Regression

Machine Learning Practice

Regression

High-level problem definition:

- Supervised learning problem
- In general, inputs can be numerical or categorical data
  - For now, our focus is on numerical inputs
- Outputs are numerical

Regression

Error metrics

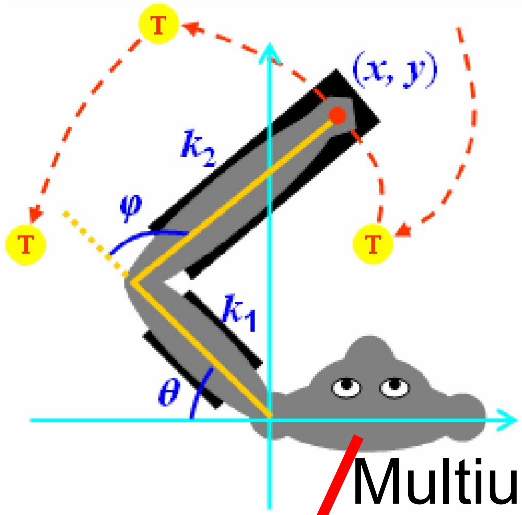
- Generally: a function of the difference between ground truth and predicted values
- Common:
  - Sum squared error (or mean squared error)
  - Sum absolute error (or mean absolute error)

Brain-Machine Interface Problem

CS 5703: Machine Learning Practice

Brain-Machine Interfaces

Command prosthetic arm



Estimate of intended movement

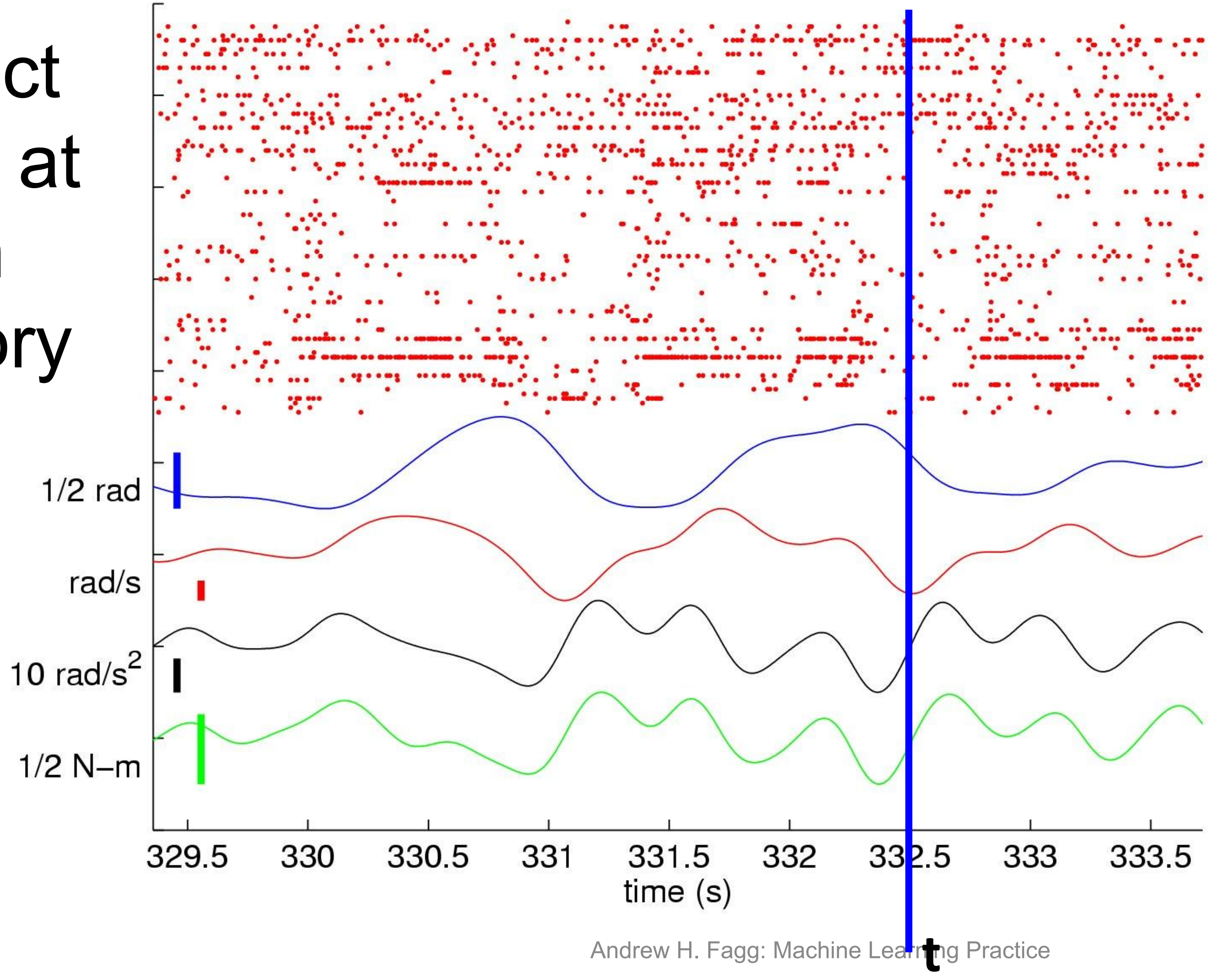
Multiunit recording

Predictive model

In collaboration with Nicholas G. Hatsopoulos and Lee E. Miller

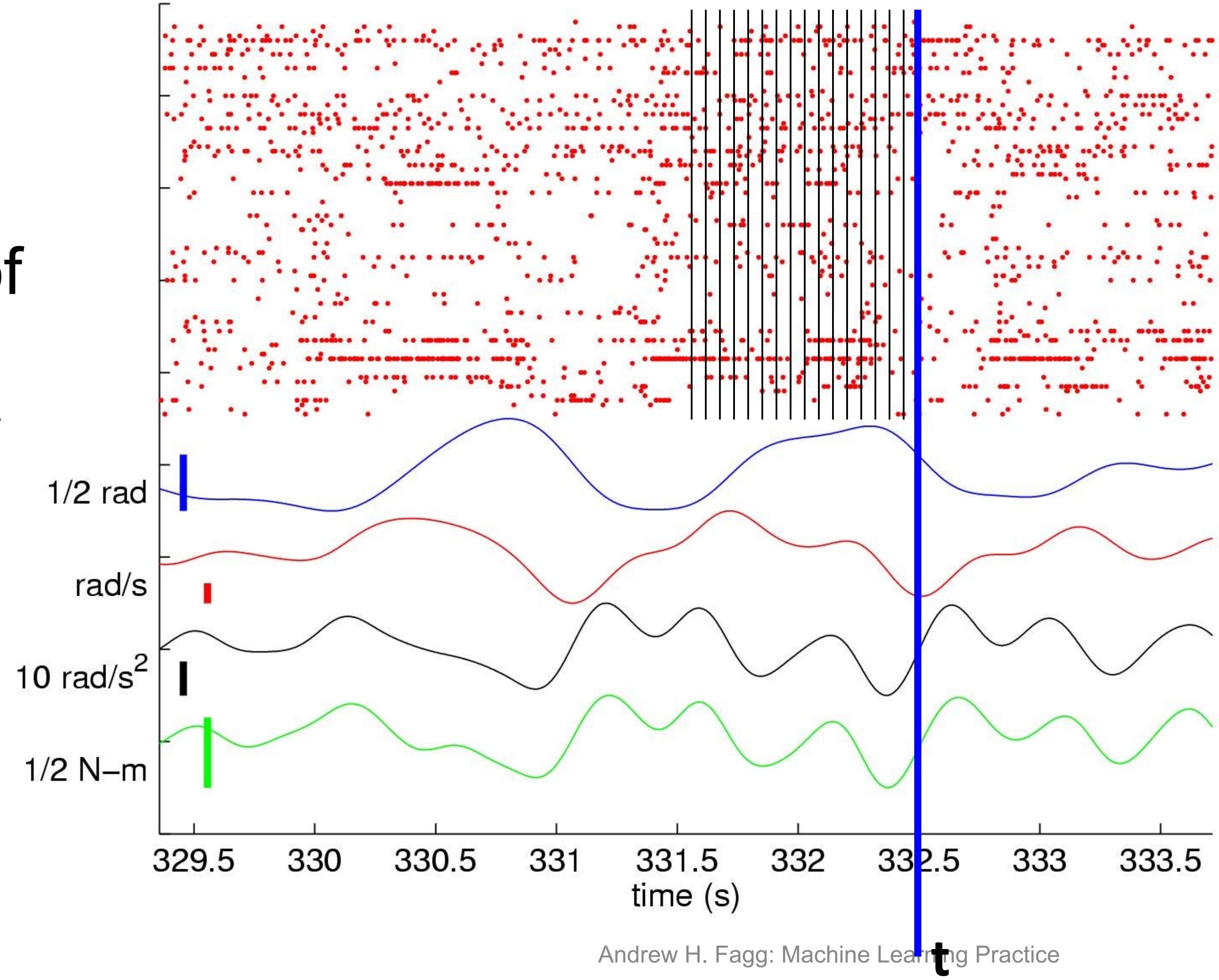
Decoding Arm State

Want to predict arm motion at time t given recent history of spiking behavior



Decoding Arm State

50ms bins: 20 descriptors of neural activation for each cell



BMI Data Configuration

- Data already cut into 20 independent folds
- Time is continuous, but with gaps
  - We kept only valid time periods
- Each sample contains 20 spike bins for each neuron
  - Each count corresponds to 50ms of time
  - A single row is a contiguous set of samples (no gaps!)

Performance Measure:
Fraction of Variance Accounted For (FVAF)

Y = vector of desired outputs

Error = Y - Y'

FVAF = 1 - MSError / VAR(Y)

-inf < FVAF <= 1

- Can interpret similarly to R^2
- Unitless: so can compare for predictors with different units
- Also called "explained variance"

Limits of the Normal Equation

- The “Normal Equation” requires the inversion of an N+1 x N+1 matrix, where N is the number of features
- This can be really expensive as N becomes large
  – And unnecessary if the features are rather sparse

Gradient Descent Methods

Gradient Descent Approach:

- Guess at an initial set of parameters
- Update the parameters in a direction so that the error metric is lowered
- Repeat until error is low enough or stops improving

Gradient Descent Challenges

- It is hard to tell *a priori* how many steps will be necessary
- Unclear what the "learning rate" should be
- Computing the gradient of the error with respect to the parameters:
  - Computation of the gradient is done for each training sample
  - These gradients are then summed together to estimate the global gradient
  - This is ***Batch Gradient Descent***
  - If the training set is large, then this is a computationally expensive process

Estimating the Gradient

- Stochastic Gradient Descent
  - Randomly select a single training example, compute the gradient and update the parameters
- Mini-Batch Gradient Descent
  - Cut the training set into batches
  - Use one batch at a time to compute gradient and update parameters
  - Cycle through these batches
- Stochastic Mini-Batch
  - Each training step: sample M training examples & use these to compute the gradient and update parameters

Example: Training Sensitivity

Machine Learning Practice

Number of Training Steps

How many training steps do we need for a given problem?

- This is an empirical question
- Can visualize using a learning curve
  - Take a small step
  - Record performance on a training set and a validation set
  - Repeat

Training Set Size

With our first regression-based models:

- Performance with the training set was high
- But, performance with an independent data set was generally quite poor
- In our problem, this is due to a dramatic over-fit of the training data
  - Note: 961 parameters and only 1193 samples

Training Set Size

Whenever we face a new problem, it is ***very important*** to ask the question of whether we have enough training data

- One approach: train a model with varying amounts of training data & ask how the model performs on an independent data set
- Sensitive to training set size: you are overfitting and need more data
- Insensitive: you have plenty of training data

Note that this is a model-specific (and hyper-parameter-specific) question

Training Set Size Experiment: Pseudo-Code

```
nfolds = [1,2,3, …]
base_model = LinearRegression()

# Validation data set
ins_valid = ins_folds[valid_fold]
outs_valid = outs_folds[valid_fold]

perf_training = []
perf_validation = []

# Loop over training set sizes
for n in nfolds:
    # Construct a training set
    ins = np.concatenate(ins_folds[:n])
    outs = np.concatenate(outs_folds[:n])

    # Fit the model
    model = clone(base_model)
    model.fit(ins.outs)

    # Evaluate the model
    perf_training.append(evaluate(model, ins, outs))
    perf_validation.append(evaluate(model, ins_valid, outs_valid))
```

Multi-Regression

Machine Learning Practice

Multi-Regression

- So far, our models have only predicted a single output value for a given input
- In practice, we would like to handle entire vectors

Multi-Regression

Multi-regression is a generalization of regression

- Multiple outputs
- For our linear models, the parameters are completely separate from one-another
- Error metric is the sum of errors (or squared errors) across the individual outputs
  - Each output can have its own weight (though, for a pure linear model, there is no difference between weighted and unweighted)

Utility and Limits of Linear Regression

Machine Learning Practice

Linear Regression

Utility:

- Inexpensive to evaluate models
- Can compute the solution to a problem directly (“Normal Equation”)
- Gradient descent approach is straightforward and relatively inexpensive computationally
- There is only one minimum in the error space when using a squared error metric

Linear Regression

Limits:

- The world is rarely linear
- Would like to capture non-linear effects
- Would also like to constrain the output to match our expectations of the valid range of outputs
  - For example, if we are trying to output a probability

Next Steps in Regression

- Non-linear preprocessing of input features
  - Otherwise, the model is linear
- Non-linear on the output of the model
  - Otherwise, the model is linear
  - Logistic regression
- Non-linearities built into the model throughout

Non-Linear Preprocessing

Machine Learning Practice

The Overfitting Problem

Machine Learning Practice

Overfitting

- Any situation where a model performs well on a training set, but not on an independent data set drawn from the same distribution as the training set
- In this case, the learned model has captured the peculiarities of the training set, but not the general trend of the entire distribution of samples
- Detecting this situation is done by comparing model performance on training and independent data

Sources of Overfitting (or Apparent Overfitting)

- Training set is too small relative to the complexity of the model that is being fit
  - One clue: # of samples ~ # of model parameters
- Training set samples are not drawn independently
- Training data not actually drawn from the same distribution as the rest of the data
- Features do not vary systematically with the predicted output

Regularization

Machine Learning Practice

Regularization

Approach: add terms to our cost function that punish models that have large coefficients

Regularization

- LMS: happy with high coefficients
- Ridge: wants to make coefficients small, especially ones that are already large
  - But, is happy to have very small coefficients
- Lasso: wants to make coefficients small
  - Also wants to set as many coefficients as possible to zero
- Elastic Net: also wants to make coefficients small
  - Can walk smoothly between the Ridge and Lasso solutions

Regularization

- Simple regression problem
- Compare Ridge, Lasso and Elastic Net Solutions

Example: Regularization in the BMI Problem

Machine Learning Practice

Regularization in the BMI Problem

- We have already shown that LMS does not perform well with small training data set sizes
- How does regularization help with small training sets?