COMP 551 – Applied Machine Learning Lecture 6: Performance evaluation. Model assessment and selection.

Instructor: Joelle Pineau (*jpineau@cs.mcgill.ca)*

Class web page: *www.cs.mcgill.ca/~jpineau/comp551*

Today's quiz (on myCourses)

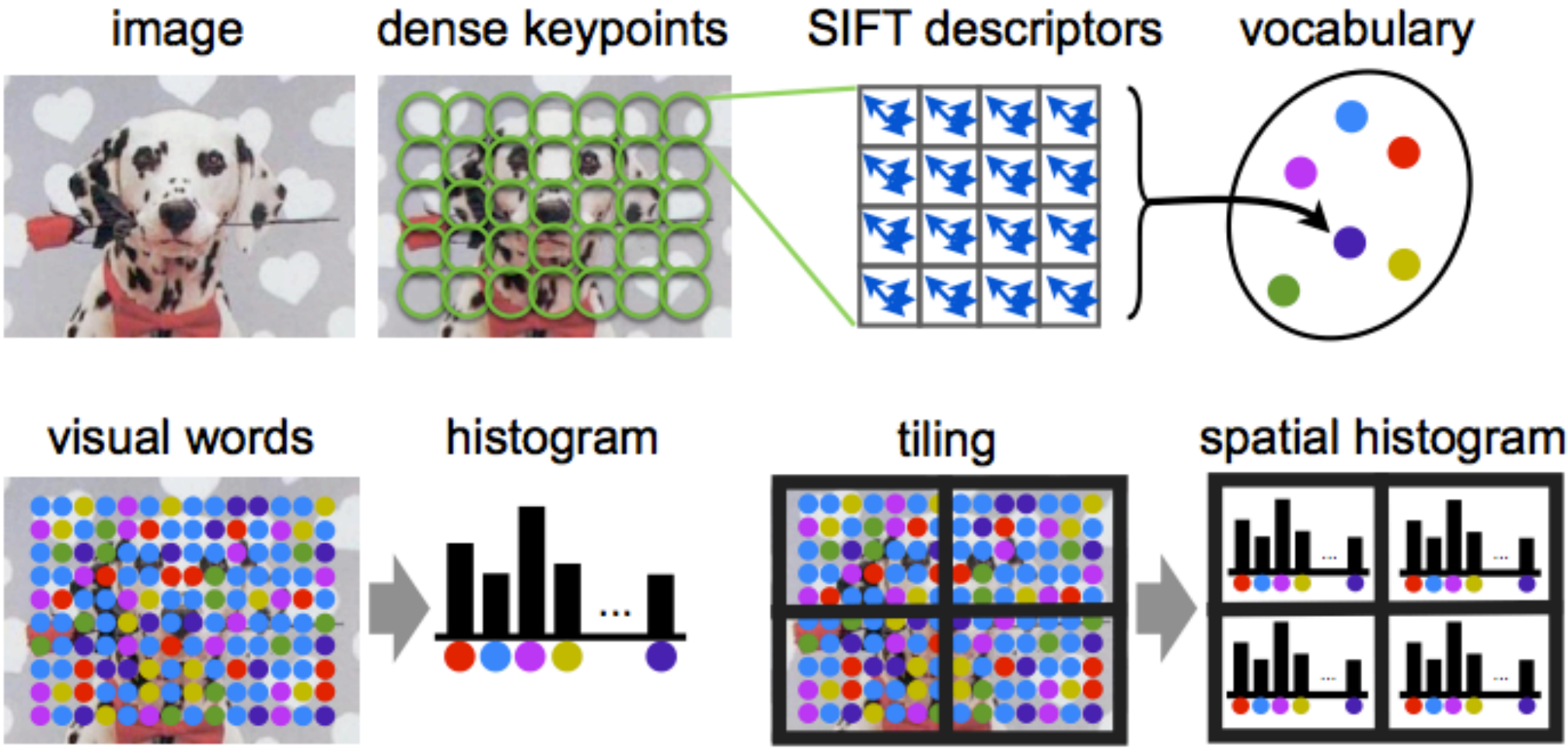
1. Name one advantage of LDA over Naive Bayes.

2. Name one disadvantage of LDA over Naive Bayes.

3. **True** or **False**: Generative learning typically requires learning more parameters than discriminative learning (assuming the same number of features and examples).

4. Why?

Real-world classification tasks



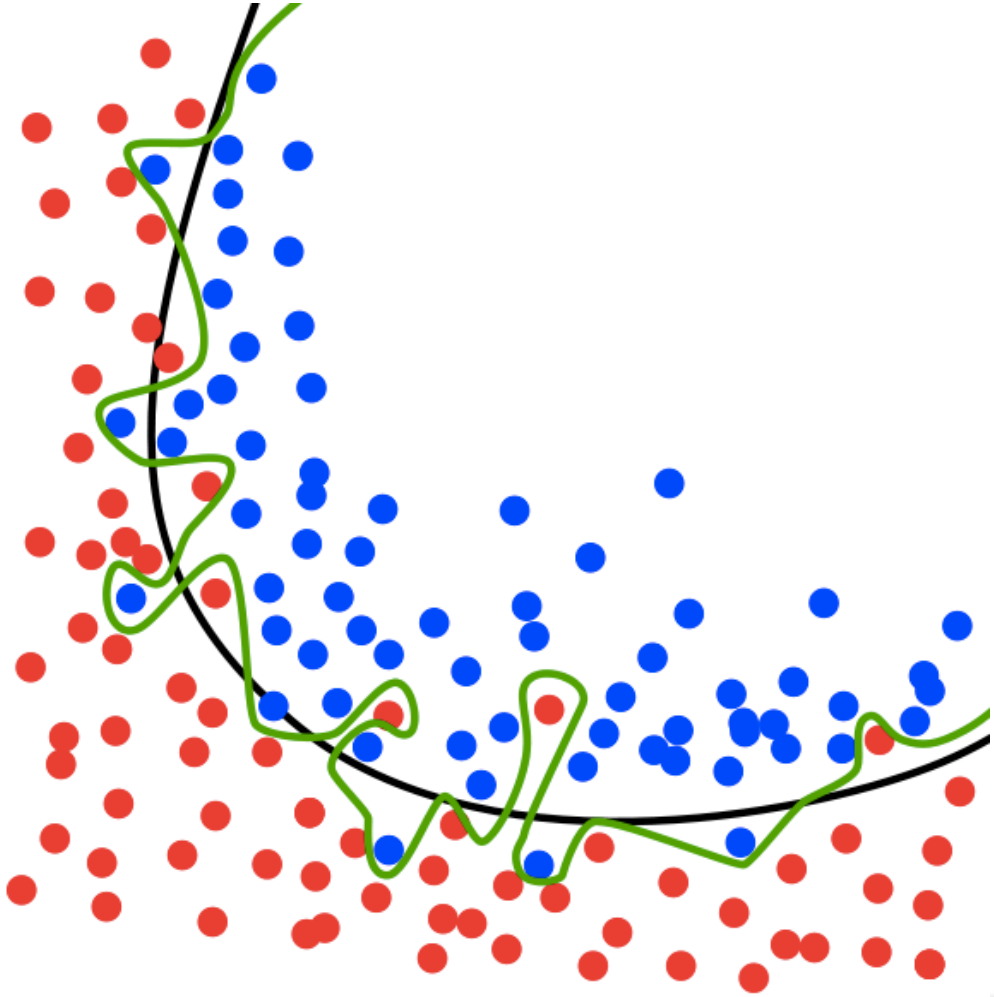
http://www.di.ens.fr/willow/events/cvml2011/materials/practical-classification/

Evaluating performance

- Different objectives:
  - Selecting the right model for a problem.
  - Testing performance of a new algorithm.
  - Evaluating impact on a new application.

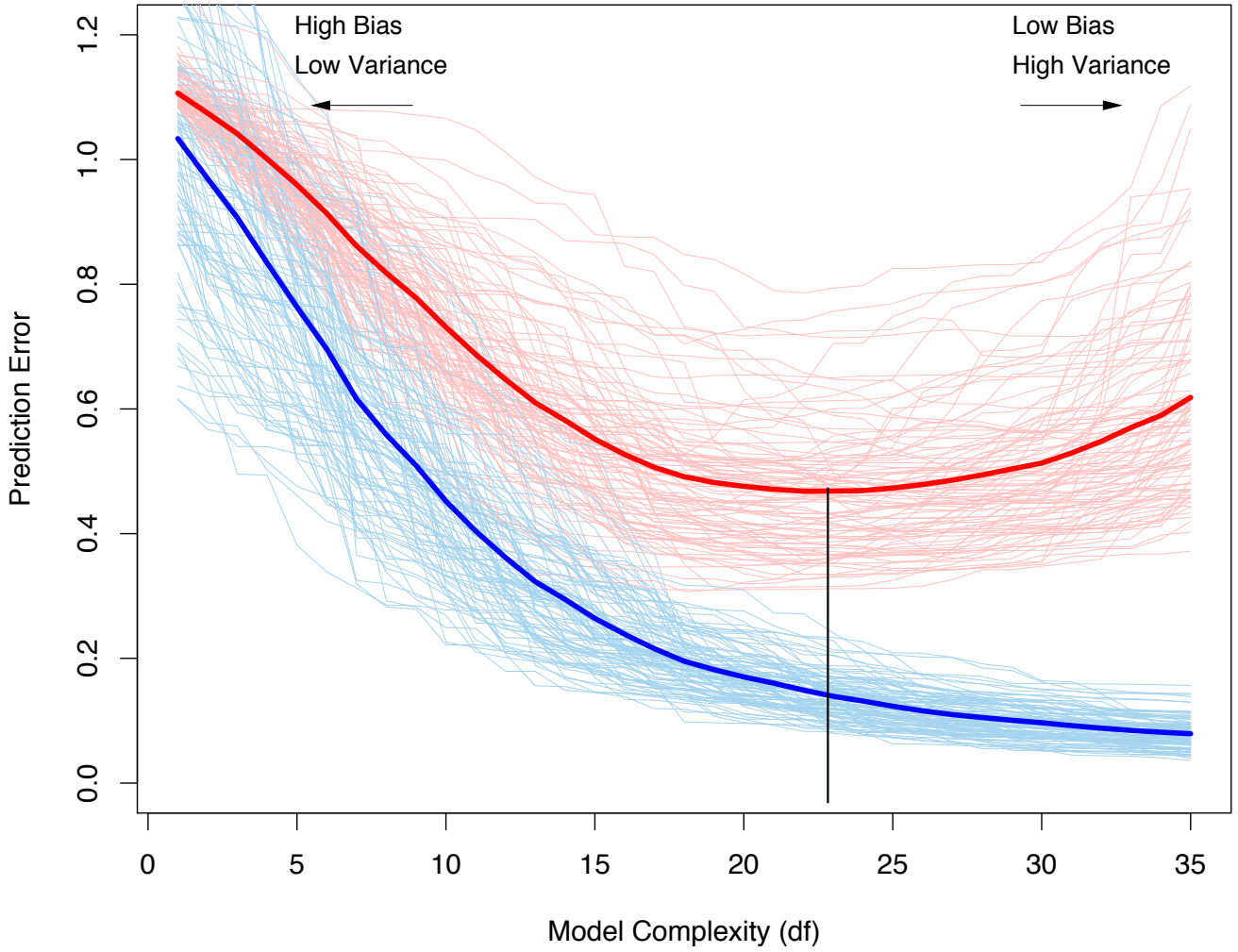
Overfitting

- Adding more degrees of freedom (more features) always seems to improve the solution!



Minimizing the error

- Find the low point in the validation error:



Validation error

Train error

Performance metrics for classification

- Not all errors have equal impact!

- There are different types of mistakes, particularly in the classification setting.

Example 1

Int J Neural Syst. 2013 Oct;23(5):1350023. doi: 10.1142/S0129065713500238. Epub 2013 Jul 21.

Application of intrinsic time-scale decomposition (ITD) to EEG signals for automated seizure prediction.

Martis RJ[1], Acharya UR, Tan JH, Petznick A, Tong L, Chua CK, Ng EY.

⊕ **Author information**

Abstract

Intrinsic time-scale decomposition (ITD) is a new nonlinear method of time-frequency representation which can decipher the minute changes in the nonlinear EEG signals. In this work, we have automatically classified normal, interictal and ictal EEG signals using the features derived from the ITD representation. The energy, fractal dimension and sample entropy features computed on ITD representation coupled with decision tree classifier has yielded an average classification accuracy of 95.67%, sensitivity and specificity of 99% and 99.5%, respectively using 10-fold cross validation scheme. With application of the nonlinear ITD representation, along with conceptual advancement and improvement of the accuracy, the developed system is clinically ready for mass screening in resource constrained and emerging economy scenarios.

Example 1

Int J Neural Syst. 2013 Oct;23(5):1350023. doi: 10.1142/S0129065713500238. Epub 2013 Jul 21.

Application of intrinsic time-scale decomposition (ITD) to EEG signals for automated seizure prediction.

Martis RJ[1], Acharya UR, Tan JH, Petznick A, Tong L, Chua CK, Ng EY.

⊕ **Author information**

Abstract

Intrinsic time-scale decomposition (ITD) is a new nonlinear method of time-frequency representation which can decipher the minute changes in the nonlinear EEG signals. In this work, we have automatically classified normal, interictal and ictal EEG signals using the features derived from the ITD representation. The energy, fractal dimension and sample entropy features computed on ITD representation coupled with decision tree classifier has yielded an average classification accuracy of 95.67%, sensitivity and specificity of 99% and 99.5%, respectively using 10-fold cross validation scheme. With application of the nonlinear ITD representation, along with conceptual advancement and improvement of the accuracy, the developed system is clinically ready for mass screening in resource constrained and emerging economy scenarios.

Accuracy = True positives + True Negatives / Total number of examples
Sensitivity = True positives / Total number of actual positives
Specificity = True negatives / Total number of actual negatives

Performance metrics for classification

- Not all errors have equal impact!

- There are different types of mistakes, particularly in the classification setting.
  - E.g. Consider the diagnostic of a disease. Two types of mis-diagnostics:
    - Patient does not have disease but received positive diagnostic (Type I error);
    - Patient has disease but it was not detected (Type II error).

Performance metrics for classification

- Not all errors have equal impact!

- There are different types of mistakes, particularly in the classification setting.
  - E.g. Consider the diagnostic of a disease. Two types of mis-diagnostics:
    - Patient does not have disease but received positive diagnostic (Type I error);
    - Patient has disease but it was not detected (Type II error).
  - E.g. Consider the problem of spam classification:
    - A message that is not spam is assigned to the spam folder (Type I error);
    - A message that is spam appears in the regular folder (Type II error).

Performance metrics for classification

- Not all errors have equal impact!

- There are different types of mistakes, particularly in the classification setting.
  - E.g. Consider the diagnostic of a disease. Two types of mis-diagnostics:
    - Patient does not have disease but received positive diagnostic (Type I error);
    - Patient has disease but it was not detected (Type II error).
  - E.g. Consider the problem of spam classification:
    - A message that is not spam is assigned to the spam folder (Type I error);
    - A message that is spam appears in the regular folder (Type II error).

- How many Type I errors are you willing to tolerate, for a reasonable rate of Type II errors ?

Example 2

Clin Neurophysiol. 2013 Sep;124(9):1745-54. doi: 10.1016/j.clinph.2013.04.006. Epub 2013 May 3.

Seizure prediction in patients with mesial temporal lobe epilepsy using EEG measures of state similarity.

Gadhoumi K[1], Lina JM, Gotman J.

Author information

Abstract

OBJECTIVES: In patients with intractable epilepsy, predicting seizures above chance and with clinically acceptable performance has yet to be demonstrated. In this study, an intracranial EEG-based seizure prediction method using measures of similarity with a reference state is proposed.

METHODS: 1565 h of continuous intracranial EEG data from 17 patients with mesial temporal lobe epilepsy were investigated. The recordings included 175 seizures. In each patient the data was split into a training set and a testing set. EEG segments were analyzed using continuous wavelet transform. During training, a reference state was defined in the immediate preictal data and used to derive three features quantifying the discrimination between preictal and interictal states. A classifier was then trained in the feature space. Its performance was assessed using testing set and compared with a random predictor for statistical validation.

RESULTS: Better than random prediction performance was achieved in 7 patients. The sensitivity was higher than 85%, the warning rate was less than 0.35/h and the proportion of time under warning was less than 30%.

CONCLUSION: Seizures are predicted above chance in 41% of patients using measures of state similarity.

SIGNIFICANCE: Sensitivity and specificity levels are potentially interesting for closed-loop seizure control applications.

Example 3

Conf Proc IEEE Eng Med Biol Soc. 2012;2012:1061-4. doi: 10.1109/EMBC.2012.6346117.

Low complexity algorithm for seizure prediction using Adaboost.

Ayinala M[1], Parhi KK.

⊕ Author information

Abstract

This paper presents a novel low-complexity patient-specific algorithm for seizure prediction. Adaboost algorithm is used in two stages of the algorithm: feature selection and classification. The algorithm extracts spectral power features in 9 different sub-bands from the electroencephalogram (EEG) recordings. We have proposed a new feature ranking method to rank the features. The key (top ranked) features are used to make a prediction on the seizure event. Further, to reduce the complexity of classification stage, a non-linear classifier is built based on the Adaboost algorithm using decision stumps (linear classifier) as the base classifier. The proposed algorithm achieves a sensitivity of 94.375% for a total of 71 seizure events with a low false alarm rate of 0.13 per hour and 6.5% of time spent in false alarms using an average of 5 features for the Freiburg database. The low computational complexity of the proposed algorithm makes it suitable for an implantable device.

PMID: 23366078 [PubMed - indexed for MEDLINE]

Terminology

- Type of classification outputs:
  - True positive (m11): Example of class 1 predicted as class 1.
  - False positive (m01): Example of class 0 predicted as class 1. Type 1 error.
  - True negative (m00): Example of class 0 predicted as class 0.
  - False negative (m10): Example of class 1 predicted as class 0. Type II error.
- Total number of instances: m = m00 + m01 + m10 + m11

Terminology

- Type of classification outputs:
  - True positive (m11): Example of class 1 predicted as class 1.
  - False positive (m01): Example of class 0 predicted as class 1. Type 1 error.
  - True negative (m00): Example of class 0 predicted as class 0.
  - False negative (m10): Example of class 1 predicted as class 0. Type II error.
- Total number of instances: m = m00 + m01 + m10 + m11
- Error rate: (m01 + m10) / m
  - If the classes are imbalanced (e.g. 10% from class 1, 90% from class 0), one can achieve low error (e.g. 10%) by classifying everything as coming from class 0!

Confusion matrix

- Many software packages output this matrix.

$$\begin{bmatrix} m_{00} & m_{01} \\ m_{10} & m_{11} \end{bmatrix}$$

Confusion matrix

- Many software packages output this matrix.

$$\begin{bmatrix} m_{00} & m_{01} \\ m_{10} & m_{11} \end{bmatrix}$$

- Be careful! Sometimes the format is slightly different

(E.g. *http://en.wikipedia.org/wiki/Precision_and_recall#Definition_.28classification_context.29*)

	actual class (observation)	
predicted class (expectation)	**tp** (true positive) Correct result	**fp** (false positive) Unexpected result
	fn (false negative) Missing result	**tn** (true negative) Correct absence of result

Common measures

- Accuracy = (TP+ TN) / (TP + FP + FN + TN)
- Precision = True positives / Total number of declared positives
  = TP / (TP+ FP)
- Recall = True positives / Total number of actual positives
  = TP / (TP + FN)

Common measures

- Accuracy = (TP+ TN) / (TP + FP + FN + TN)
- Precision = True positives / Total number of declared positives
  = TP / (TP+ FP)

Text classification

- Recall = True positives / Total number of actual positives
  = TP / (TP + FN)
- Sensitivity is the same as recall.

Medicine

- Specificity = True negatives / Total number of actual negatives
  = TN / (FP + TN)

Common measures

- Accuracy = (TP+ TN) / (TP + FP + FN + TN)
- Precision = True positives / Total number of declared positives
  = TP / (TP+ FP)

Text classification

- Recall = True positives / Total number of actual positives
  = TP / (TP + FN)
- Sensitivity is the same as recall.

Medicine

- Specificity = True negatives / Total number of actual negatives
  = TN / (FP + TN)
- False positive rate = FP / (FP + TN)

Common measures

- Accuracy = (TP+ TN) / (TP + FP + FN + TN)
- Precision = True positives / Total number of declared positives
  = TP / (TP+ FP)

Text classification

- Recall = True positives / Total number of actual positives
  = TP / (TP + FN)
- Sensitivity is the same as recall.

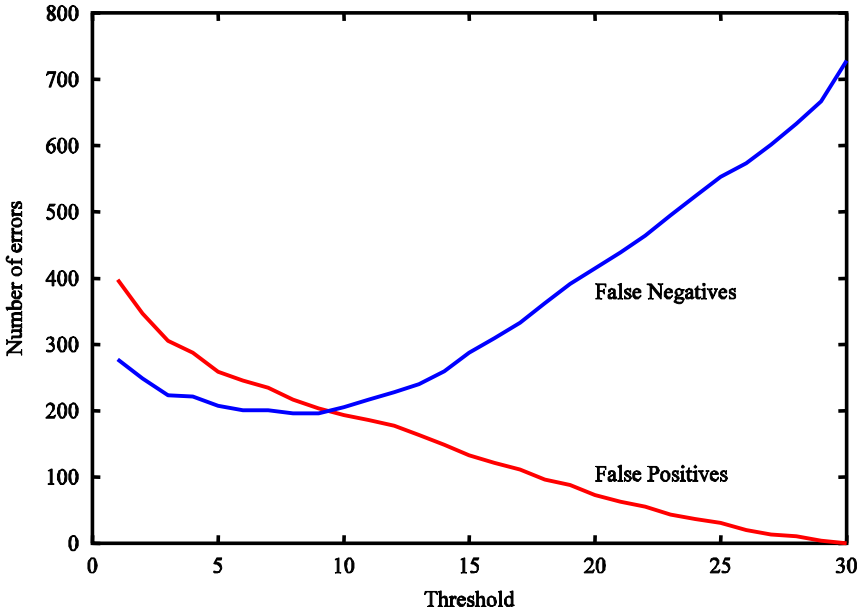
Medicine

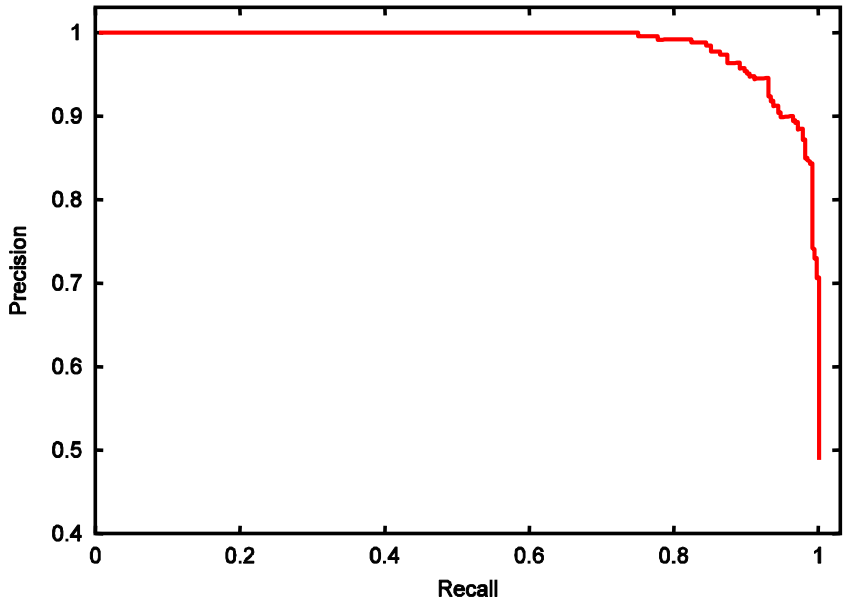
- Specificity = True negatives / Total number of actual negatives
  = TN / (FP + TN)
- False positive rate = FP / (FP + TN)
- F1 measure $$F = 2 \cdot \frac{\text{precision} \cdot \text{recall}}{\text{precision} + \text{recall}}$$

Trade-off

- Often have a trade-off between false positives and false negatives.

  E.g. Consider 30 different classifiers trained on a class. Classify a new sample as positive if K classifiers output positive. Vary K between 0 and 30.





Receiver-operator characteristic (ROC) curve

- Characterizes the performance of a binary classifier over a range of classification thresholds

Data from 4 prediction results:

A

TP=63	FP=28	91
FN=37	TN=72	109
100	100	200

TPR = 0.63
FPR = 0.28
PPV = 0.69
F1 = 0.66
ACC = 0.68

B

TP=77	FP=77	154
FN=23	TN=23	46
100	100	200

TPR = 0.77
FPR = 0.77
PPV = 0.50
F1 = 0.61
ACC = 0.50

C

TP=24	FP=88	112
FN=76	TN=12	88
100	100	200

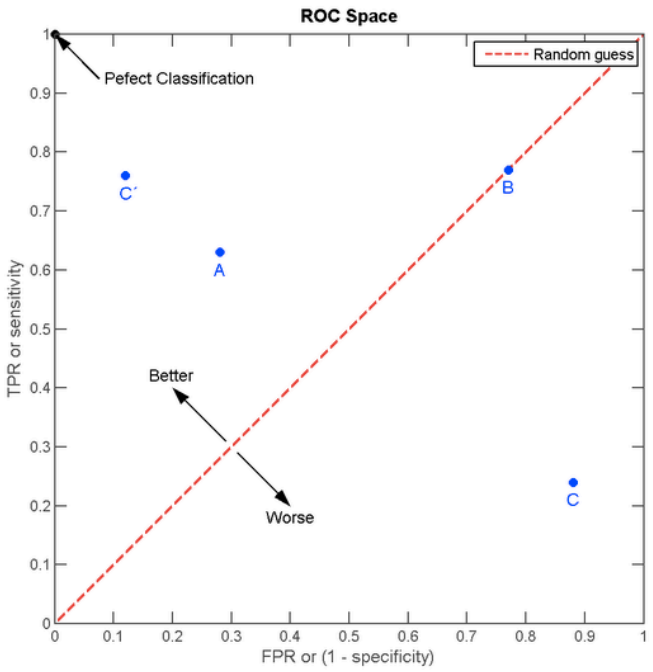
TPR = 0.24
FPR = 0.88
PPV = 0.21
F1 = 0.22
ACC = 0.18

C′

TP=76	FP=12	88
FN=24	TN=88	112
100	100	200

TPR = 0.76
FPR = 0.12
PPV = 0.86
F1 = 0.81
ACC = 0.82

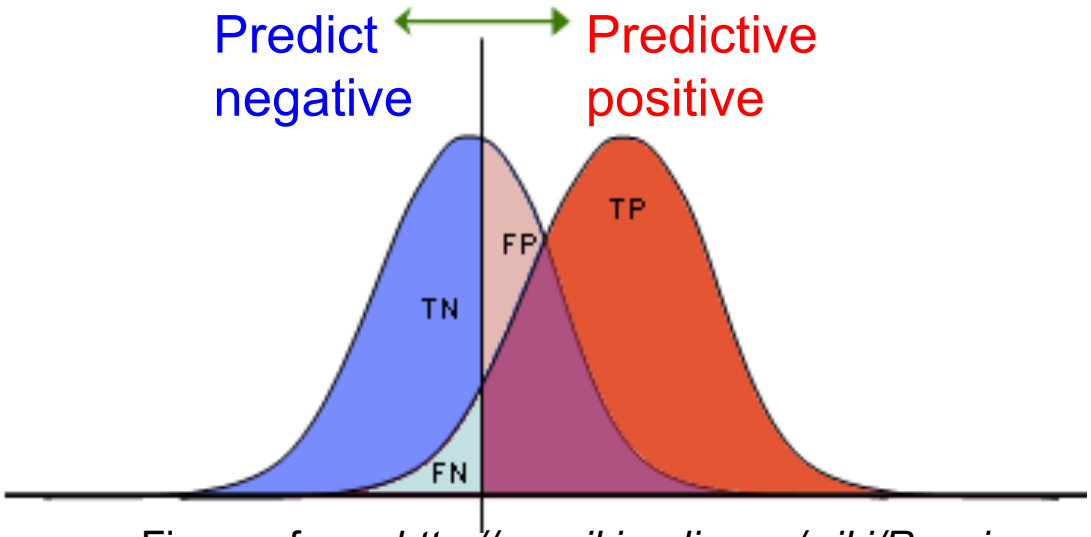
ROC curve:

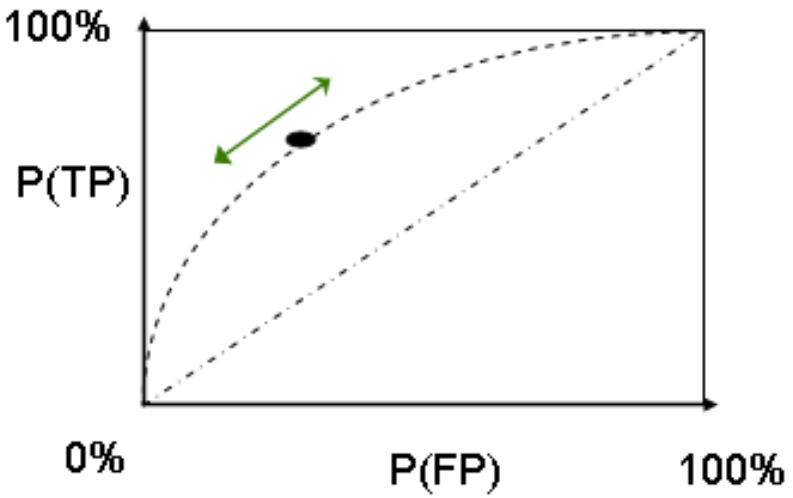


Example from: *http://en.wikipedia.org/wiki/Receiver_operating_characteristic*

Understanding the ROC curve

- Consider a classification problem where data is generated by 2 Gaussians (blue = negative class; red = positive class).
- Consider the decision boundary (shown as a vertical line on the left figure), where you predict Negative on the left of the boundary and predict Positive on the right of the boundary.
- Changing that boundary defines the ROC curve on the right.

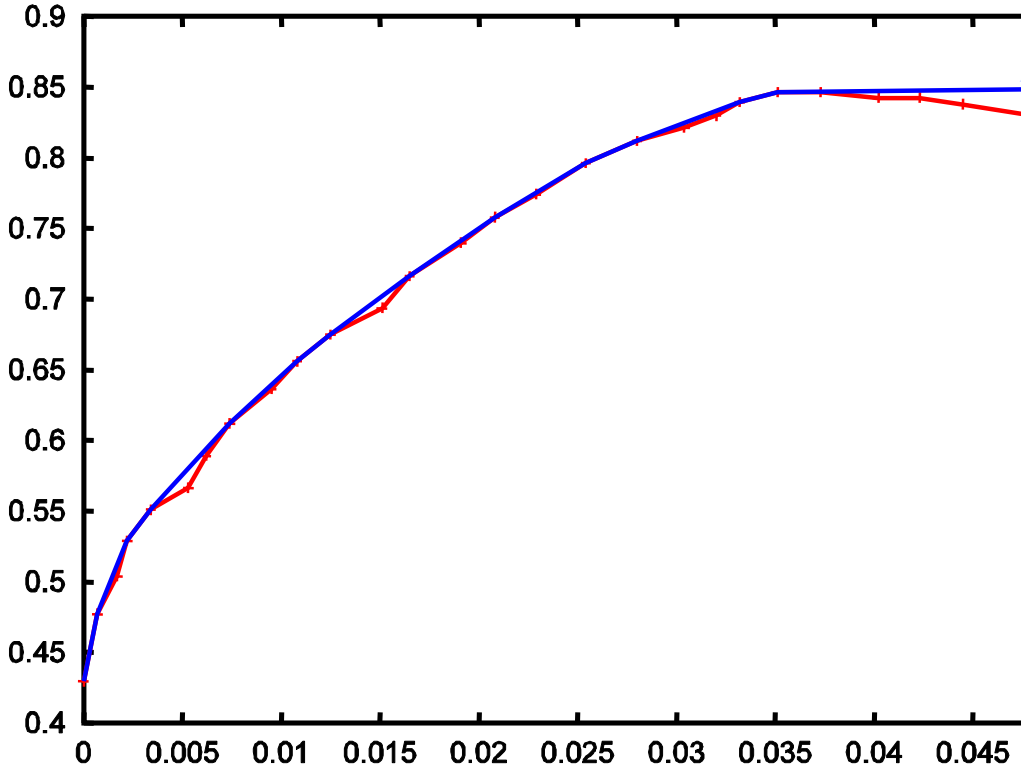




Figures from: *http://en.wikipedia.org/wiki/Receiver_operating_characteristic*

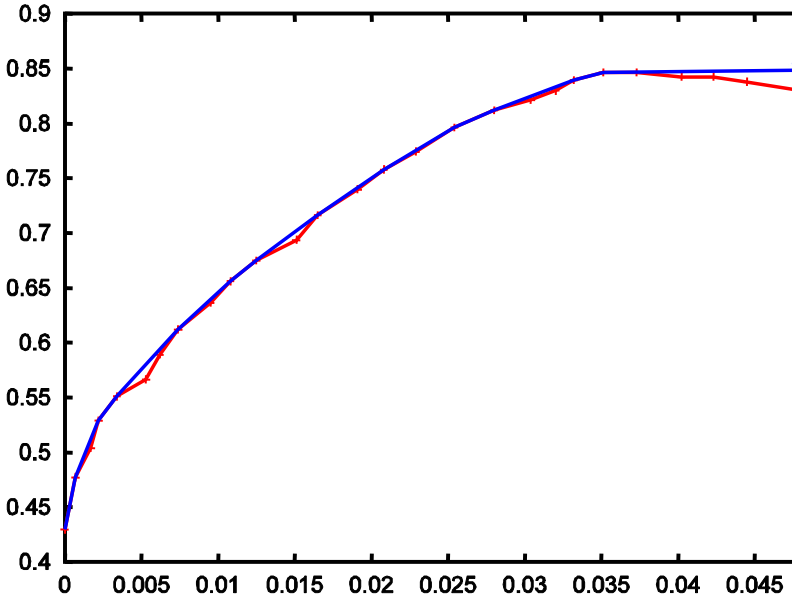
Building the ROC curve

- In many domains, the empirical ROC curve will be non-convex (red line). Take the convex hull of the points (blue line).



Using the ROC curve

- To compare 2 algorithms over a range of classification thresholds, consider the Area Under the Curve (AUC).
  - A perfect algorithm has AUC=1.
  - A random algorithm has AUC=0.5.
  - Higher AUC doesn't mean all performance measures are better.



K-fold cross-validation

- **Single test-train split**: Estimation test error with high variance.
- **4-fold test-train splits**: Better estimation of the test error, because it is averaged over four different test-train splits.

K-fold cross-validation

- K=1: High variance estimate of Err().

  Fast to compute.

- K>1: Improved estimate of Err(); wastes 1/K of the data.

  K times more expensive to compute.

K-fold cross-validation

- K=1: High variance estimate of Err().

  Fast to compute.

- K>1: Improved estimate of Err(); wastes 1/K of the data.

  K times more expensive to compute.

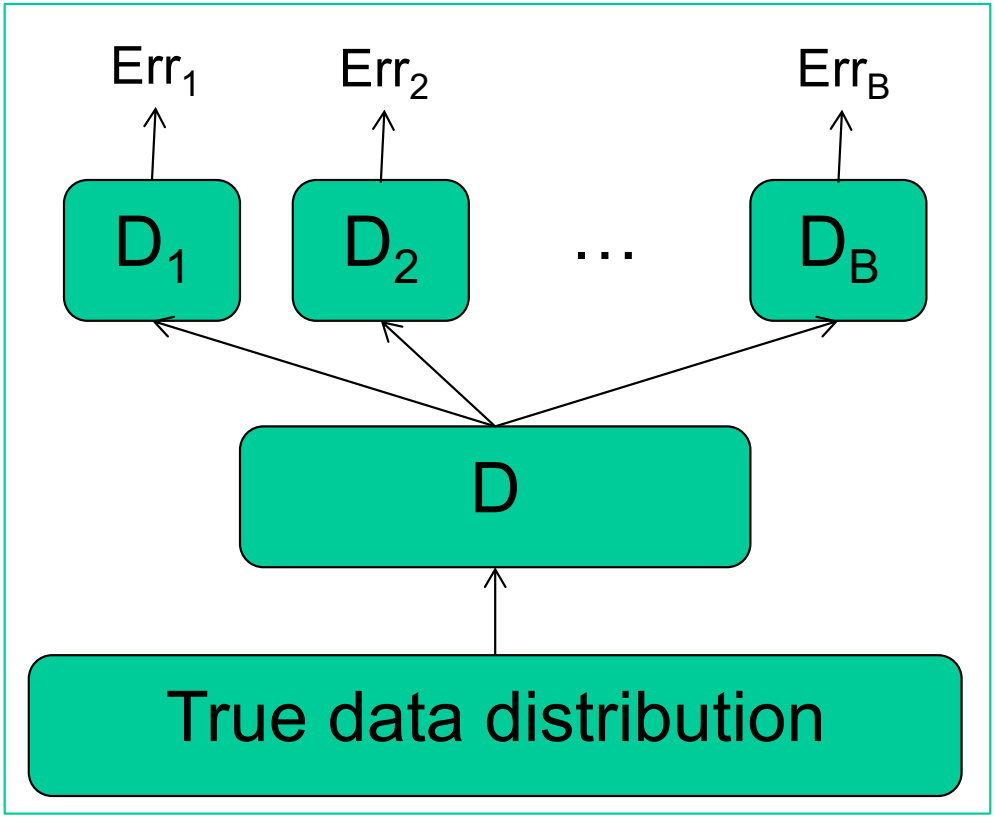
- K=N: Lowest variance estimate of Err(). Doesn't waste data.

  N times slower to compute than single train/validate split.

Brief aside: Bootstrapping

- **Basic idea**: Given a dataset *D* with *N* examples.
  - Randomly draw (with replacement) *B* datasets of size *N* from *D*.
  - Estimate the measure of interest on each of the *B* datasets.
  - Take the mean of the estimates.

Is this a good measure for estimating the error?



Bootstrapping the error

- Use a dataset *b* to fit a hypothesis f^b. Use the original dataset *D* to evaluate the error. Average over all bootstrap sets *b* in *B*.

$$\widehat{\mathrm{Err}}_{\mathrm{boot}} = \frac{1}{B}\frac{1}{N}\sum_{b=1}^{B}\sum_{i=1}^{N} L(y_i, \hat{f}^{*b}(x_i)).$$

- **Problem**: Some of the same samples are used for training the learning and validation.

Bootstrapping the error

- Use a dataset *b* to fit a hypothesis f^b. Use the original dataset *D* to evaluate the error. Average over all bootstrap sets *b* in *B*.

$$\widehat{\mathrm{Err}}_{\mathrm{boot}} = \frac{1}{B}\frac{1}{N}\sum_{b=1}^{B}\sum_{i=1}^{N} L(y_i, \hat{f}^{*b}(x_i)).$$

- **Problem**: Some of the same samples are used for training the learning and validation.

- **Better idea**: Include the error of a data sample *i* only over classifiers trained with those bootstrap sets *b* in which *i* isn't included (denoted C^{-i}).

$$\widehat{\mathrm{Err}}^{(1)} = \frac{1}{N}\sum_{i=1}^{N}\frac{1}{|C^{-i}|}\sum_{b\in C^{-i}} L(y_i, \hat{f}^{*b}(x_i)).$$

(Note: Bootstrapping is a very general ideal, which can be applied for empirically estimating many different quantities.)

Strategy #1

Consider a classification problem with a large number of features, greater than the number of examples (m>>n). Consider the following strategies to avoid over-fitting in such a problem.

Strategy 1:

1. Check for correlation between each feature (individually) and the output. Keep a small set of features showing strong correlation.
2. Divide the examples into k groups at random.
3. Using the features from step 1 and the examples from k-1 groups from step 2, build a classifier.
4. Use this classifier to predict the output for the examples in group k and measure the error.
5. Repeat steps 3-4 for each group to produce the cross-validation estimate of the error.

Strategy #2

Consider a classification problem with a large number of features, greater than the number of examples (m>>n). Consider the following strategies to avoid over-fitting in such a problem.

Strategy 2:

1. Divide the examples into k groups at random.
2. For each group, find a small set of features showing strong correlation with the output.
3. Using the features and examples from k-1 groups from step 1, build a classifier.
4. Use this classifier to predict the output for the examples in group k and measure the error.
5. Repeat 2-4 for each group to produce the cross-validation estimate of the error.

Strategy #3

Consider a classification problem with a large number of features, greater than the number of examples (m>>n). Consider the following strategies to avoid over-fitting in such a problem.

Strategy 3:

1. Randomly sample n' examples.
2. For the sampled data, find a small set of features showing strong correlation with the outptut
3. Using the examples from step 1 and features from step 2, build a classifier.
4. Use this classifier to predict the output for those examples in the dataset that are not in n' and measure the error.
5. Repeat steps 1-4 *k* times to produce the cross-validation estimate of the error.

Summary of 3 strategies

Strategy 1:

1. Check for correlation between each feature (individually) and the output. Keep a small set of features showing strong correlation.
2. Divide the examples into k groups at random.
3. Using the features from step 1 and the examples from k-1 groups from step 2, build a classifier.
4. Use this classifier to predict the output for the examples in group k and measure the error.
5. Repeat steps 3-4 for each group to produce the cross-validation estimate of the error.

Strategy 2:

1. Divide the examples into k groups at random.
2. For each group, find a small set of features showing strong correlation with the output.
3. Using the features and examples from k-1 groups from step 1, build a classifier.
4. Use this classifier to predict the output for the examples in group k and measure the error.
5. Repeat 2-4 for each group to produce the cross-validation estimate of the error.

Strategy 3:

1. Randomly sample n' examples.
2. For the sampled data, find a small set of features showing strong correlation with the ouptut
3. Using the examples from step 1 and features from step 2, build a classifier.
4. Use this classifier to predict the output for those examples in the dataset that are not in n' and measure the error.
5. Repeat steps 1-4 *k* times to produce the cross-validation estimate of the error.

Discussion

- **Strategy 1** is prone to overfitting, because the full dataset is considered in step 1, to select the features. Thus we do not get an unbiased estimate of the generalization error in step 5.

- **Strategy 2** is closest to standard k-fold cross-validation. One can view the joint procedure of selecting the features and building the classifier as the training step, to be applied (separately) on each training fold.

- **Strategy 3** is closer to a **bootstrap** estimate. It can give a good estimate of the generalization error, but the estimate will possibly have higher variance than the one obtained using Strategy 2.

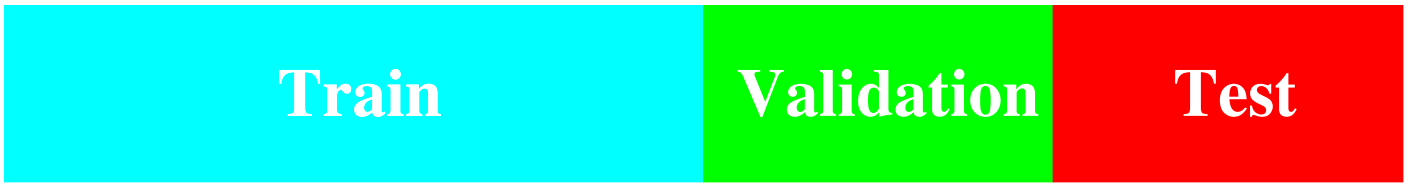
A word of caution

- Intensive use of cross-validation can overfit!
- E.g. Given a dataset with 50 examples and 1000 features.
  - Consider 1000 linear regression models, each built with a single feature.
  - The best of those 1000 will look very good!
  - But it would have looked good even if the output was random!

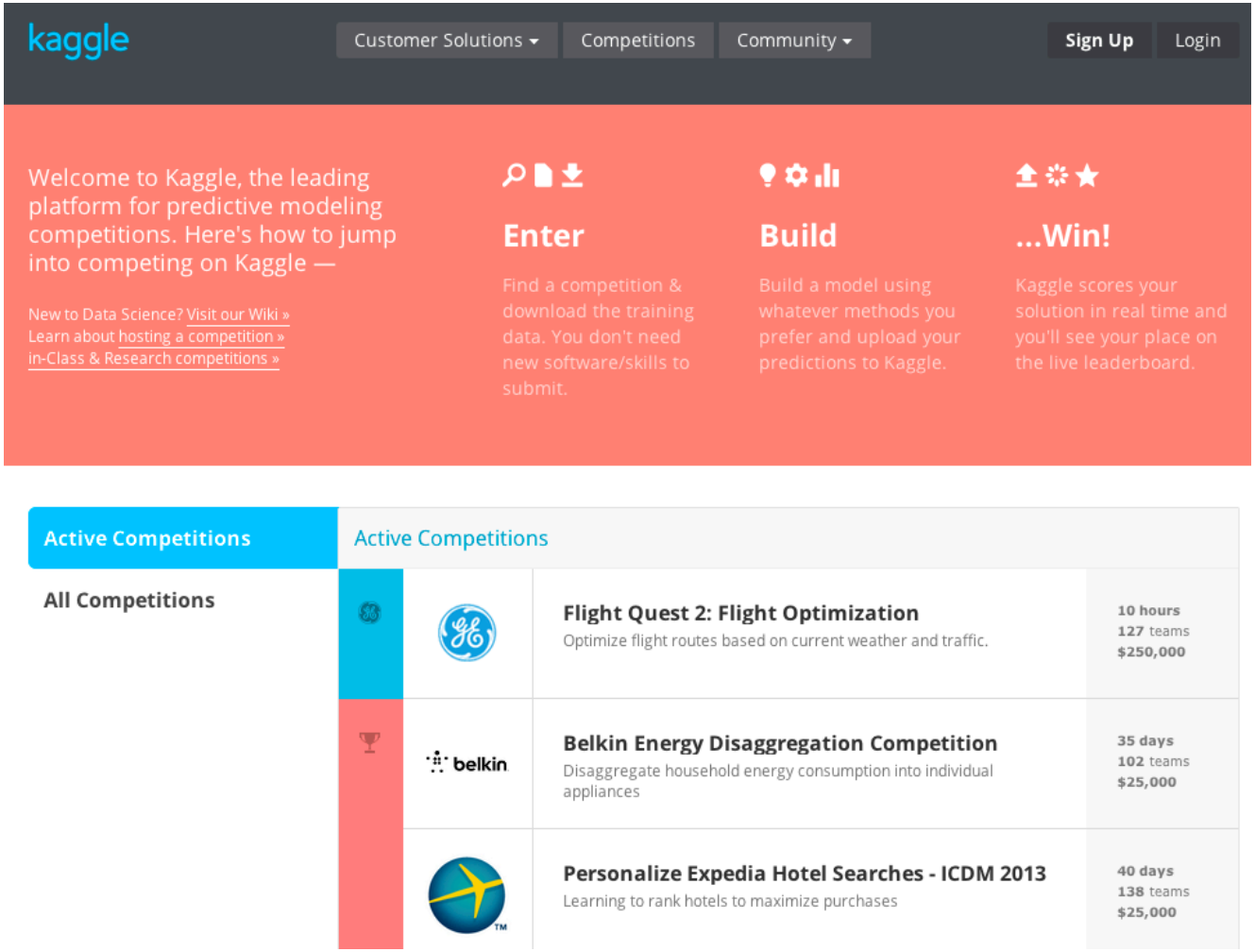
What should we do about this?

To avoid overfitting to the validation set

- When you need to optimize **many** parameters of your model or learning algorithm.
- Use three datasets:
  - The training set is used to estimate the parameters of the model.
  - The validation set is used to estimate the prediction error for the given model.
  - The test set is used to estimate the generalization error once the model is fixed.



Kaggle



http://www.kaggle.com/competitions

Lessons for evaluating ML algorithms

- Always compare to a simple baseline:
  - In classification:
    - Classify all samples as the majority class.
    - Classify with a threshold on a single variable.
  - In regression:
    - Predict the average of the output for all samples.
    - Compare to a simple linear regression.
- Use K-fold cross validation to properly estimate the error. If necessary, use a validation set to estimate hyper-parameters.
- Consider appropriate measures for fully characterizing the performance: Accuracy, Precision, Recall, F1, AUC.

What you should know

- Understand the concepts of loss, error function, bias, variance.
- Commit to correctly applying cross-validation.
- Understand the common measures of performance.
- Know how to produce and read ROC curves.
- Understand the use of bootstrapping.
- Be concerned about good practices for machine learning!

Read this paper today!

K. Wagstaff, “Machine Learning that Matters”, ICML 2012.

http://www.wkiri.com/research/papers/wagstaff-MLmatters-12.pdf