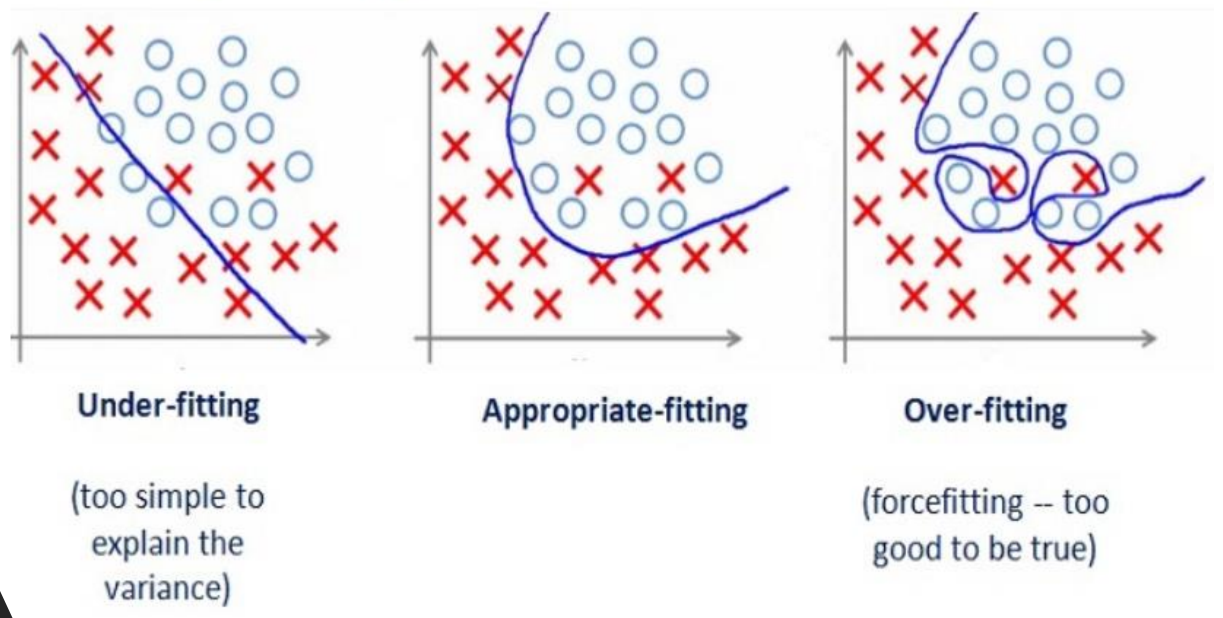
INTRODUCTION TO MACHINE LEARNING

DATA 602
Lecture 5

Dr. Tony Diana
tonydian@umbc.edu

The Rationale for Regularization


Under-fitting
(too simple to explain the variance)
Appropriate-fitting
Over-fitting
(forcefitting -- too good to be true)

UMBC

The Rationale for Regularization

- One of the key challenges that machine learning is trying to address is **overfitting**
- This happens when the model cannot learn from new data because it is trying to integrate noise in the training dataset
- One of the best way to avoid **overfitting** is to use cross validation, which is designed to estimate the error test set and identify what parameters work best for the model
- **Regularization** is a method used in regression, but also with neural networks and some other models, to constrain or shrink the estimates to zero
- The fitting procedure involves a loss function, known as residual sum of squares or RSS
- Coefficients are selected so that they minimize this loss function (**Residual Sum of Squares**):

$$\mathrm{RSS} = \sum_{i=1}^{n} \left(y_i - \beta_0 - \sum_{j=1}^{p} \beta_j x_{ij} \right)^2$$

- The **RSS** will adjust the coefficients based on the training data
- If there is noise in the training data, then the estimated coefficients will not generalize well to future data
- **Regularization** either shrinks or drives the estimates toward zero

UMBC

Regularized and Penalized Regression

- **Constraining a model** to make it **simpler** and **reduce the risk of overfitting** is called **regularization**
- We need to find the balance between fitting the training data perfectly and keeping the model simple enough to ensure that it will generalize well
- **Regularization** is a way to **reduce overfitting** by **constraining** it
  - The fewer the degrees of freedom, the harder it will be to overfit the data
- From a mathematical viewpoint, a regularizer is a penalty added to the cost function, to impose an extra condition on the evolution of the parameters:

$$L_R(X, Y; \bar{\theta}) = L(X, Y; \bar{\theta}) + \lambda g(\bar{\theta})$$

with λ controlling the strength of the regularization, which is expressed through the function $g(\bar{\theta})$

- Regularization either **constrains** or **shrinks** the coefficient estimates toward zero
- Regularization **simplifies a complex model** to avoid the risk of overfitting
  - If there is noise in the training data, then the estimated coefficients will not generalize well
  - The regularization term should be added to the cost function during training
- It is important to follow Occam's principle: **Keep things simple!** Or **"**Entia non sunt multiplicanda praeter necessitatem**"**

UMBC

The Purpose of L1 and L2 Regularization

- The fewer degrees of freedom a model has, the harder it will be for it to overfit the data
- L2 is the most common type of regularization, followed by L1
- These regularization methods work by **adding a term to the loss of the model** to get the final cost function
- The added term leads to a **decrease in the weights of the model**, which leads to **improved generalization**

L1 or Tikhonov Regularization (LASSO or Least Absolute Shrinkage and Selection Operator)

- **Objective = RSS + λ* (sum of absolute value of coefficients)**
- LASSO regression tends to eliminate the weights of the least important features (sets them to zero)
- The cost function of L1 regularization is

$$\sum_{i=1}^{n}(Y_i - \sum_{j=1}^{p} X_{ij}\beta_j)^2 + \lambda \sum_{j=1}^{p} |\beta_j|$$

- **LASSO** automatically performs feature selection and outputs a sparse model (a model with few non-zero feature weights)
- **LASSO Regression** adds "***absolute value of magnitude***" of coefficient as penalty term to the loss function
- λ is the regularization parameter. Large values of λ will drive coefficients to zero
- L1 regularization leads to weights that are very close to zero
- L1 regularization allows to focus **only on the most important inputs**, while ignoring the noisy inputs
- If λ = 0, we get the OLS

L1 and L2 Regularization (Cont.)

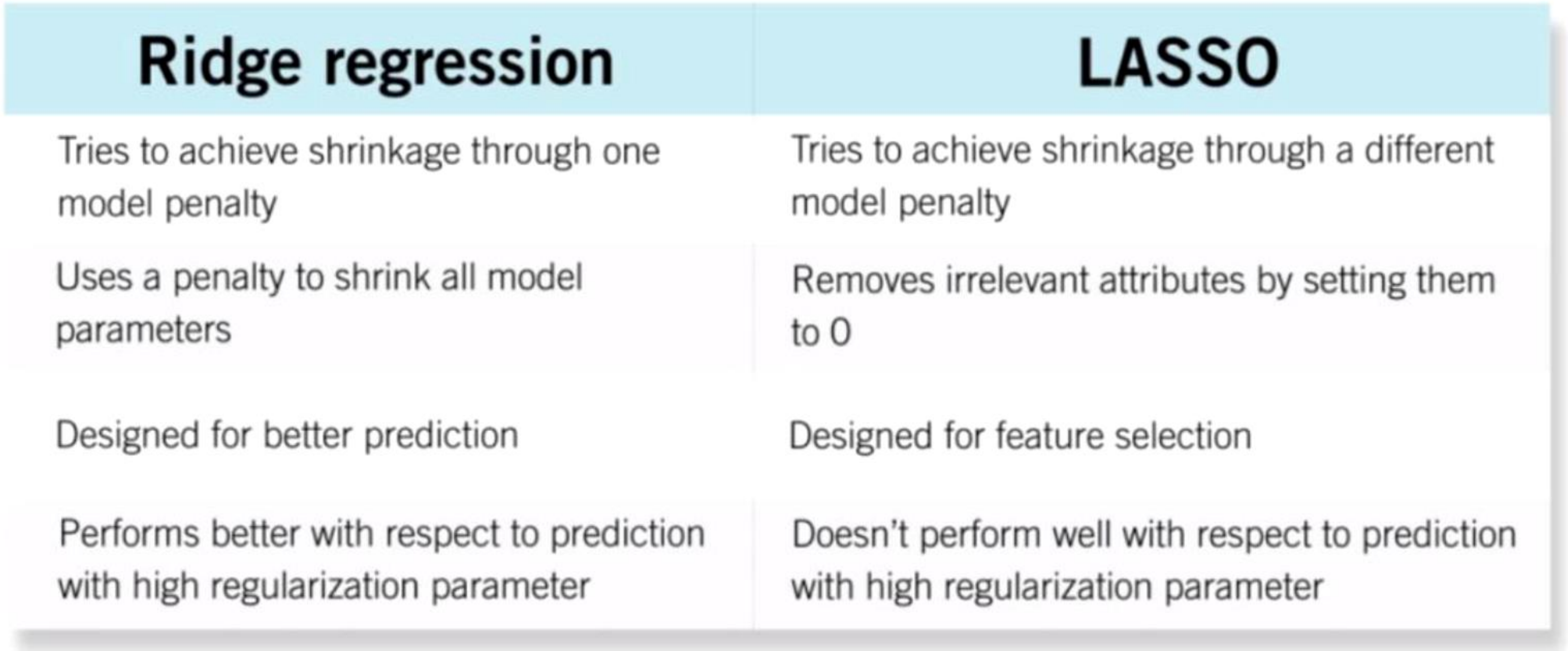
L2 Regularization (Ridge)

- **Objective = RSS + λ * (sum of square of coefficients)**
- The cost function of L2 regularization is

$$\sum_{i=1}^{n}(y_i - \sum_{j=1}^{p} x_{ij}\beta_j)^2 + \lambda \sum_{j=1}^{p} \beta_j^2$$

- Ridge regression adds "**squared magnitude**" of coefficient as penalty term to the loss function
- L2 regularization heavily penalizes high-weight vectors
- L2 regularization is also known as **weight decay** because it forces the weights of model features to decay towards zero but, unlike L1 regularization, **not exactly** to zero
- L2 regularization works well to avoid overfitting
- If λ = 0, there is no regularization and the objective becomes the same as the linear regression model
- If λ = ∞, then the coefficients will tend to zero and the model will be a constant function
- Note that the regularization term should only be added to the cost function during training
- Always scale the data using 'StandardScaler' before performing Ridge Regression because it is sensitive to the scale of the input features. This is also true for other regularized models
- We can combine L1 and L2 and implement them together in the **Elastic Net** regularization model

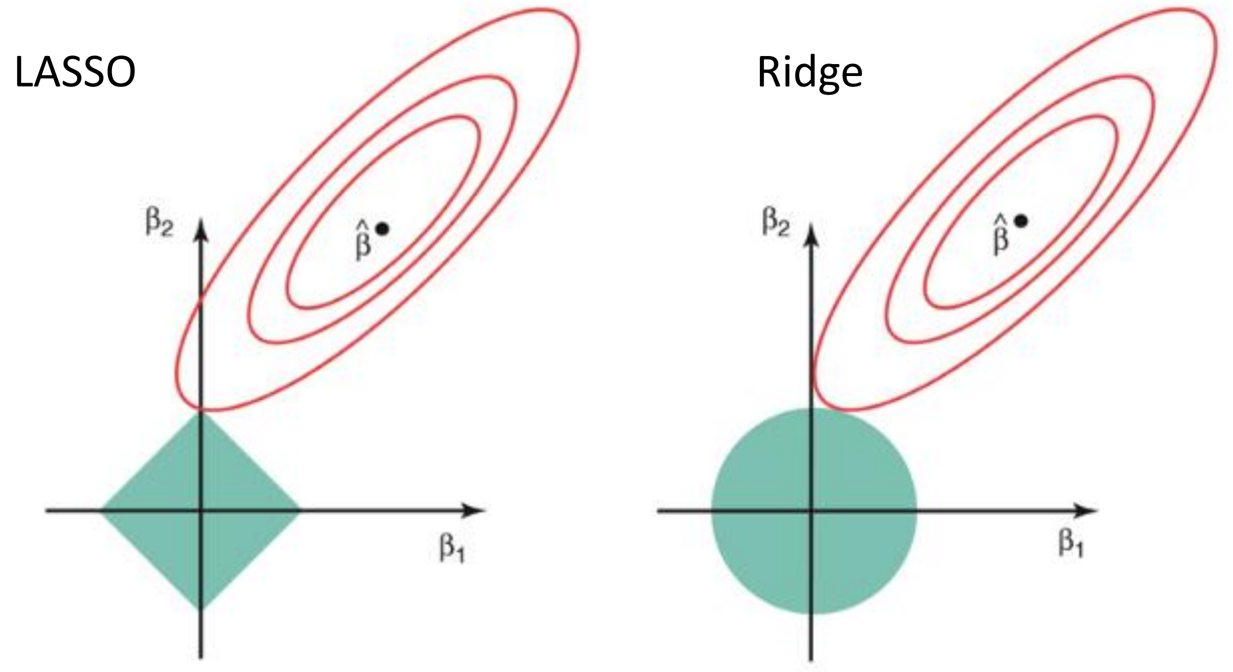
The Differences between Ridge and LASSO Models



Ridge regression	LASSO
Tries to achieve shrinkage through one model penalty	Tries to achieve shrinkage through a different model penalty
Uses a penalty to shrink all model parameters	Removes irrelevant attributes by setting them to 0
Designed for better prediction	Designed for feature selection
Performs better with respect to prediction with high regularization parameter	Doesn't perform well with respect to prediction with high regularization parameter

Source: MIT

The Differences between Ridge and LASSO Models: Pictorial Representation



- The LASSO and Ridge Regression coefficient estimates are given by the first point at which an ellipse contacts the constraint region in blue
- Since Ridge Regression has a circular constraint with no sharp points, this intersection will not generally occur on an axis. Therefore, the Ridge Regression coefficient estimates will be exclusively non-zero
- The LASSO constraint has corners at each of the axes. Therefore, the ellipse will often intersect the constraint region at an axis. When this occurs, one of the coefficients will equal zero

How to Use Ridge and LASSO Regression

Steps to Utilize the Two Methods Well:

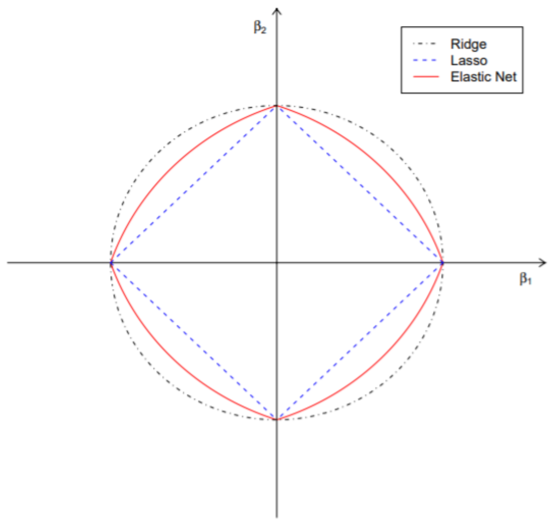
- Run LASSO Regression on a small amount of data for **feature selection**
- Run Ridge Regression on the small set of features selected by LASSO Regression

Typical Use Cases

- **Ridge Regression** is used to **prevent overfitting**. Since it includes all the features, it is not very useful in case of many features
- **LASSO Regression shrinks the coefficients of the least important features to zero**
    - Since it provides **sparse solutions**, it is generally the model of choice for **modelling cases** where there are many features
    - Getting a sparse solution is of great computational advantage as **the features with zero coefficients can simply be ignored**

Regularized Models: Elastic Net

2-dimensional illustration $\alpha = 0.5$



Elastic Net Regression

- **The model** is $$\hat{\beta} \equiv \underset{\beta}{\operatorname{argmin}}(\|y - X\beta\|^2 + \lambda_2\|\beta\|^2 + \lambda_1\|\beta\|_1).$$
- The L1 part of the penalty generates a sparse model
- The quadratic part of the penalty
  - Removes the limitation on the number of selected variables
  - Stabilizes the L1 regularization path
- **Elastic Net** first emerged as a result of critiques about **LASSO**, whose variable selection can be too dependent on data and thus unstable
- The solution is to combine the penalties of **Ridge Regression** and **LASSO** to get the best of both worlds
- **Elastic Net** produces a regression model that is penalized with both the **L1-norm** and **L2-norm**
- The consequence of the combination is to effectively shrink coefficients (like in **Ridge Regression**) and to set some coefficients to zero (as in **LASSO**)
- The challenge consists in finding the two λ's
- When λ = 0, Elastic Net is equivalent to **Ridge Regression**
- When λ = 1, it is equivalent to **LASSO Regression**

Where Else Do You Use Regularization?

Neural Networks

NN are complex models that can include many layers and parameters that can increase overfitting

- **L2** can be used to constrain a NN's connection weights
  - The l2() function returns a regularizer that will be called at each step during training to compute the regularization loss. It is then added to the final loss
- **L1** can help derive a sparse model (with many weights equal to zero)
  - You can use You can use 'keras.regularizers.l1()'. If you want both L1 and L2 then use keras.regularizers.l1_l2()
- **Early Stopping** is a different way of regularizing iterative learning algorithms such as **Gradient Descent** to stop training as soon as the validation error reaches a minimum
  - After a while, the validation error starts decreasing and starts to go back up again
  - This is an indication that the model has started to overfit the training data
  - **Early stopping** allows to stop training as soon as the validation error reaches the minimum. It is an efficient regularization technique

UMBC

Where Else Do You Use Regularization?

Neural Networks (Cont.)

- **Dropout** is one of the most popular technique for NN (Hinton, 2012)
  - You can get an accuracy boost from 1 to 2% percent by specifying the probability of dropping input neurons during the training step (the p hyperparameter)
- **Max-Norm Regularization** does not add a regularization loss term to the overall loss function. It simply rescales weights at each training step

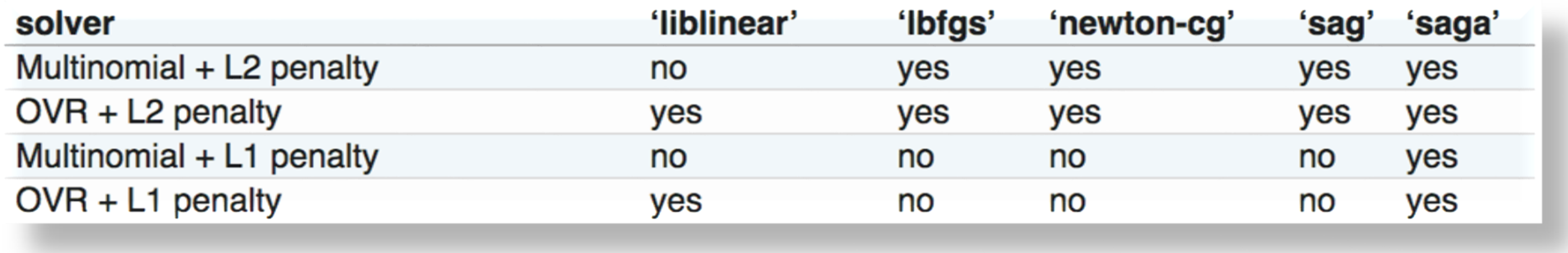
Support Vector Machine (SVM)

- The regularization parameter (lambda) serves as a degree of importance that is given to miss-classifications
- **SVM** looks for maximizing the margin between classes and minimizing the amount of miss-classifications
- For non-separable problems, in order to find a solution, the miss-classification constraint must be relaxed, and this is done by setting the mentioned "regularization"
- When lambda tends to 0 (without being 0) the more the miss-classifications are allowed

Where Else Do You Use Regularization?

Logistic Regression

- You can use **L1**, **L2,** and **Elastic Net** penalization with **Logistic Regression**
- The 'newton-cg', 'sag' and 'lbfgs' solvers support only **L2** penalties
- 'elasticnet' is only supported by the 'saga' solver
- If 'none' (not supported by the liblinear solver), no regularization is applied
- The solvers are explained in the next page as a reference



solver	'liblinear'	'lbfgs'	'newton-cg'	'sag'	'saga'
Multinomial + L2 penalty	no	yes	yes	yes	yes
OVR + L2 penalty	yes	yes	yes	yes	yes
Multinomial + L1 penalty	no	no	no	no	yes
OVR + L1 penalty	yes	no	no	no	yes

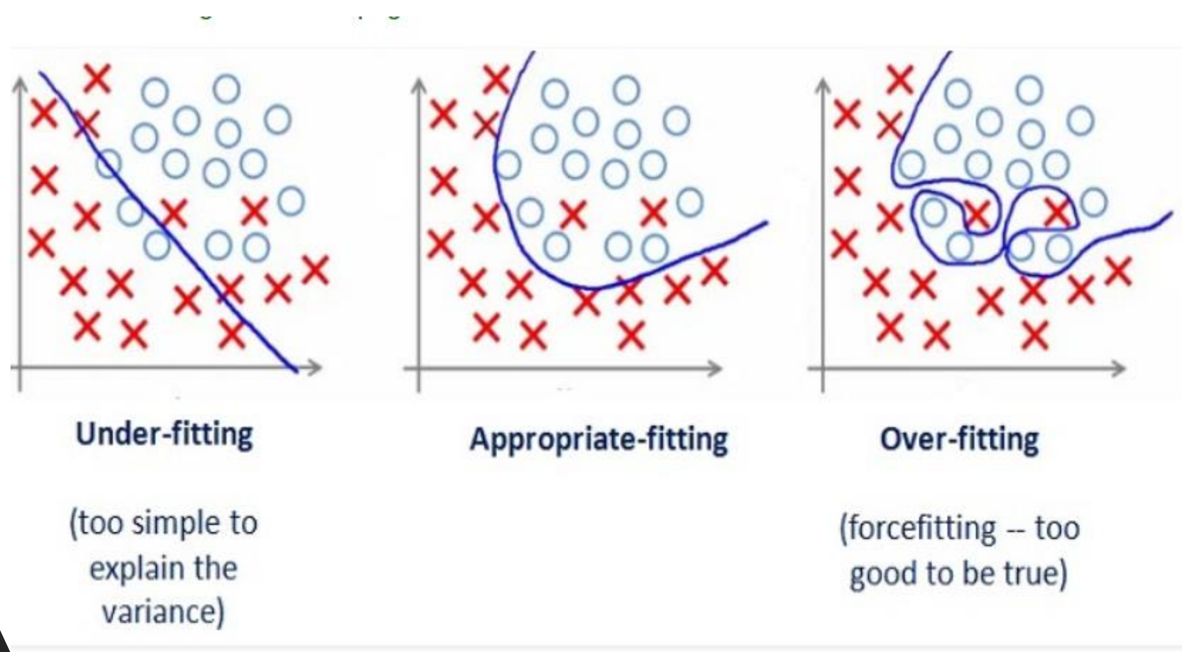
Logistic Regression (Cont.)

newton-cg — A **Newton method**. Newton methods use an exact Hessian matrix. It's slow for large datasets, because it computes the second derivatives
lbfgs — Stands for **Limited-memory Broyden–Fletcher–Goldfarb–Shanno**. It approximates the second derivative matrix updates with gradient evaluations. It stores only the last few updates, so it saves memory. It is not fast with large data sets. It is the default solver as of Scikit-learn version 0.22.0
liblinear — **Library for Large Linear Classification**. Uses a coordinate descent algorithm. Coordinate descent is based on minimizing a multivariate function by solving univariate optimization problems in a loop. In other words, it moves toward the minimum in one direction at a time. It is the default solver for Scikit-learn versions earlier than 0.22.0. It performs well with high dimensionality. It does have a number of drawbacks. It can get stuck, is unable to run in parallel, and can only solve multi-class logistic regression with one-vs.-rest
sag — **Stochastic Average Gradient Descent**. A variation of gradient descent and incremental aggregated gradient approaches that uses a random sample of previous gradient values. Fast for big datasets.
saga — **Extension of *sag* that also allows for L1 regularization**. Should generally train faster than *sag*

Ensemble Learning

- Some sequential algorithms such as XGBoost have regularization methods to prevent overfitting
- Regularization can be implemented through the ‘learning rate’
- Narrowly, regularization (**commonly defined as adding Ridge or LASSO penalty**) is difficult to implement for Tree algorithms since they are mostly heuristic
- However, regularization (**as any means to prevent overfit**) can be accomplished through these methods:
  - Limiting the maximum depth of trees
  - Using bagging ensembles (they are designed to reduce variance as parallel methods)
  - Set stricter stopping criterion on when to split a node further (e.g. set minimum gain, or the number of samples)

Alternative Models


Under-fitting
(too simple to explain the variance)
Appropriate-fitting
Over-fitting
(forcefitting -- too good to be true)

A Different Look at the World: Bayesian Linear Regression

- We formulate linear regression using **probability distributions** rather than **point estimates**
- The response y is not estimated as a single value, but is assumed to be drawn from a probability distribution
- The model for **Bayesian Linear Regression** with the response sampled from a normal distribution is

$$y \sim N(\beta^T X, \sigma^2 I)$$

- The output y is generated from a normal (Gaussian) distribution characterized by a mean and variance
- The mean for linear regression is the transpose of the weight matrix multiplied by the predictor matrix
- The variance is the square of the standard deviation σ (multiplied by the Identity matrix because this is a multi-dimensional formulation of the model)

UMBC

A Different Look at the World: Bayesian Linear Regression

- The aim of Bayesian Linear Regression **is not to find the single "best" value** of the model parameters, but rather to **determine the posterior distribution for the model parameters**
- We start out with an initial estimate, **our prior**, and as we gather more evidence, **our model becomes more accurate**
- In problems where we have **limited data** or **have some prior knowledge**, the Bayesian Linear Regression approach can both incorporate **prior information** and integrate **uncertainty**
- The **posterior probability** distribution of the model parameters is defined as

$$Posterior = \frac{Likelihood * Prior}{Normalization}$$

- L1 and L2 can also have a Bayesian interpretation
  - See http://bjlkeng.github.io/posts/probabilistic-interpretation-of-regularization/

Least-Angle Regression (LARS)

- **LARS** is similar to forward stepwise regression
- At each step, **LARS** finds the predictor most correlated with the response
- When multiple predictors have equal correlation, instead of continuing along the same predictor, it proceeds in a direction ‘equi-angular’ between the predictors
- It follows the same general scheme of forward stepwise regression
- However, it does not add a predictor fully into the model
- The coefficient of that predictor is increased only until that predictor is no longer the one most correlated with the residual r
- The algorithm starts with all coefficients equal to zero
- Then, it finds the predictor that is most correlated with y
- It increases the coefficient in the direction of the sign of its correlation with y
- Then, it stops when other predictors reach the same degree of correlation with r as the first predictor
- It is useful when the number of dimensions is significantly greater than the number of points
- If two variables are almost equally correlated with the response, then their coefficients should increase at approximately the same rate
- LARS does not work well if the data is high dimensional or has multicollinearity