Linear Models

Agenda

Discussion Flow

- Linear Model for a continuous response
- Linear Regression with Scikit-Learn
- Model Reduction and Regularisation
- Linear Model (Logistic Regression) for classification
- Logistic Regression with Scikit-Learn

Linear Model for Continuous Response (Linear Regression)

What is a linear model

Response or the quantity being predicted can be written as either directly as linear combination of predictors or function of linear combination of predictors

What is a linear regression

- Response is continuous numeric
- e.g. Sales , Interest Rates etc
- Cost function is generally some variation of difference between predicted and real value

Cost Functions

$$SSE = Sum\ of\ square\ of\ errors = \sum_{i=1}^{n} (y_i - \beta_0 - \beta_1 * x_{1i} - \beta_1 * x_{1i} \ldots.)^2$$

$$SAE = Sum\ of\ square\ of\ errors = \sum_{i=1}^{n} |y_i - \beta_0 - \beta_1 * x_{1i} - \beta_1 * x_{1i} \ldots.|$$

Note : SSE is more popular due to ease of gradient calculation

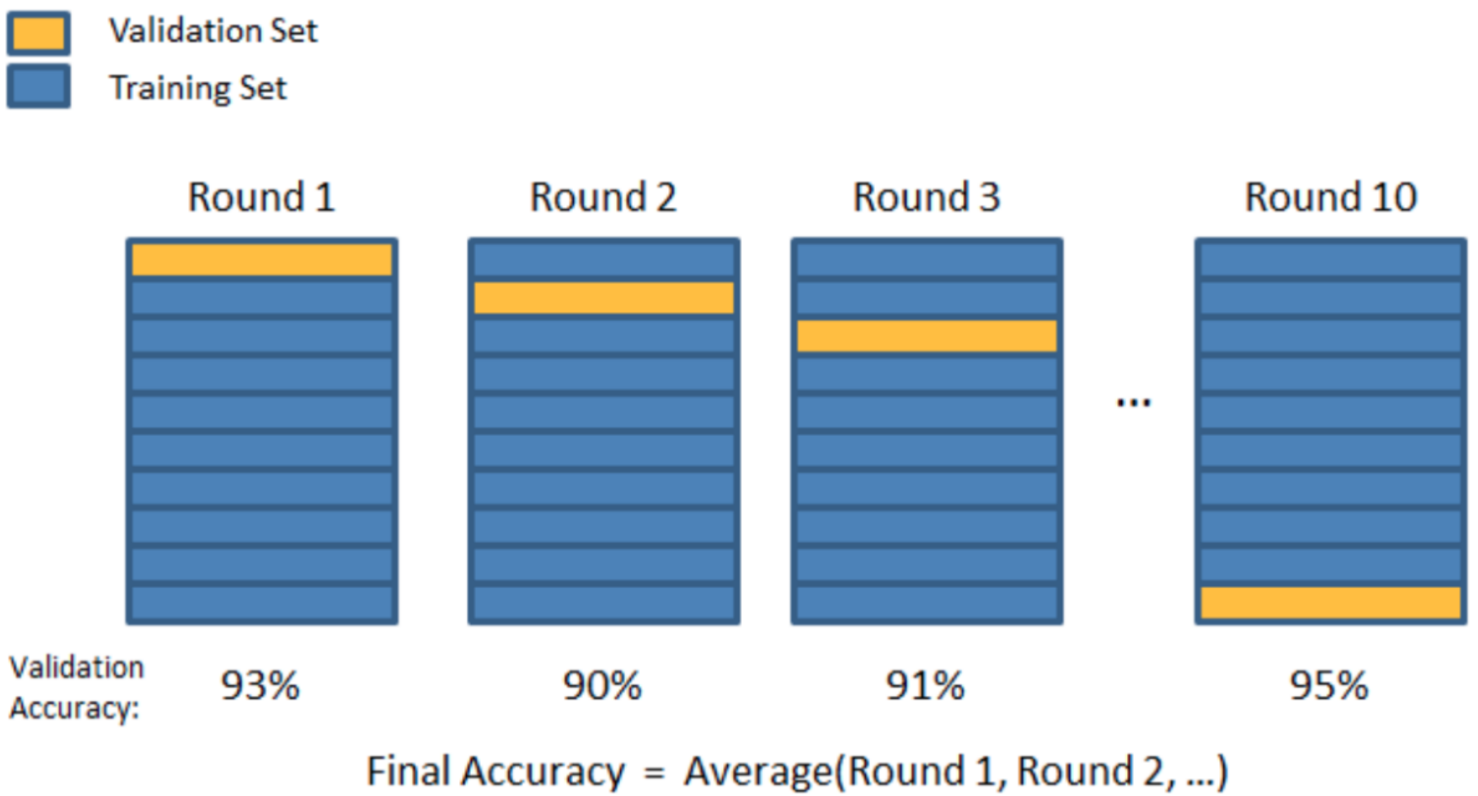
Preparing Data Before Starting with Model Building

- All variables need to be numeric
- Categorical Variables will converted to either numbers or dummy vars
- Eventually the columns/data which is being used for train, should be same for test/production data
- Data should not contain any missing values
- Package Used : Scikit-Learn

Performance Measure

- We can either separate our data in two parts or can use cross validation for performance measure
- Simple average of errors will not make sense as +ve and -ve errors will cancel each other out
- RMSE : Root Mean Square Error
- MAE : Mean Absolute Error
- These measures will be at the scale of response , so there is no universal good range of these performance measures.

Cross Validation



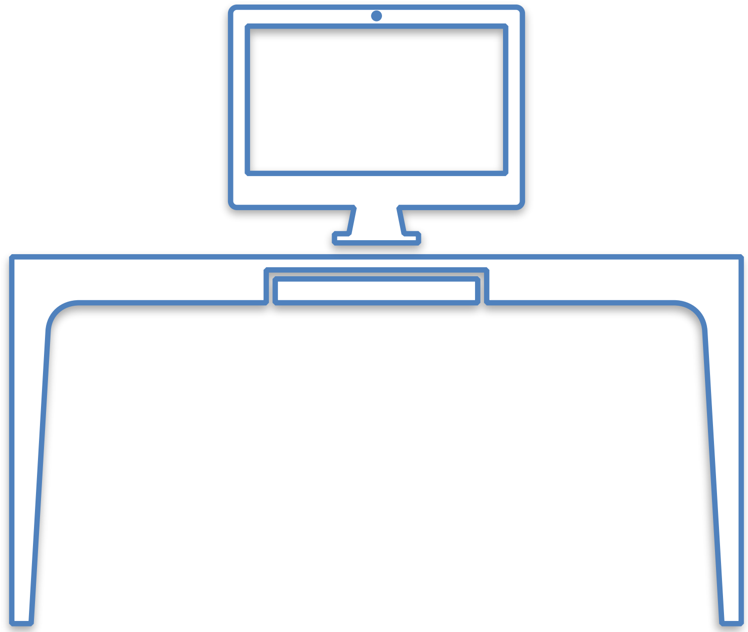
Interpretation of the model

$$y_i = \beta_0 + \beta_1 * x_{1i} \ldots$$

β_0 → Value of response when all other predictors effects are absent/zero

β_j → If j^{th} variable is changed by 1 unit , then response changes by these many units

Lets see it in action in Python



Discarding/Suppressing Effect of Bad Vars

- All the variables in the data might not constructively contribute towards explaining your response
- Ideally a non related variable should have their coefficient as exact zero (which doesnt happen)
- We need a mathematical idea to suppress coefficient of such variables to as close to zero as possible
- Method : Regularisation (General name for methods to reduce overfit)

Regularisation (in Linear Models)

- In linear models , regularisation is achieved by adding penalty for parameter size to the cost function
- Two Popular Regularisation Penalties : Ridge / Lasso

Ridge

$$(Y - X\beta)^2 \; + \; \lambda \sum_{i=1}^{p} \beta_i^2$$

Lasso

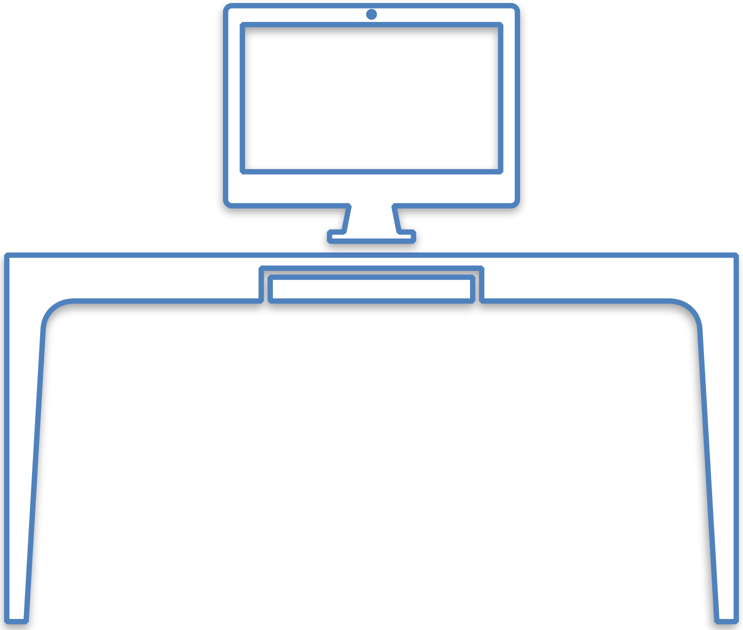
$$(Y - X\beta)^2 \; + \; \lambda \sum_{i=1}^{p} |\beta_i|$$

Note : lamda here is a hyper parameter and represents extent of penalty

Difference between Ridge and Lasso

- Ridge has a closed form solution
- Cost function with lasso penalty is not directly differentiable and hence doesnt have a closed form solution
- Ridge Doesn’t result in model reduction as it can not make coefficient exactly zero
- Lasso can make coefficient exactly zero

Lets see it in action in Python



Linear Model for Classification (Logistic Regression)

What are we trying to predict?

- When the outcome are classes / not numbers
- e.g. Response to a campaign (Yes/No), Products Defects (Good/Bad)
- Although we can convert them to 0/1, but it doesn't make sense to try and predict just 0 or 1
- e.g. predicting whether somebody might have a heart disease given their age. Predicting “Yes” for both Age=45 and Age=85 is not informative enough
- Probability of outcome being “Yes” is much more informative and lets us differentiate between different predictor values and their effect
- That implies , we will be trying to predict $P(Y=1|X)$

Model Equation

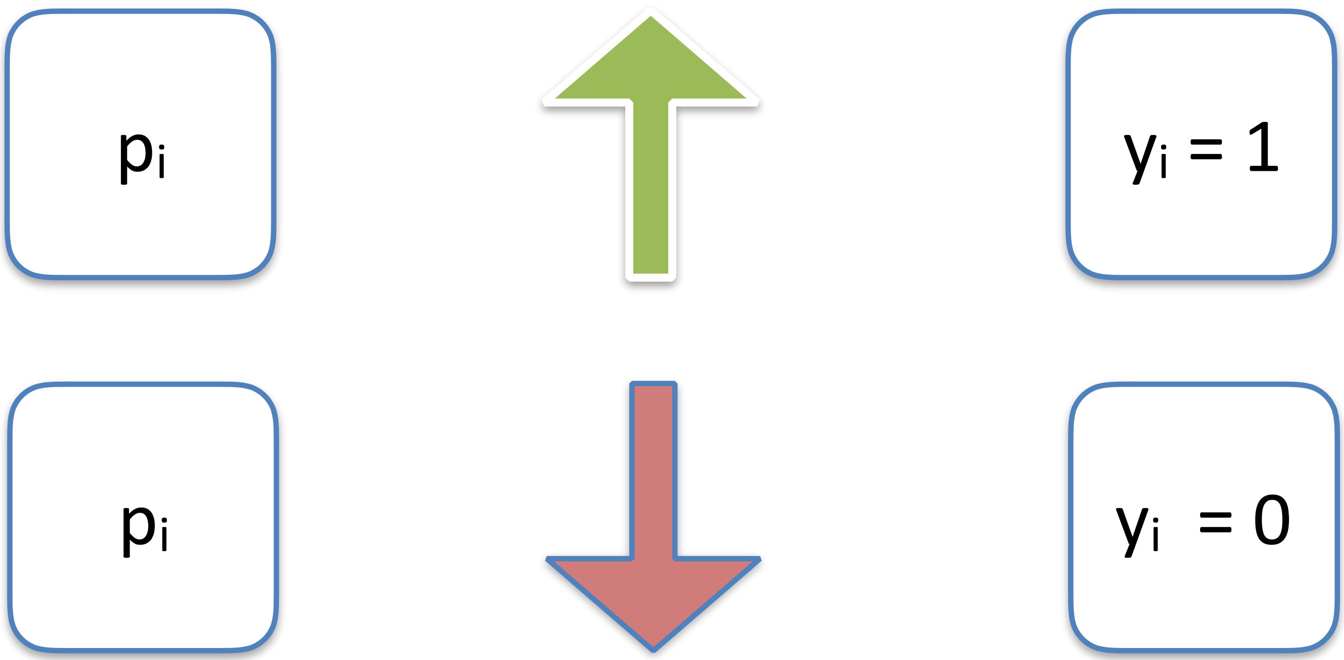
$$P(y_i = 1|X_i) => p_i$$

$$log(\frac{p_i}{1 - p_i}) = \beta_0 + \beta_1 * x_{1i} + \beta_2 * x_{2i} \ldots.$$

$$p_i = \frac{1}{1 + e^{-(\beta_0 + \beta_1 * x_{1i} + \beta_2 * x_{2i} \ldots.)}}$$

Our Expectations From the Predicted Probabilities

$$P(y_i = 1 | X_i) => p_i$$



Note : We essentially want our predicted probabilities to be as aligned with the real outcome as possible

Cost Function

Likelihood for one Observation

$$p_i^{y_i} * (1 - p_i)^{(1-y_i)}$$

Likelihood for all observation

$$\prod_{i=1}^{n} p_i^{y_i} * (1 - p_i)^{(1-y_i)}$$

Cost Function

$$-\sum_{i=1}^{n} y_i * log(p_i) + (1 - y_i) * log(1 - p_i)$$

From Probability Score to Hard Classes

$$p_i > cutoff => y_i = 1$$

$$p_i <= cutoff => y_i = 0$$

Confusion Matrix and Performance Measures

Predicted

Real

	Positive	*Negative*
Positive	***TP***	***FN***
Negative	***FP***	***TN***

Contd..

$$P = TP + FN \qquad N = TN + FP$$

$$Total = P + N$$

$$Sensitivity = \frac{TP}{P}$$

$$Specificity = \frac{TN}{N}$$

$$Accuracy = \frac{TP + TN}{Total}$$

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

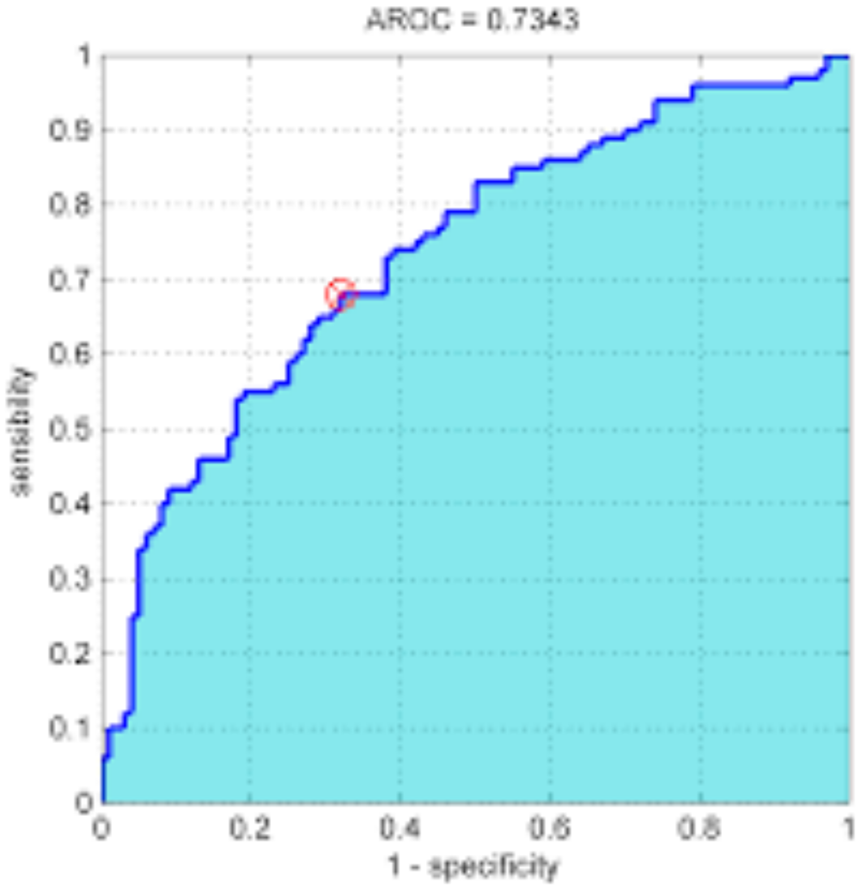
$$Recall = \frac{TP}{P}$$

Finding Cutoff

$$KS = \frac{TP}{P} - \frac{FP}{N}$$

$$F_{\beta} = \frac{(1+\beta^2) * Precision * Recall}{(\beta^2 * Precision) + Recall}$$

Performance Measure for Probability Score : AUC Score



Interpretation of the model

$$\frac{p_i}{1-p_i} = e^{\beta_0} * e^{(\beta_1 * x_{1i})} \ldots$$

β_j

If j^{th} variable is changed by 1 unit , then odds in favour of y=1 change by a factor of

$$e^{\beta_j}$$

Lets see it in action in Python