Practical ML Advice

Rishabh Iyer

Based on slides from Jude Shavlik and Tom Dietterich

Proper Experimental Methodology Can Have a Huge Impact:

A 2002 paper in *Nature* (a major journal) needed to be corrected due to "training on the testing set"

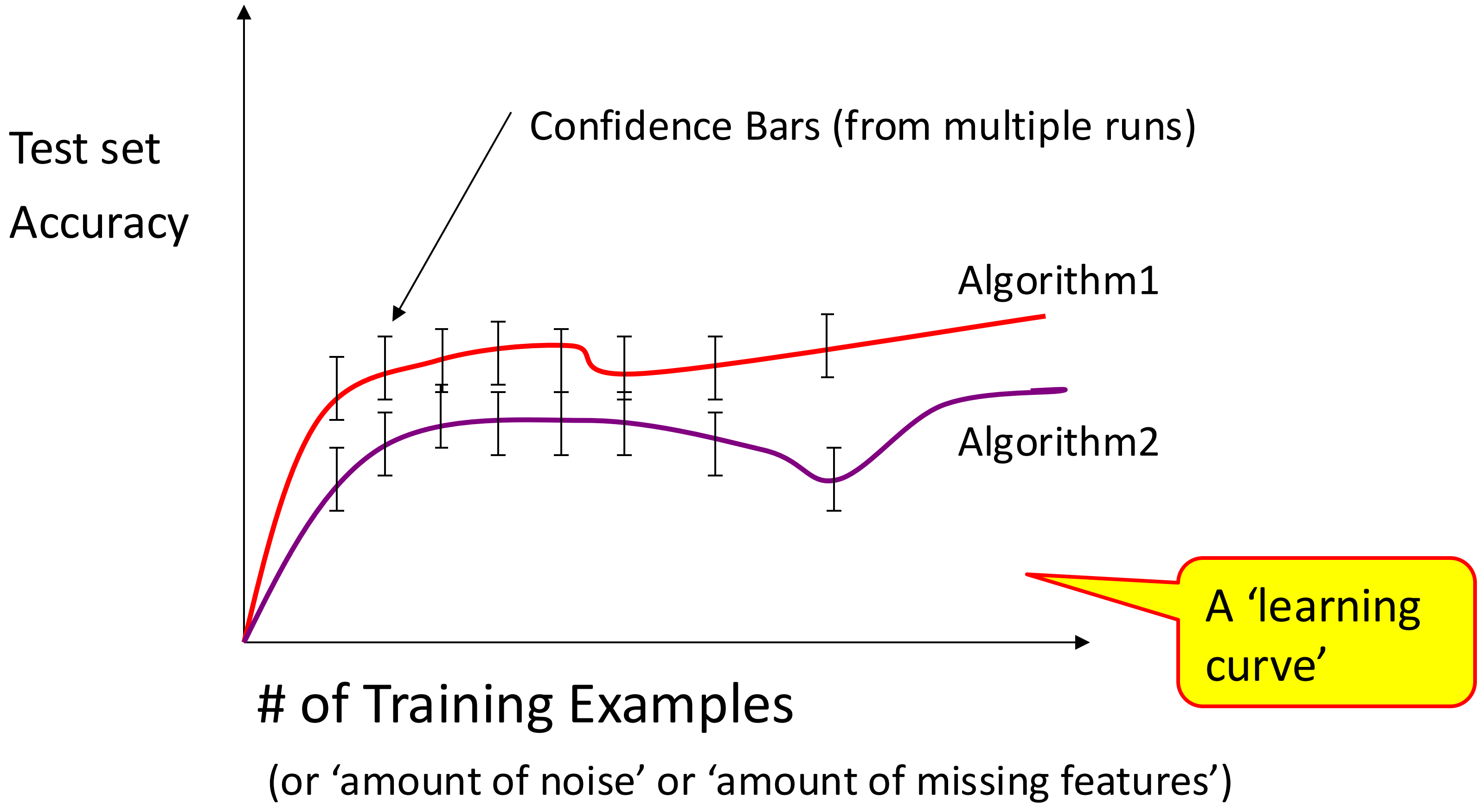
Original report : 95% accuracy (5% error rate)

Corrected report (which still is buggy):

73% accuracy (27% error rate)

Error rate increased over 400%!!!

Some Typical ML Experiments



Typical Experiments

	Test Set Performance
Full System	80%
Without Module A	75%
Without Module B	62%

Experimental Methodology

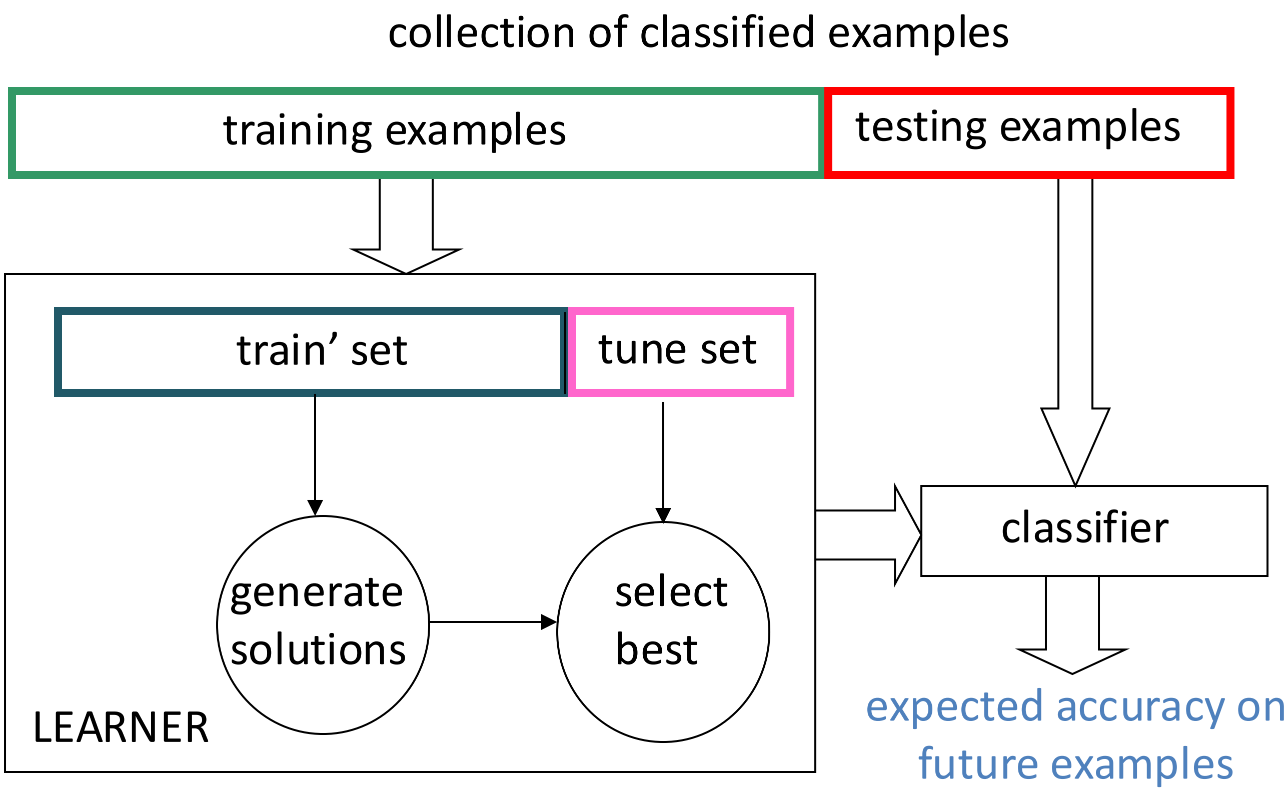
1) Start with a dataset of labeled examples
2) Randomly partition into *N* groups
3a) *N* times, combine *N* -1 groups into a train set
3b) Provide training set to learning system
3c) Measure accuracy on “left out” group (the test set)

Called *N*-fold cross validation

Validation Sets

- Often, an ML system has to choose when to stop learning, select among alternative answers, etc.
- One wants the model that produces the highest accuracy on **future** examples ("overfitting avoidance")
- It is a **"cheat"** to look at the **test** set while still learning
- Better method
  - Set aside part of the training set
  - Measure performance on this validation data to estimate future performance for a given set of hyperparameters
  - Use best hyperparameter settings, train with **all** training data (except **test** set) to estimate future performance on **new** examples

A typical Learning system



Statistical techniques such as 10-fold cross validation and *t*-tests are used to get meaningful results

Multiple Tuning sets

- Using a **single** tuning set can be unreliable predictor, plus some data “wasted”

1) For each possible set of hyperparameters

   a) Divide training data into **train** and **valid.** sets, using ***N*-fold cross validation**

   b) Score this set of hyperparameter values: average **valid.** set accuracy over the *N* folds

2) Use **best** set of hyperparameter settings and **all** (train + valid.) examples

3) Apply resulting model to **test** set

EVALUATING ML MODELS

Regression Evaluation Metrics

1. Mean Absolute Error (MAE)

$$\mathrm{MAE} = \frac{1}{n}\sum_{i=1}^{n}|y_i - \hat{y}_i|$$

- Measures the average magnitude of errors in predictions.
- **Robust to outliers**, all errors are treated equally.

2. Mean Squared Error (MSE)

$$\mathrm{MSE} = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2$$

- Squares the errors before averaging.
- **Penalizes larger errors** more than smaller ones.
- Sensitive to outliers.

Regression Evaluation Metrics

3. Root Mean Squared Error (RMSE)

$$\mathrm{RMSE} = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2}$$

- Square root of MSE.
- **Interpreted in the same units** as the target variable.
- Useful when **larger errors are more significant**.

4. R-squared (R^2 Score)

$$R^2 = 1 - \frac{\sum_{i=1}^{n}(y_i - \hat{y}_i)^2}{\sum_{i=1}^{n}(y_i - \bar{y})^2}$$

- Measures the **proportion of variance explained** by the model.
- Ranges from **−∞ to 1**:
  - $R^2 = 1$: Perfect prediction
  - $R^2 = 0$: Model no better than the mean
  - $R^2 < 0$: Worse than predicting the mean

Contingency Tables

(special case of ‘confusion matrices’)

		True Answer	
		+	-
Algorithm Answer	+	n(1,1) [true pos]	n(1,0) [false pos]
	-	n(0,1) [false neg]	n(0,0) [true neg]

Counts of occurrences

TPR and FPR

True Positive Rate (TPR)

$= n(1,1) \,/\, (\, n(1,1) + n(0,1)\,)$

$=$ correctly categorized +'s / total positives

$\sim$ P(algo outputs + | + is correct)

False Positive Rate (FPR)

$= n(1,0) \,/\, (\, n(1,0) + n(0,0)\,)$

$=$ incorrectly categorized –'s / total neg's

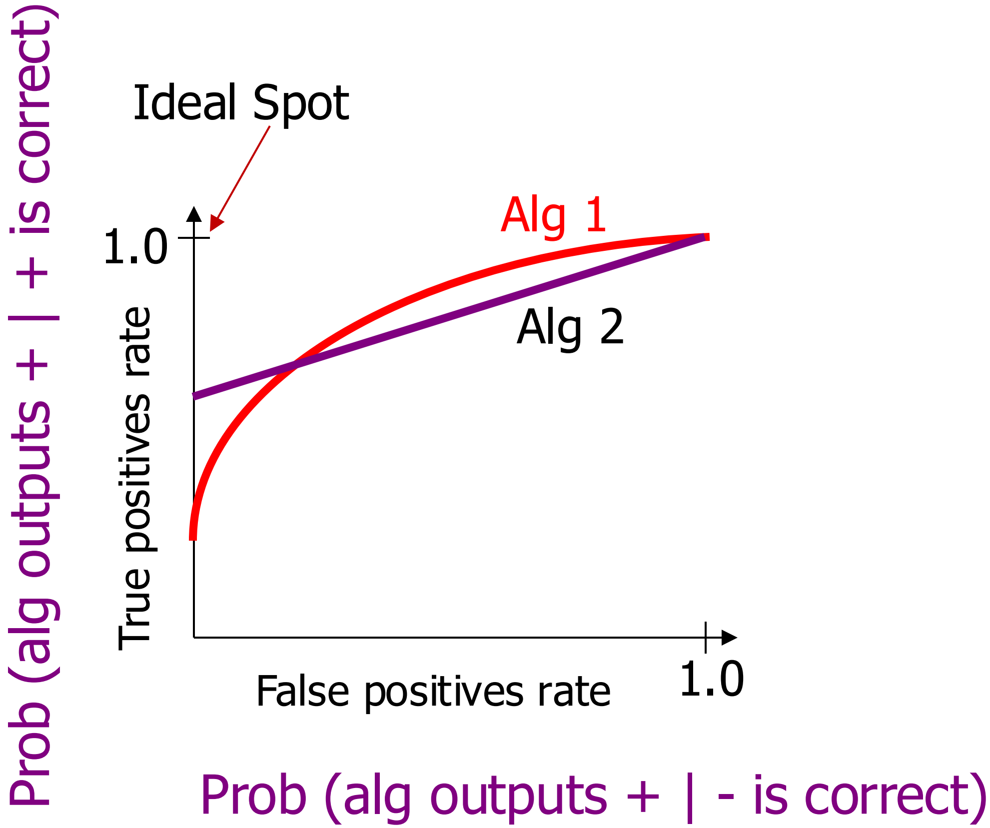
$\sim$ P(algo outputs + | - is correct)

Can similarly define False Negative Rate and True Negative Rate

ROC Curves

- ROC: *Receiver Operating Characteristics*
- Started for radar research during WWII
- Judging algorithms on accuracy alone may not be good enough when **getting a positive wrong** costs more than **getting a negative wrong** (or vice versa)
  - e.g., medical tests for serious diseases
  - e.g., a movie-recommender system

ROC Curves Graphically



Different algorithms can work better in different parts of ROC space. This depends on cost of false + vs false -

Creating an ROC Curve

The Standard Approach:

- You need an ML algorithm that outputs NUMERIC results such as prob(example is +)

- You can use ensemble methods to get this from a model that only provides Boolean outputs
  - e.g., have 100 models vote & count votes

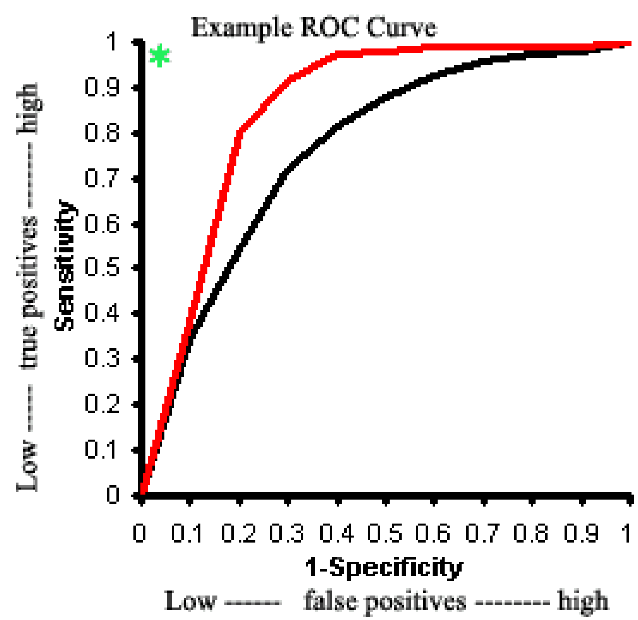
Alg. for Creating ROC Curves

<u>Step 1</u>: Sort predictions on test set

<u>Step 2</u>: Locate a *threshold* between examples with opposite categories

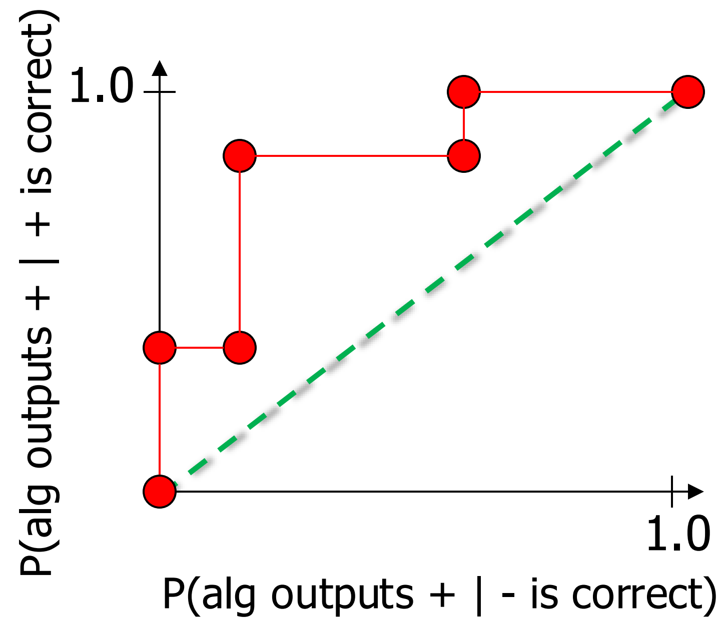
<u>Step 3</u>: Compute TPR & FPR for each threshold of Step 2

<u>Step 4</u>: Connect the dots



Plotting ROC Curves - Example

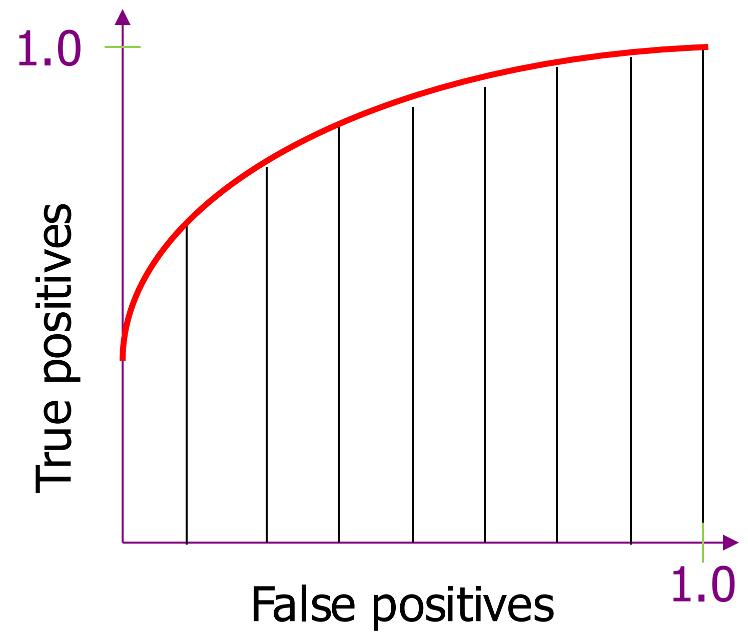
ML Algo Output (Sorted)			Correct Category
Ex 9	.99		+
Ex 7	.98	TPR=(2/5), FPR=(0/5)	+
Ex 1	.72	TPR=(2/5), FPR=(1/5)	-
Ex 2	.70		+
Ex 6	.65	TPR=(4/5), FPR=(1/5)	+
Ex 10	.51		-
Ex 3	.39	TPR=(4/5), FPR=(3/5)	-
Ex 5	.24	TPR=(5/5), FPR=(3/5)	+
Ex 4	.11		-
Ex 8	.01	TPR=(5/5), FPR=(5/5)	-



Algorithm predicts + if its output is ≥ 0

Area Under ROC Curve

- A common metric for experiments is to numerically integrate the ROC Curve
  - Usually called AUC
  - Probability that ML alg. will "rank" a randomly chosen positive instance higher than a randomly chosen negative one
  - Can summarize the curve **too much** in practice



Asymmetric Error Costs

- Assume that cost(FP) ≠ cost(FN)
- You would like to pick a threshold that minimizes

$$E(total\ cost) \\ = \ cost(FP) \times \mathrm{pr}(FP) \times (\#\ of\ neg\ ex's) + \\ cost(FN) \times \mathrm{pr}(FN) \times (\#\ of\ pos\ ex's)$$

- You could also have (maybe negative) costs for TP and TN (assumed zero in above)

ROC's & Skewed Data

- One strength of ROC curves is that they are a good way to deal with skewed data (|+| >> |-|) since the axes are fractions (rates) independent of the # of examples

- You must be careful though!
  - Low FPR * (many negative ex) = sizable number of FP
  - Possibly more than # of TP

Precision vs. Recall

- Think about search engines…

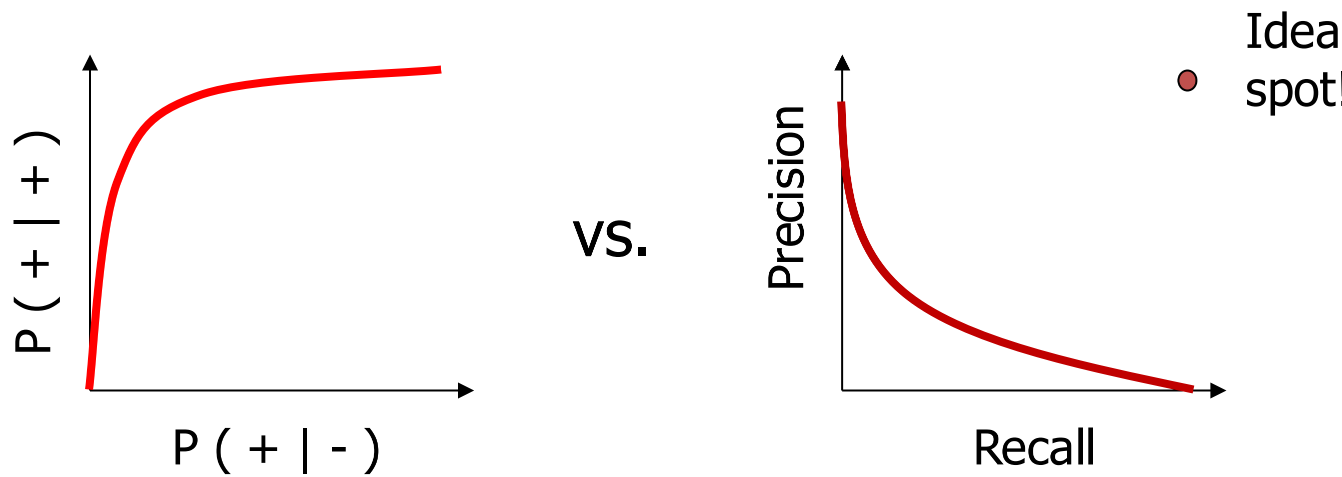
- **Precision** = (# of relevant items retrieved) / (total # of items retrieved)
  $= n(1,1) \,/\, (\, n(1,1) + n(1,0) \,)$
  $\cong P(\text{is pos} \mid \text{called pos})$

- **Recall** = (# of relevant items retrieved) / (# of relevant items that exist)
  $= n(1,1) \,/\, (\, n(1,1) + n(0,1) \,)$ = <u>TPR</u>
  $\cong P(\text{called pos} \mid \text{is pos})$

- Notice that n(0,0) is not used in either formula
  Therefore you get <u>no</u> credit for filtering out <u>ir</u>relevant items

ROC vs. Precision-Recall

You can get very different visual results on the same data!



The reason for this is that there may be lots of – ex's
(e.g., might need to include 100 neg's to get 1 more pos)

The F1 Measure

- Figure of merit that combines precision and recall

$$F_1 = 2 \cdot \frac{P \cdot R}{P + R}$$

where P = precision; R = recall. This is twice the harmonic mean of P and R.

- We can plot $F1$ as a function of the classification threshold θ

Summary of Evaluation Metrics

Metric	Best Use Case
Accuracy	Balanced classes with equal error cost
F1 Score	Imbalanced classes, equal importance to precision/recall
Precision	When **false positives** are costly (e.g., spam filtering)
Recall	When **false negatives** are costly (e.g., medical diagnosis)
AUC	Imbalanced data, ranking-focused applications
MAE (L1)	Regression with **outliers**, stable performance
RMSE (L2)	Regression where **larger errors are worse**
R^2 Score	Overall model fit in regression