understanding and Diagnosis

In addition to *measuring* performance, also important to *understand* performance

Want to be able to answer:

- Why does the model make the predictions it does?
- What kinds of errors does the model make?
- What can be done to improve the model?

Error Analysis

What kinds of mistakes does a model make?

Which classes does a classifier tend to mix up?

Error Analysis

Error Analysis

A **confusion matrix** (or **error matrix**) is a table that counts the number of test instances with each true label vs each predicted label.

		Predicted			
		Iris-setosa	Iris-versicolor	Iris-virginica	Σ
Actual	Iris-setosa	100.0 %	0.0 %	0.0 %	50
	Iris-versicolor	0.0 %	88.7 %	6.4 %	50
	Iris-virginica	0.0 %	11.3 %	93.6 %	50
	Σ	50	53	47	150

From: https://docs.orange.biolab.si/3/visual-programming/widgets/evaluation/confusionmatrix.html

Error Analysis

A **confusion matrix** (or **error matrix**) is a table that counts the number of test instances with each true label vs each predicted label.

- In binary classification, the confusion matrix is just a 2x2 table of true/false positive/negatives
- But can be generalized to multiclass settings

Error Analysis

Just as false negatives vs false positives are two different types of errors where one may be preferable, different types of multiclass errors may have different importance

- Mistaking a deer for an antelope is not such a bad mistake
- Mistaking a deer for a cereal box would be an odd mistake

Are the mistakes acceptable? Need to look at confusions, not just a summary statistic.

Error Analysis

Another way to understand why your classifier is behaving a certain way is to examine the parameters that are learned after training

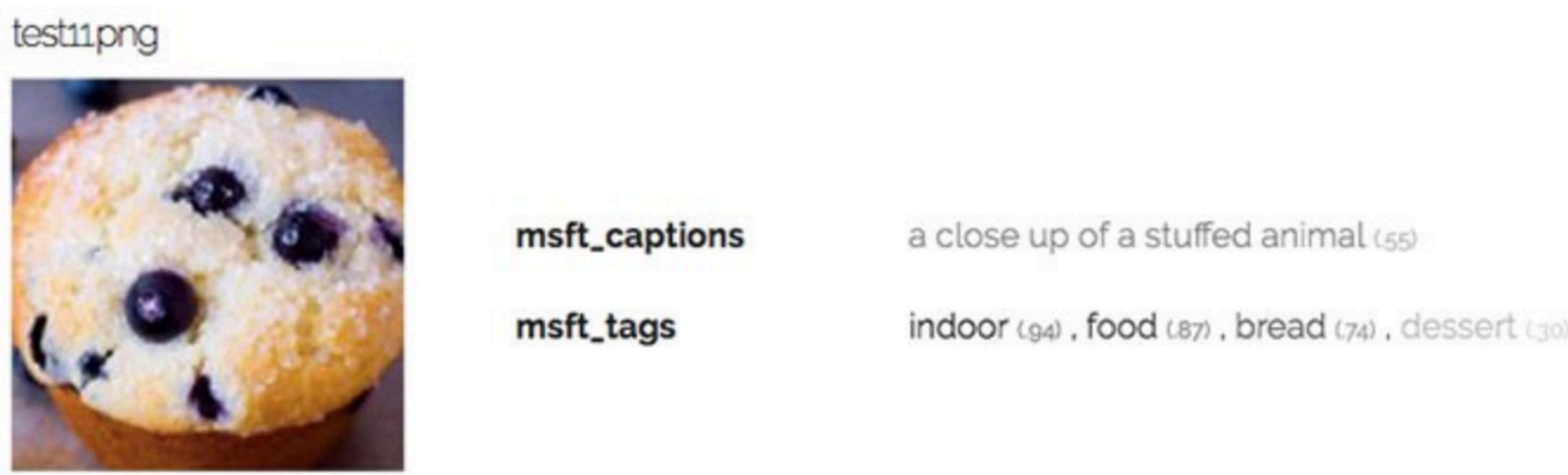
- e.g., the decision tree structure, or the weight vector values

If features are associated with classes in a way that doesn't make sense to you, that might mean the model is not working the way you intended

- Caveat: features interact in ways that can be hard to understand, so unintuitive parameters not necessarily wrong

Error Analysis

Another good practice: look at a sample of misclassified instances



From: https://medium.freecodecamp.org/chihuahua-or-muffin-my-search-for-the-best-computer-vision-api-cbda4d6b425d

Error Analysis

Another good practice: look at a sample of misclassified instances

- Might help you understand why the classifier is making those mistakes
- Might help you understand what kinds of instances the classifier makes mistakes on
  - Maybe they were ambiguous to begin with, so not surprising the classifier had trouble

Error Analysis

Another good practice: look at a sample of misclassified instances

- If you have multiple models, can compare how they do on individual instances
- If your model outputs probabilities, useful to examine
  - If the correct class was the 2nd most probable, that's a better mistake than if it was the 10th most probable

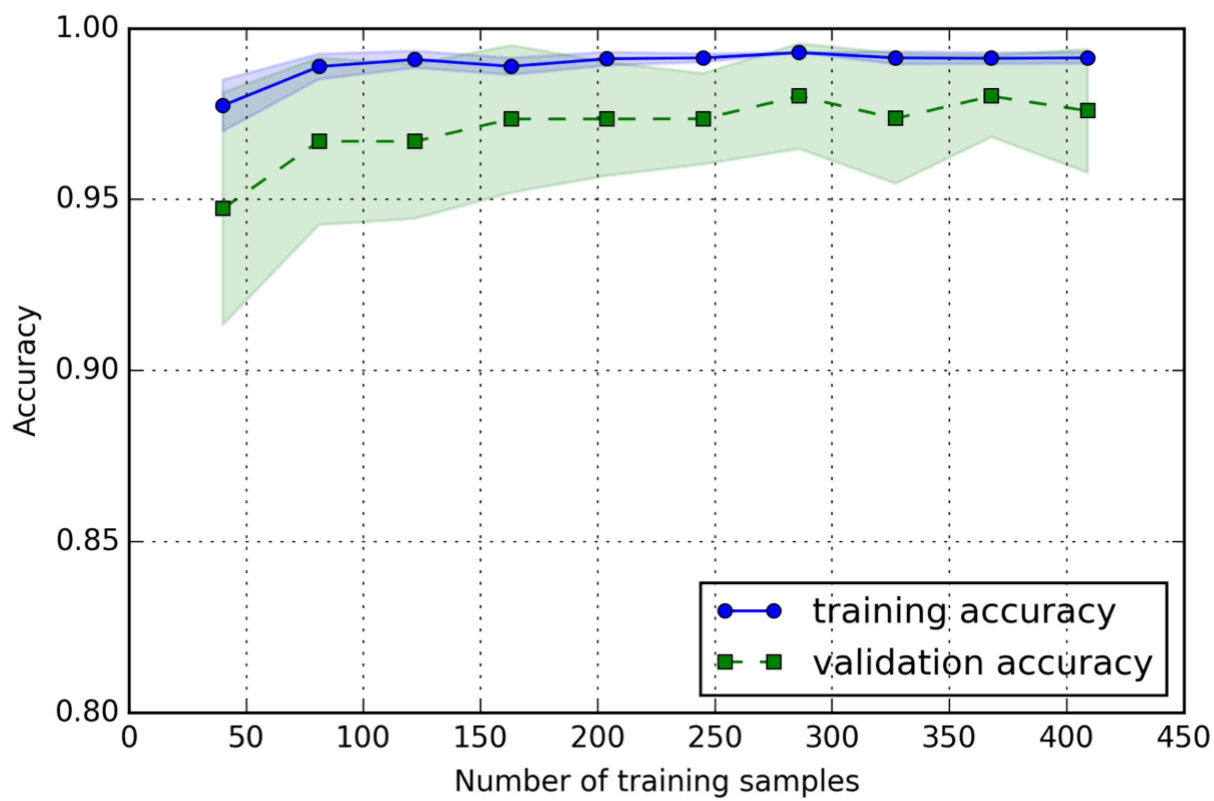
Error Analysis

Error analysis can help inform:

- Feature engineering
  - If you observe that certain classes are more easily confused, think about creating new features that could distinguish those classes
- Feature selection
  - If you observe that certain features might be hurting performance (maybe the classifier is picking up on an association between a feature and a class that isn’t meaningful), you could remove it

Learning Curves

A **learning curve** measures the performance of a model at different amounts of training data



Learning Curves

Primarily used to understand two things:

- How much training data is needed?
- Bias/variance tradeoff

Learning Curves

How much training data is needed?

Usually the validation accuracy increases noticeable after an initial increase in training data, then levels off after a while

- Might still be increasing, but with diminishing returns

The learning curve can be used to predict the benefit you'll get from adding more training data

- Still increasing rapidly → get more data
- Completely flattened out → more data won't help
- Gradually increasing → need a lot more data to help

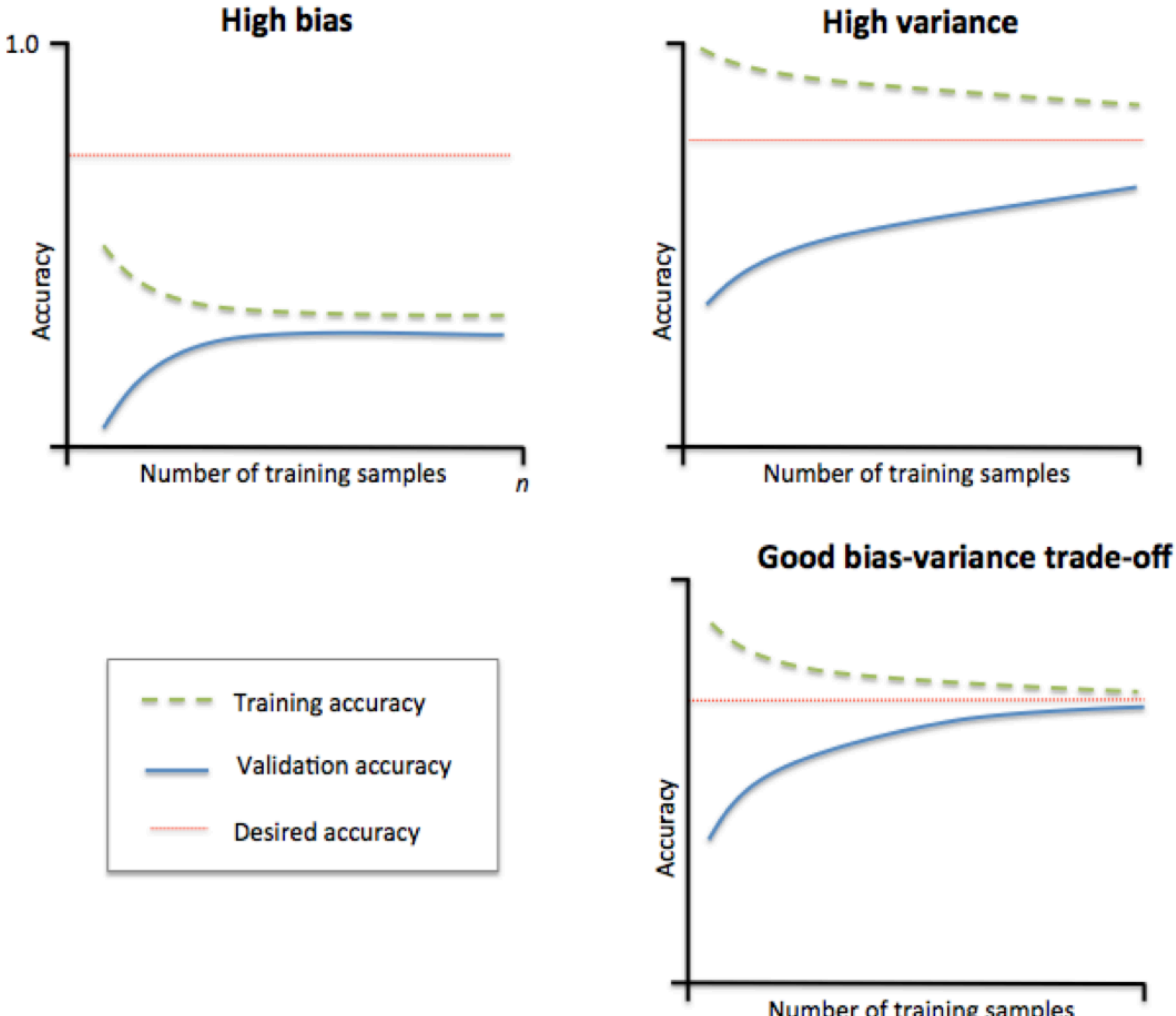
Learning Curves

A typical pattern is that:

- Training accuracy decreases with more data
- Validation accuracy increases
- The two accuracies should converge to be similar

Something to look for:

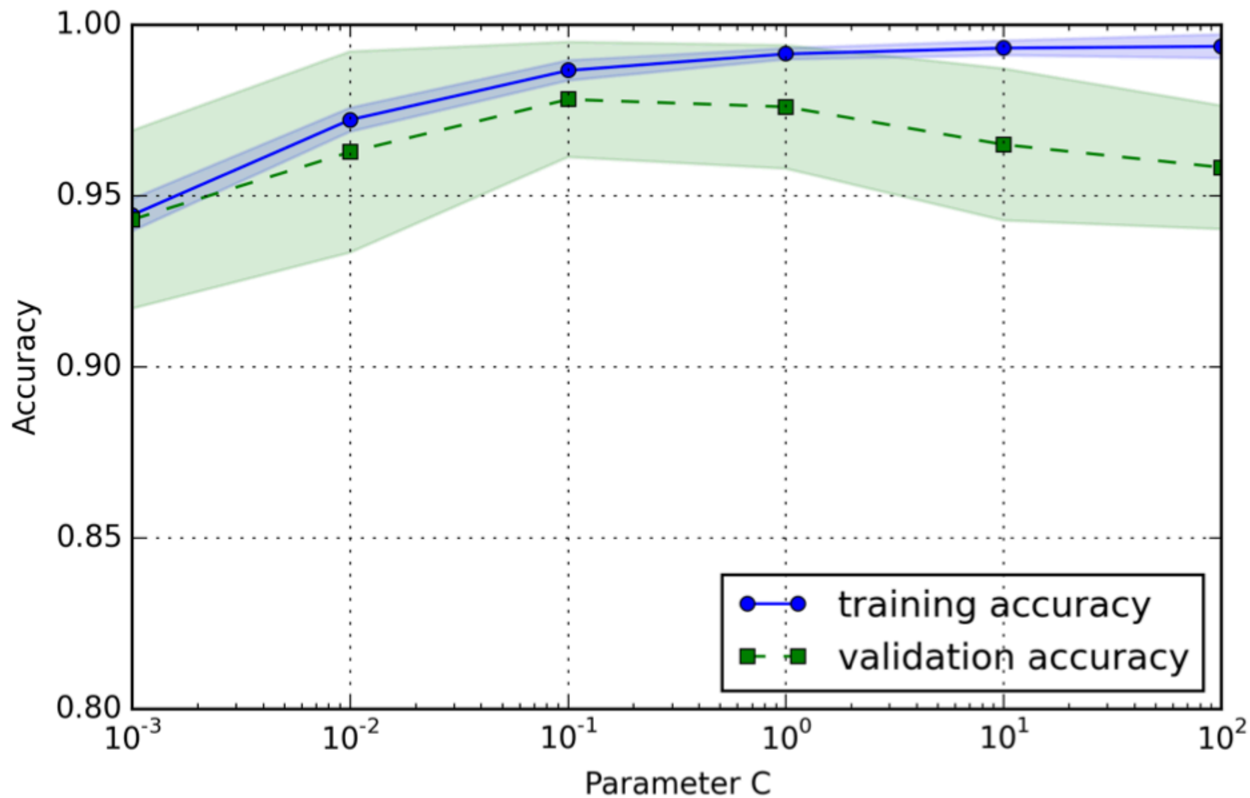
- If gap between train/validation performance isn't closing, probably too much variance (overfitting)
- If gap between train/validation closes quickly, might suggest high bias (underfitting)



High bias
1.0
Accuracy
Number of training samples
n
High variance
Accuracy
Number of training samples
Training accuracy
Validation accuracy
Desired accuracy
Good bias-variance trade-off
Accuracy
Number of training samples

Validation Curves

A **validation curve** measures the performance of a model at different hyperparameter settings

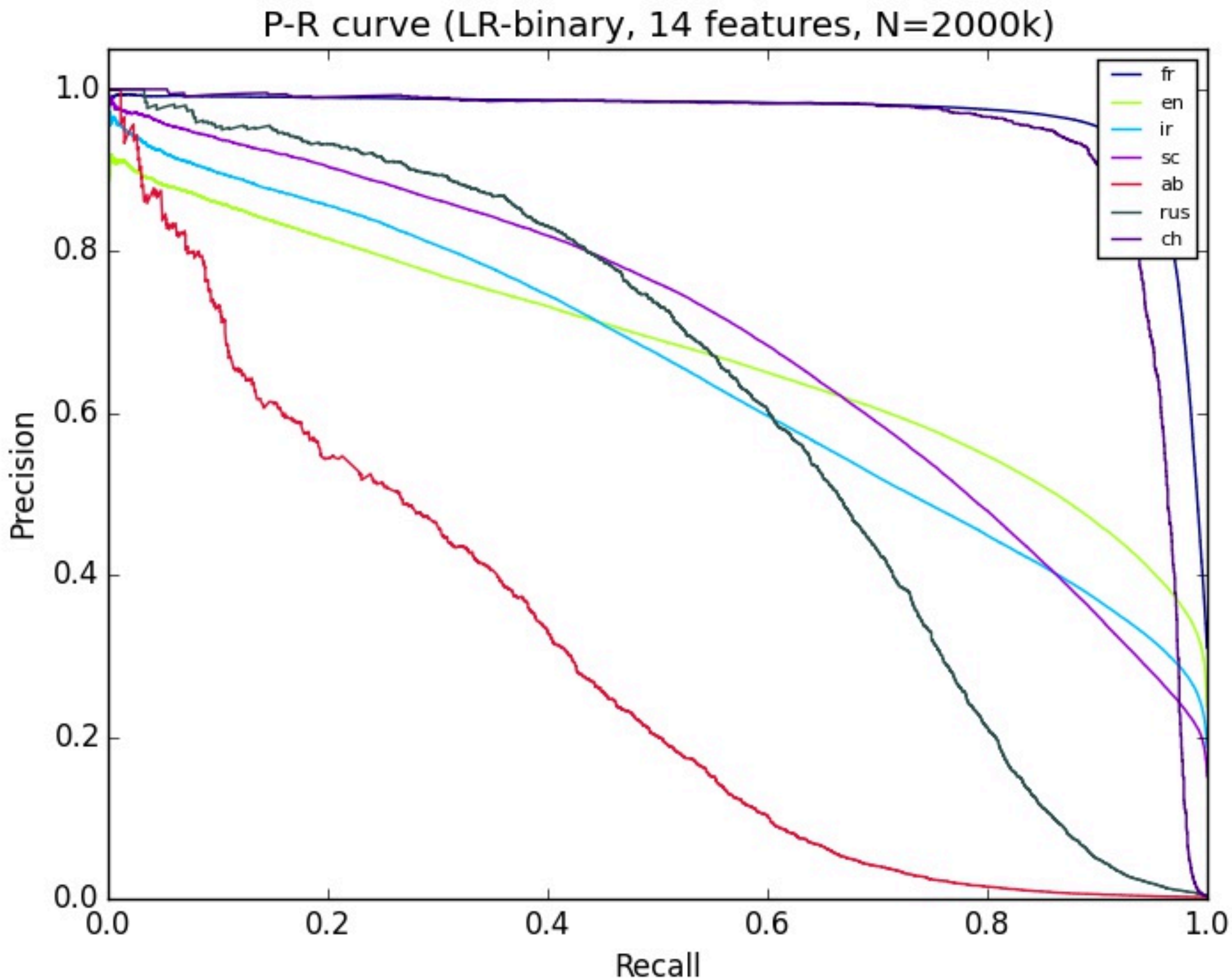


Validation Curves

Validation curves help you understand the effect of hyperparameters, and also help understand the bias/variance tradeoff

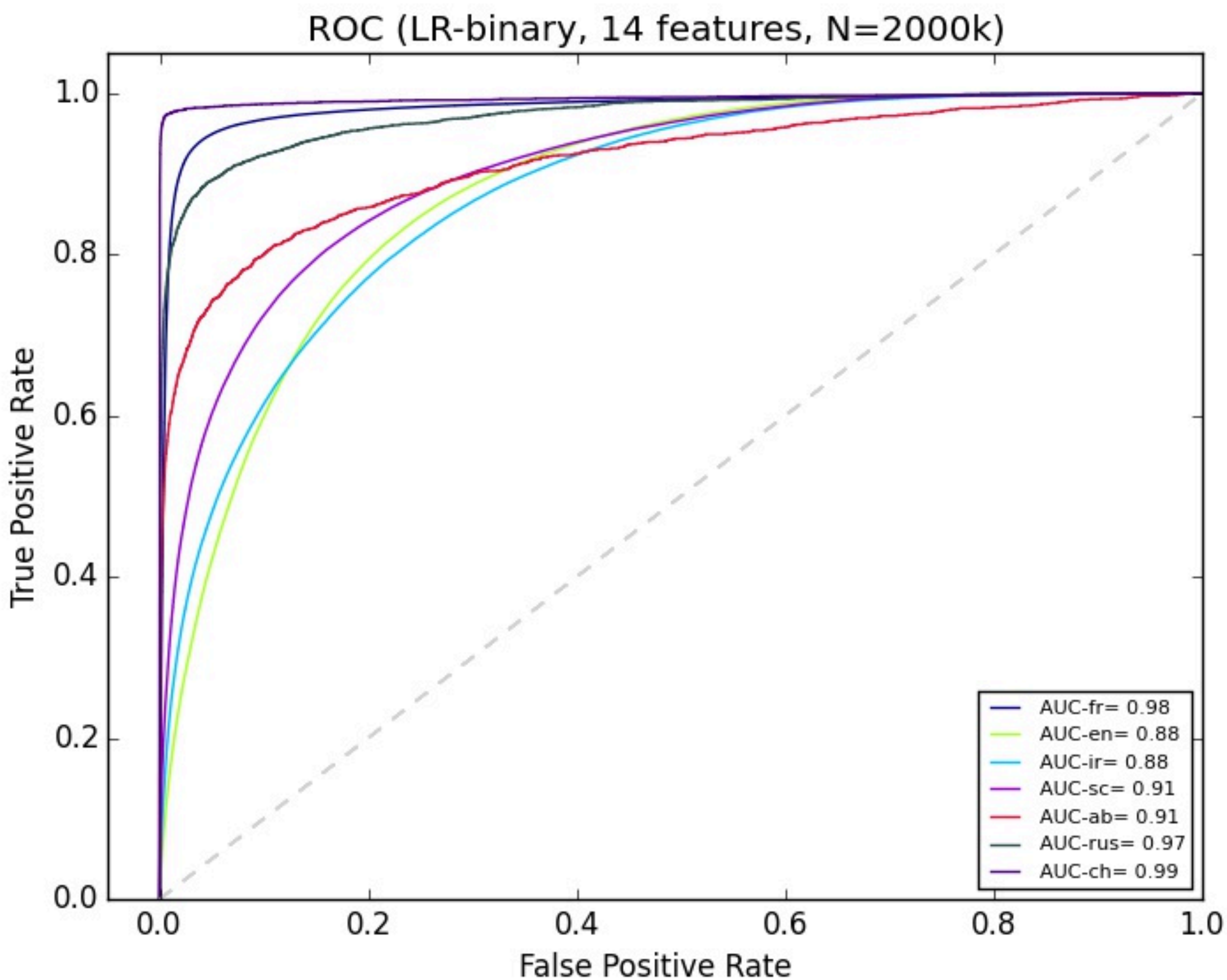
- Want to find a setting where train/validation performance is similar (low variance)
- Of the settings where train/validation performance are similar, pick the one with the highest validation accuracy (low bias)

Precision-Recall Curve



From: https://stackoverflow.com/questions/33294574/good-roc-curve-but-poor-precision-recall-curve

ROC Curve



From: https://stackoverflow.com/questions/33294574/good-roc-curve-but-poor-precision-recall-curve

Debugging

Lots of places in the pipeline where there could be an implementation mistake:

- Data preprocessing
- Feature engineering
- Training algorithm
- Validation pipeline

Best to start simple, compare to existing data/systems, then expand from there