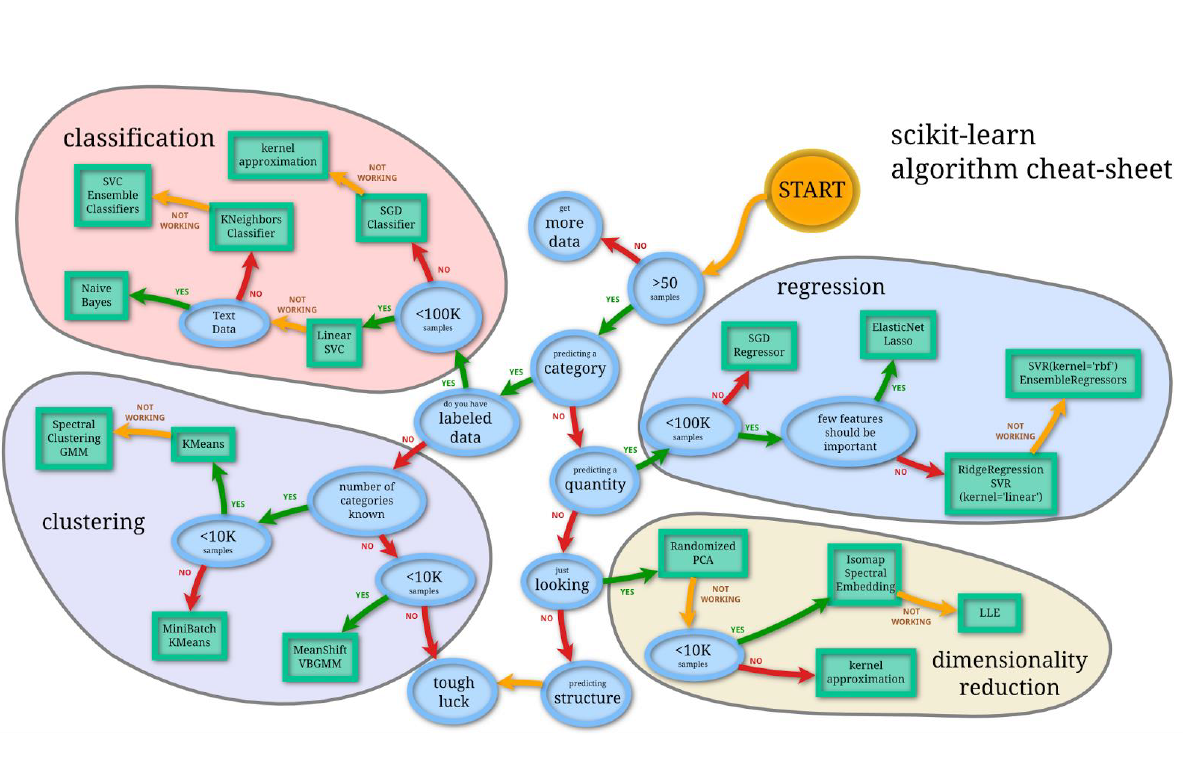
**Machine Learning Algorithms:**



**Metrics for Regressions:**

RSquared:

TSS = ESS + RSS where ESS is the variance attributed to error and RSS (Residual Sum of Squares) is one way of quantifying how much error existing in the fitted model.

RSquared = ESS / (RSS + ESS) or ESS / TSS

Note: RSquared increases artificially as a side effect of increasing the number of independent variables.

Adjusted RSquared:

Adjusted RSquared = RSquared – ( (1 – Rsquared) \* (K / (N – K – 1) )

K = number of variables, N is the sample size.

Mean Squared Error (MSE) and “Square” Root Mean Squared Error (RMSE)

F-Statistics:

F-Statistic = (RSquared / (1 – Rsquared)) \* ( (N – K – 1) / K )

(F-Statistics is not the same as the F1-measure)

Plot Residuals vs Actual y values.

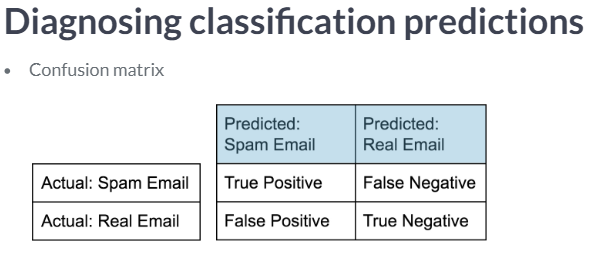
Plot Actual y vs. Predicted y

Plot histogram of residuals

Plot feature Xi vs Xj to look for dependencies.

**Metrics for Classification:**

Confusion Matrix



From Confusion Matrix Calculate or run Classification Report

Precision = true positive / (true positive + false positive)

Recall = true positive / (true positive + false negative)

(also calledsensitivity, hit rate, true positive rate)

Accuracy = (true positive + true negative) /

(true positive + true negative + false positive + false negative)

F1score = 2 \* (Precision \* Recall) / (Precision + Recall)

The **support** is the number of samples of the true response that lie in that class

High Precision (not many false positives)

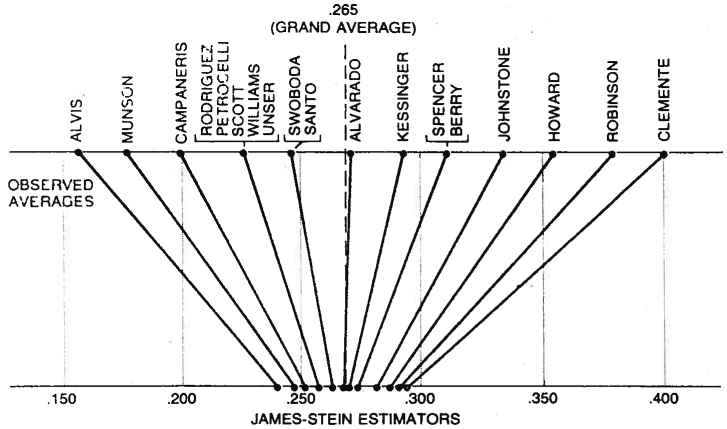
High Recall (predicted most true positives successfully)

**Algorithm: Linear Regression**

Technical Information:

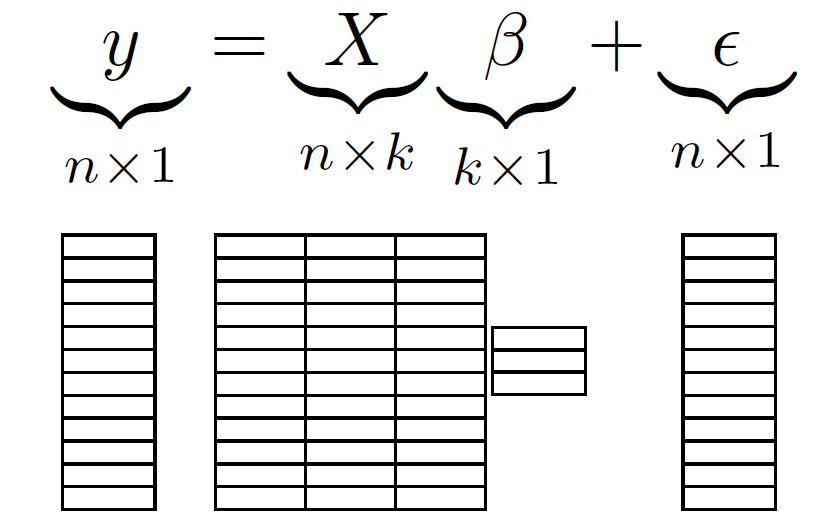
RTTM – Regression Toward the Mean. If you do excellent in first year sports, it could be a combination of skill and luck.

Stein’s Paradox: Shrinking all individual averages toward the grand average (average of averages)



MTTM is a statistical phenomenon that occurs when unusually large or unusually small measurement values are followed by values that are closer to the population mean. This is due to random measurement error or, put another way, non-systematic fluctuations around the true mean.

OLS – ordinary least squares is the criterion for calculating the linear relationship.



k is the number of predictor variables (independent variables for features)

n is the sample size

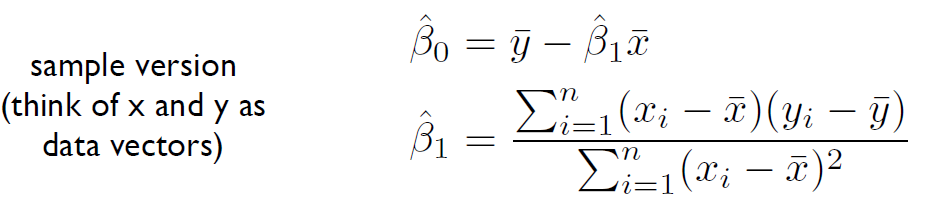
beta’s are the coefficients that you are trying to predict for determining y.

epsilon is the error

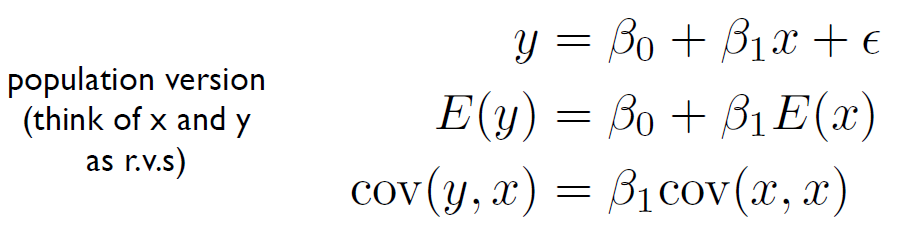
y is the dependent variable or target variable you are trying to predict.

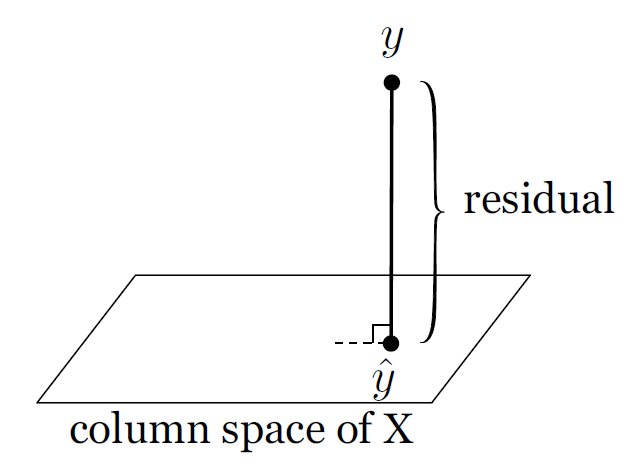
NOTE: X’s don’t have to be at power of 1. X’s can be of higher degree. The beta’s must be degree one.

Calculating OLS:



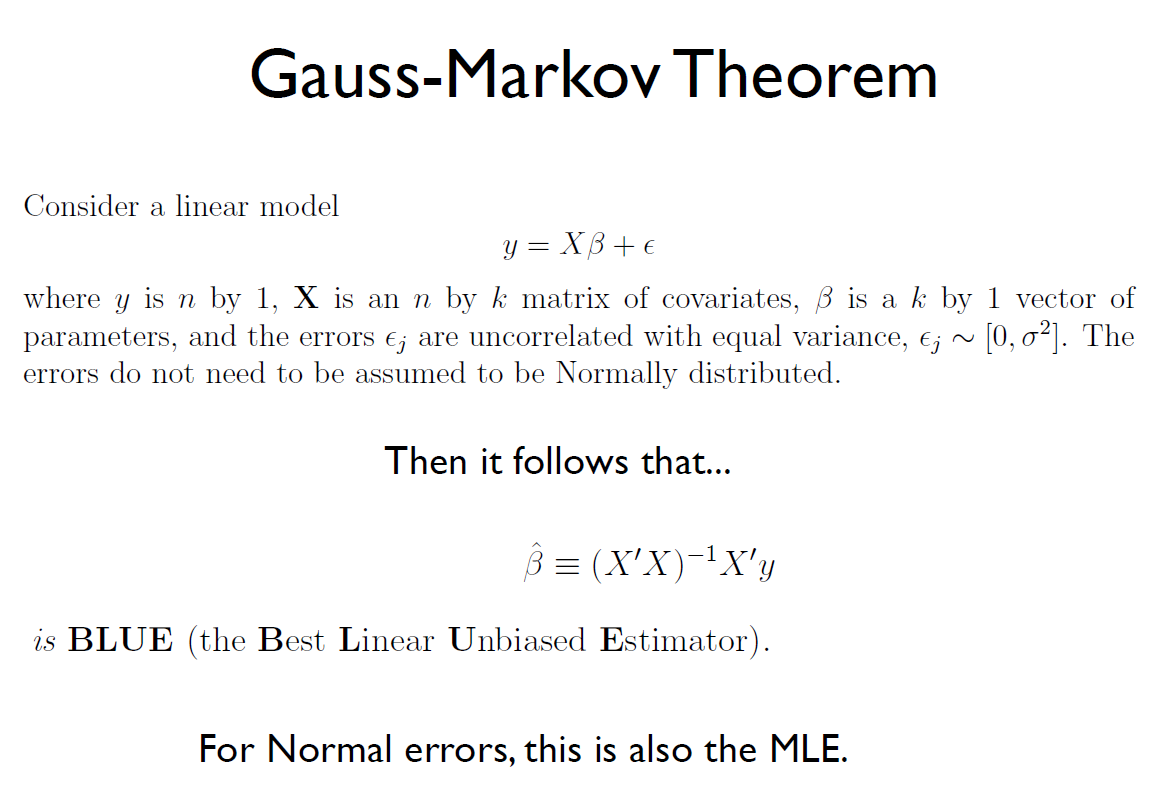
Another way to think about it is:





Regression is a projection. Find the closest point in the column space of X. What combination of X can you get that is the closest to y.

If you are working with normal distributions then the best predictor of y is linear. Theorem as follow states this.



Seven OLS Assumptions for Linear Regression:

1. The regression model is linear in the coefficients and the error term. a regression model is linear when all terms in the model are either the constant or a parameter multiplied by an independent variable.
2. The error term (epsilon) has a population mean of zero. The error term accounts for the variation in the dependent variable that the independent variables do not explain.
3. All independent variables are uncorrelated with the error term. If an independent variable is correlated with the error term, we can use the independent variable to predict the error term, which violates the notion that the error term represents unpredictable random error.
4. Observations of the error term are uncorrelated with each other. One observation of the error term should not predict the next observation. Plot Residuals vs. Observed Values…you should see no pattern.
5. The error term has a constant variance (no heteroscedasticity). The variance of the errors should be consistent for all observations. Again, look at the Residuals vs. Observed Values.
6. No independent variable is a perfect linear function of other explanatory variables. Perfect correlation suggests that two variables are different forms of the same variable using Pearson’s correlation coefficient. EG… SLG + OBP = OPS. If SLG and OBP are X’s then you get a perfect linear relationship…SLG and OBP are just another way of saying OPS.
7. The error term is normally distributed. OLS does not require that the error term follows a [normal distribution](https://statisticsbyjim.com/basics/normal-distribution/) to produce unbiased estimates with the minimum variance. However, satisfying this assumption allows you to perform statistical hypothesis testing and generate reliable confidence intervals and prediction intervals. Use a QQPlot to plot residuals or test them using a normal test.

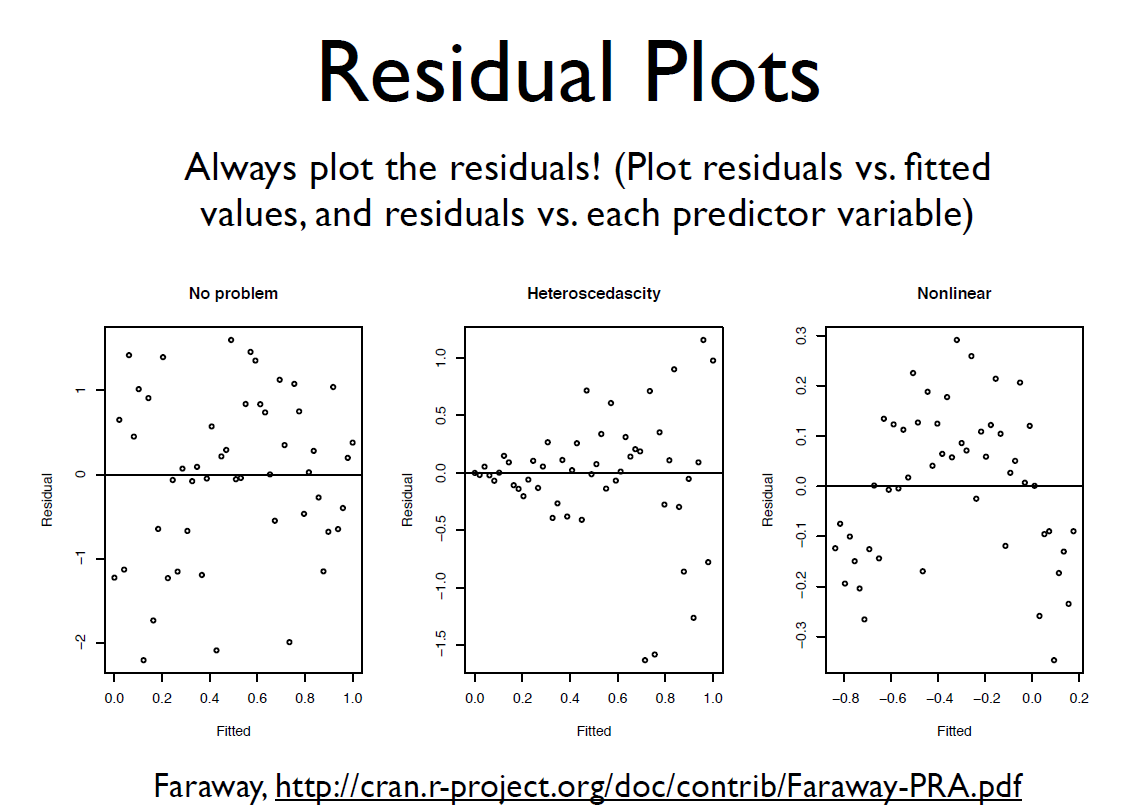
In a nutshell, your linear model should produce residuals that have a mean of zero, have a constant variance, and are not correlated with themselves or other variables. If these assumptions hold true, the OLS procedure creates the best possible estimates.

The best estimates are those that are unbiased and have the minimum variance. When your model satisfies the assumptions, the Gauss-Markov theorem states that the OLS procedure produces unbiased estimates that have the minimum variance.

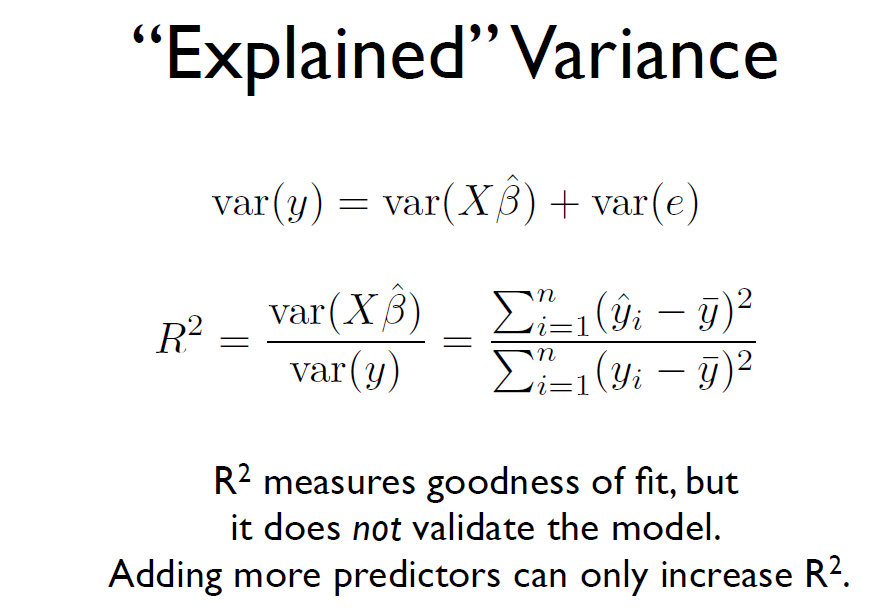
Gauss-Markov Theorem states that OLS is BLUE (Best Linear Unbiased Estimator) – minimum variance or narrowest sampling distribution. That is,  when your model satisfies the assumptions, OLS coefficient estimates follow the tightest possible sampling distribution of unbiased estimates compared to other linear estimation methods.

Should avoid having predictor variables that are highly correlated with each other (collinearity

results in instability, high variances in estimates, and worse interpretability)



How good is the model for the business problem:



Sample Code:

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X,y, test\_size = 0.4, random\_state=61)

reg = LinearRegression()

reg.fit(X\_train, y\_train)

y\_pred = reg.predict(X\_test)

# function that calculates RSquared, MSE, RMSE, and presents plots of y and predicted y, residuals and y # and residual histograms.

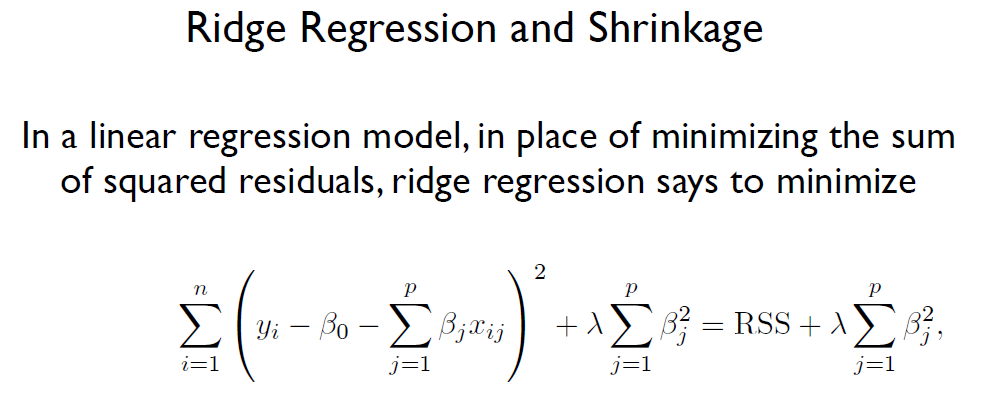
lr\_results(df,X\_test,y\_test,y\_pred,path,'OPSpredictions.csv',stats\_list,reg)

**NOTE: some predictors or features need to be rescaled. High values and relatively low feature values can affect the model in erroneous ways. You can normalize data by zero-mean normalization (Xi – mean (Xi)) / standard deviation (Xi). Or by min/max normalization. (Xi – min(Xi)) / (max(Xi) – min(Xi))**

Variations of Linear Regression: Takes into account Shrinkage and Curse of Dimensionality.

**Regularization: Penalizes for large coefficients. Larger coefficients tends to lead to overfitting**.

**Ridge Regression (extension of linear regression):**



Lamda is a tuning value…use cross validation to choose it. This is a class of penalty imposing regression models. Don’t want sum of squared betas to be too large.

Ridge regression uses a type of [shrinkage estimator](https://www.statisticshowto.datasciencecentral.com/shrinkage-estimator/) called a ridge estimator. Shrinkage estimators theoretically produce new [estimators](https://www.statisticshowto.datasciencecentral.com/estimator/)that are shrunk closer to the “true” population parameters. The ridge estimator is especially good at improving the least-squares estimate when multicollinearity is present. Deals with situations where variables (k) are more than observations (n).

**Multicollinearity can be briefly described as the phenomenon in which two or more identified predictor variables are linearly related, or codependent. Regularization techniques can be used to combat multicollinearity : L1 (Lasso), L2 (Ridge Regression), and Elastic Nets. A Pearson Coefficient of 0.8 or higher may indicate severe multicollinearity. Possible for individual regression coefficients to be insignificant but for the overall fit of the equation to be high (overfitting)**

L2 Regularization: Ridge regression penalizes sum of squared coefficients. Makes coefficients small where there is collinearity.

# Ridge Regression sample code

ridge = Ridge(alpha=0.001, normalize=True)

ridge.fit(X\_train, y\_train)

y\_pred = ridge.predict(X\_test)

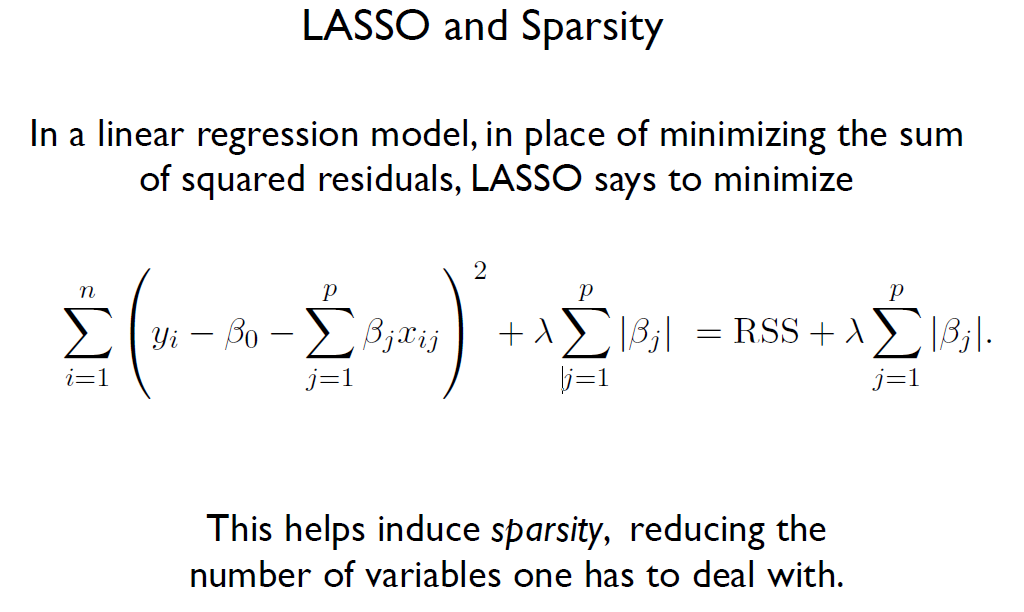
print('\n')

print('Ridge ...')

print('\n')

lr\_results(df,X\_test,y\_test,y\_pred,path,'OPSpredictionsRidge.csv',stats\_list,ridge) # custom function

**Lasso Regression (extension of linear regression) – shrinking the features to features of most impact.**



Takes variables which don’t do much to zero.

L1 Regularization: penalizes the sum of their absolute values. Takes variables to zero what don’t do much for the model. (ones that have collinearity)

Lamda is a tuning value…use cross validation to choose it.

lasso = Lasso(alpha=0.0001)

lasso\_coef = lasso.fit(X\_train, y\_train).coef\_

y\_pred = lasso.predict(X\_test)

lr\_results(df,X\_test,y\_test,y\_pred,path,'OPSpredictionsLasso.csv',stats\_list,lasso)

cols = feature\_list

plt.plot(range(len(cols)), lasso\_coef)

plt.xticks(range(len(cols)), cols, rotation=45)

plt.ylabel('Coefficients')

plt.show()

**Elastic Net Regression – combination of Ridge and Lasso regression.**

Elastic Net is a mix of both L1 and L2 Regularizations. A penalty is applied to the sum of the absolute values and to the sum of the squared values.

#Elasticnet with GridSearchCV

# Create the hyperparameter grid

l1\_space = np.linspace(0, 1, 30)

param\_grid = {'l1\_ratio': l1\_space}

# Instantiate the ElasticNet regressor: elastic\_net

elastic\_net = ElasticNet()

# Setup the GridSearchCV object: gm\_cv

gm\_cv = GridSearchCV(elastic\_net, param\_grid, cv=5)

# Fit it to the training data

gm\_cv.fit(X\_train, y\_train)

# Predict on the test set and compute metrics

y\_pred = gm\_cv.predict(X\_test)

lr\_results(df,X\_test,y\_test,y\_pred,path,'OPSpredictionsGSElastic.csv',stats\_list,gm\_cv)

print("Tuned ElasticNet l1 ratio: {}".format(gm\_cv.best\_params\_))

**Decision Trees**

**Pros**

•Fast training

•Can use categories

•Don’t have to scale values

•Fast prediction

•Easy to understand

•Easy to interpret

•Can do multiple classification (not just binary)

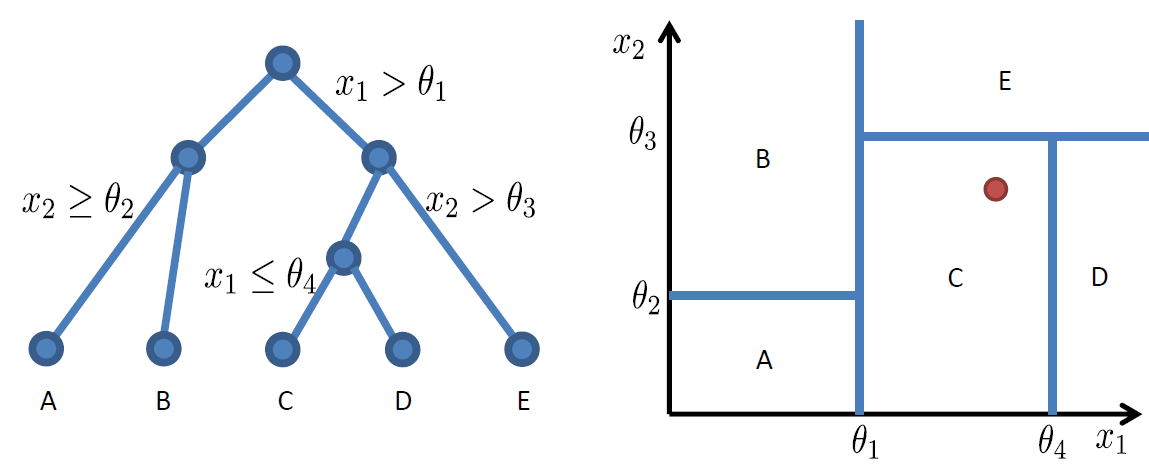
**Cons**

•Sensitive to small changes in the data

•Overfitting (not popular for this reason)

•Only looks at a node at a time

•Only axis aligned splits



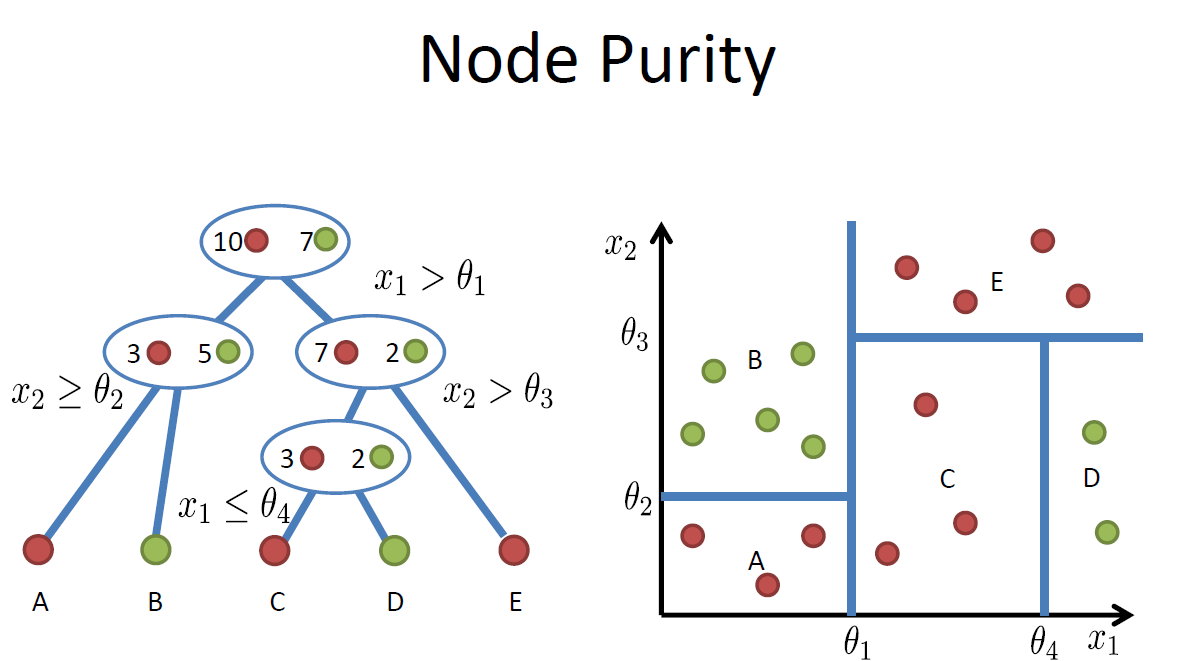
**Only does axis aligned splits (not diagonal)**

**Decisions?**

**Which features to query.**

**What thresholds to choose (how deep do you go).**

**Lowest level tells you the category it belongs to (classification)**

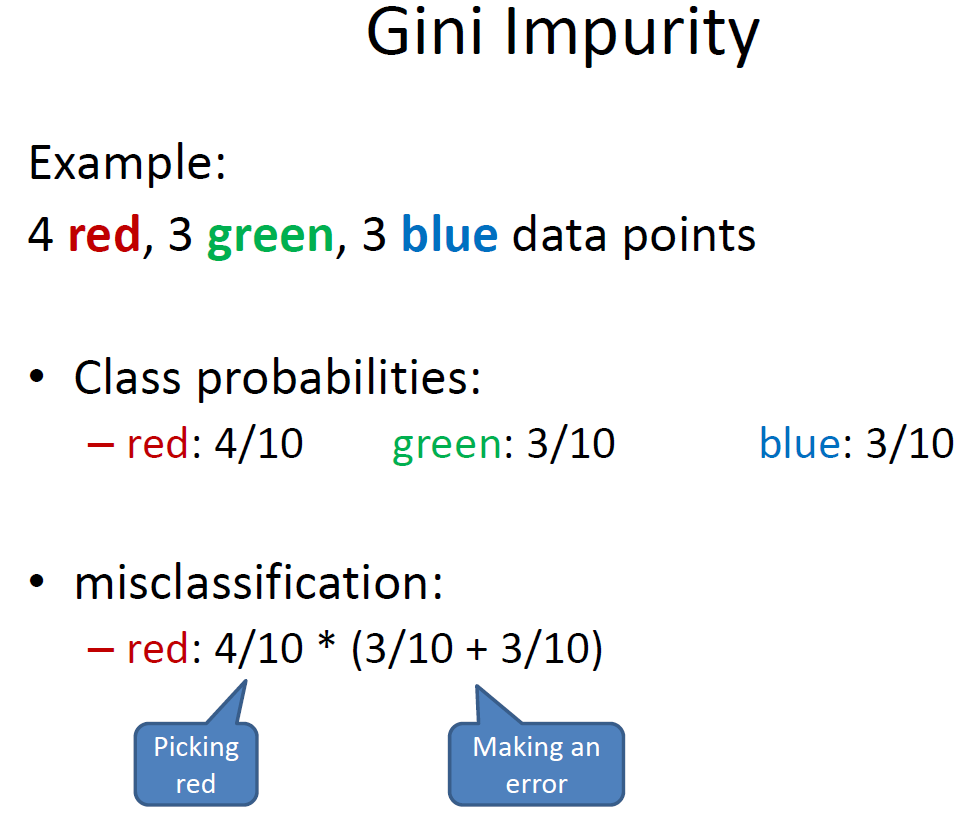


**End nodes only have red or green exclusively (only one class in them). Extreme case would have leaf node for every point (overfitting).**

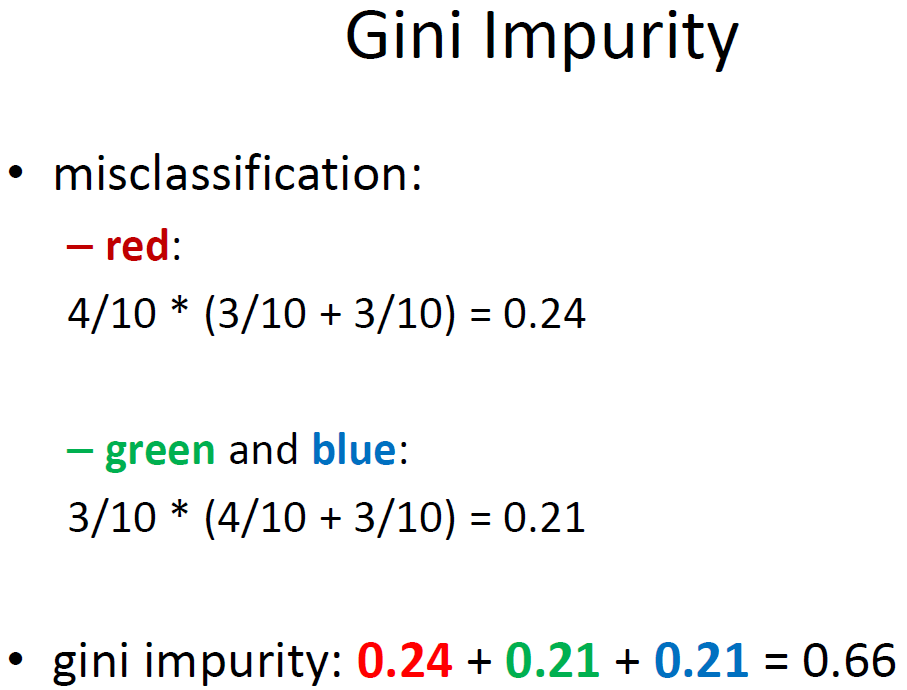
**Gini Impurity is used for node purity. Used in sklearn.**

**Gini Impurity:**

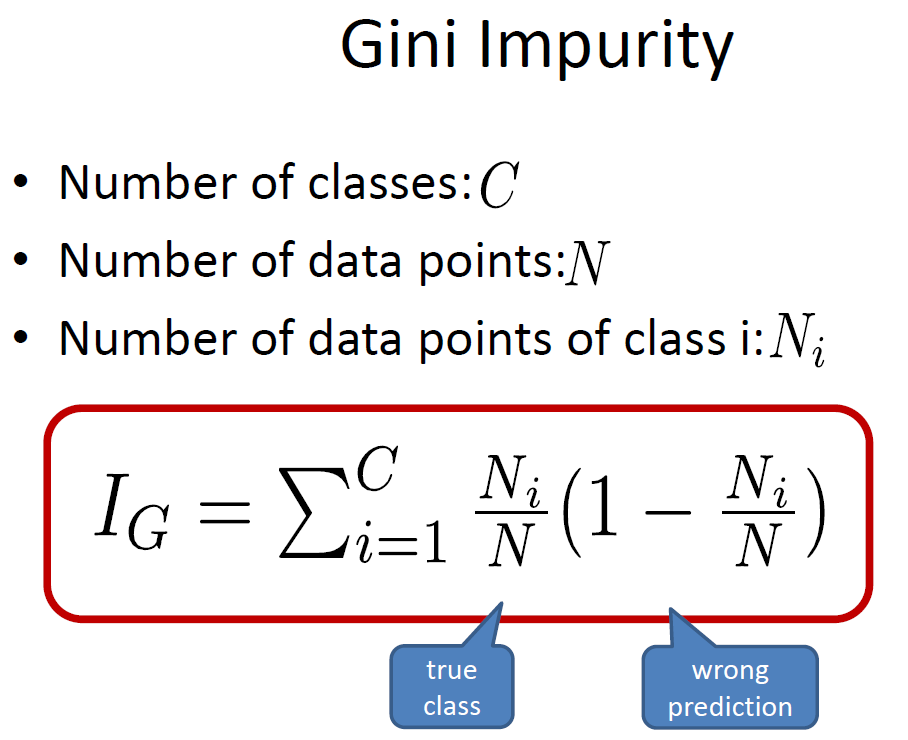
* **Expected Error**
* **If you randomly choose a sample**
* **And predict the class of the entire node based upon it.**

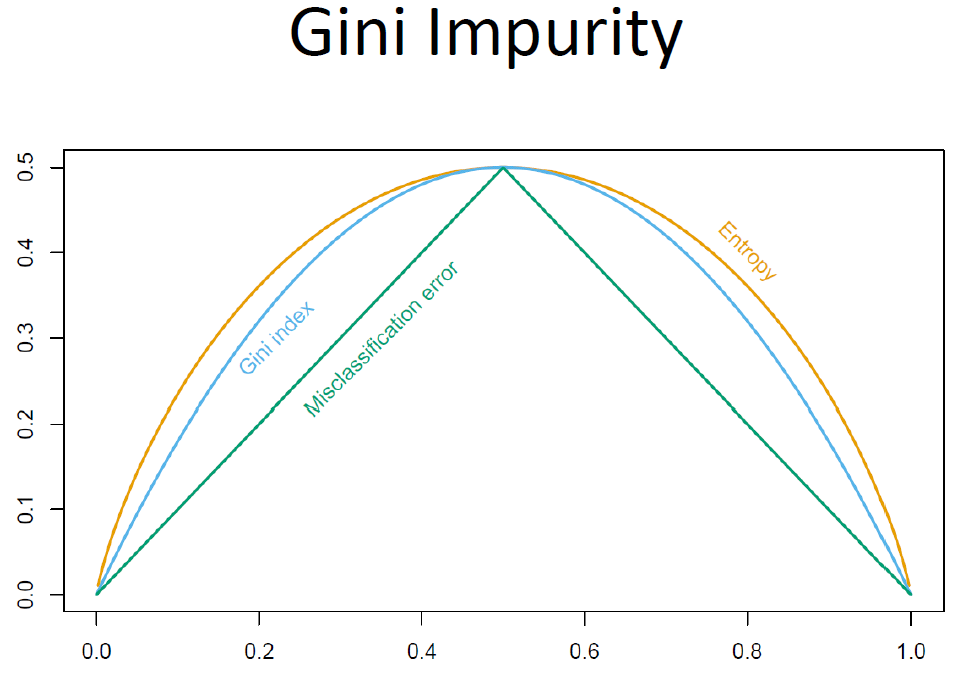


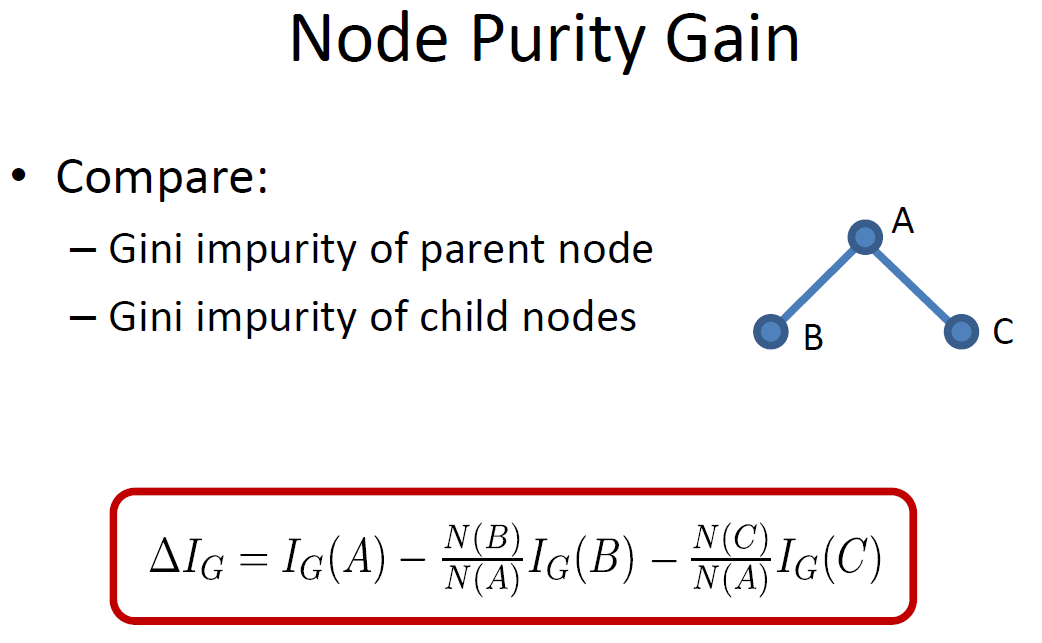
**Probability picking a red ball and chance of making an error.**

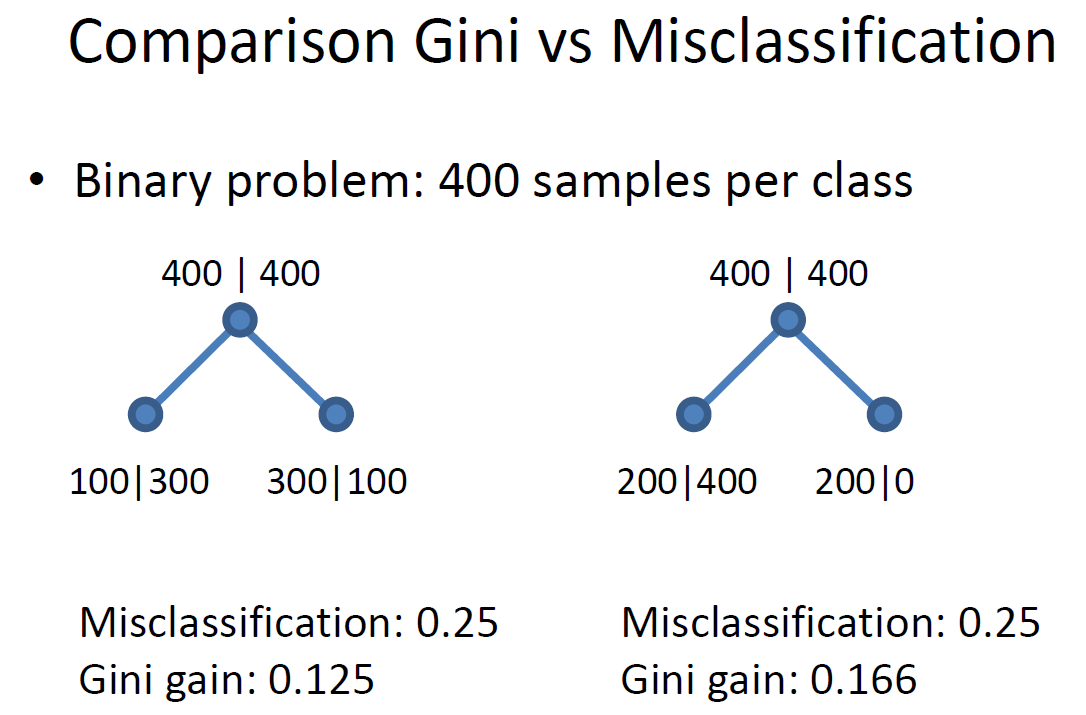


**Gives gini impurity for a specific node.**









Pseudocode:

Check if already finished

For each feature Xi

Calculate the gain from splitting on Xi

Let Xbest be the feature with the highest gain

Create a decision node that splits on Xbest

Repeat on the sub-nodes

Stop When:

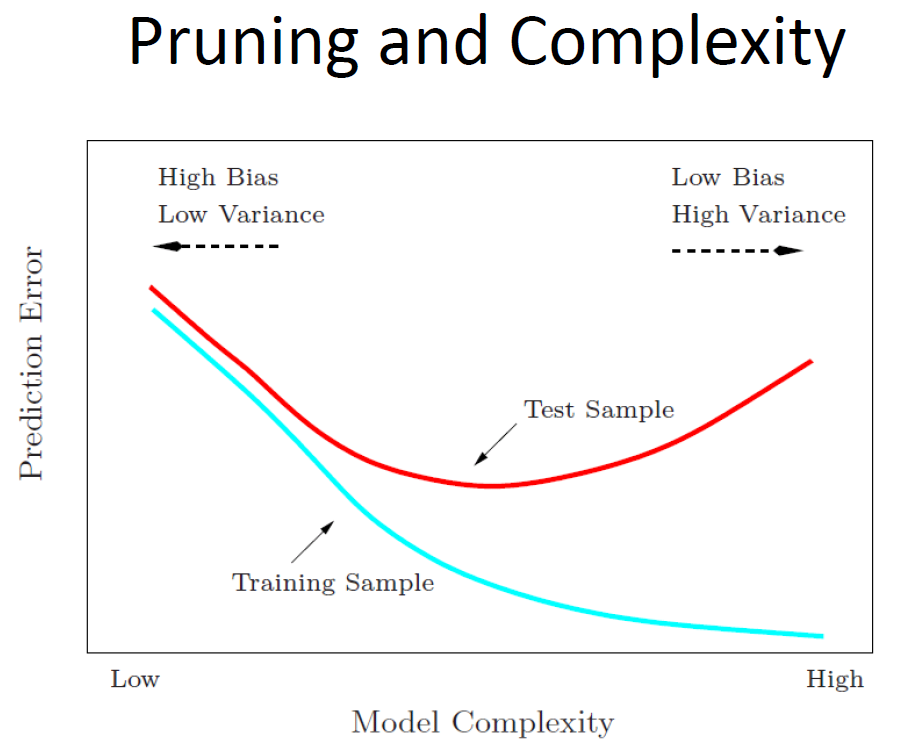
Node contains only one class

Node contains less than x data points

Max depth is reached

Node purity is sufficient

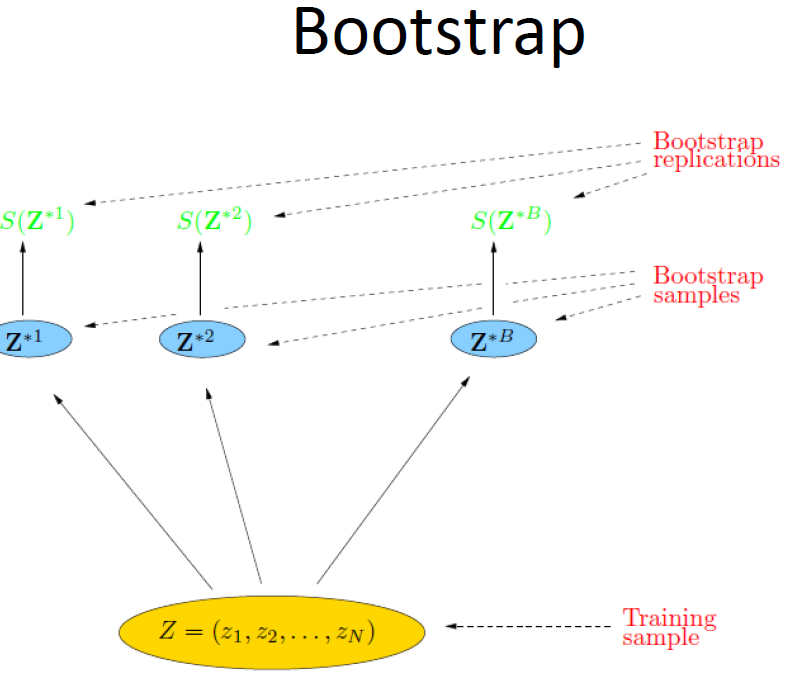
Overfitting occurs (cross-validation)



**Random Forest (Ensemble Methods) – learning multiple trees with bagging.**

**Bootstrapping**

* **Resampling method from statistics**
* **Useful to get error bars on estimates**
* **Take N data points**
* **Draw N times with replacement**
* **Get estimate for each bootstrap sampled**



**Probability of choosing observation n = 1/N**

**Probability of not choosing n = 1 – (1/N)**

**Probability of not choosing n in N draws = (1 – (1/N)) \*\* N**

**Probability of choosing n in N draws = 1 – (1 – (1/N)) \*\* N**

**p(n in bootstrap) = 1 – e \*\* -1 or approximately 0.632 or the bootstrap sample omits about 36.8% of data**

**BAGGING (Bootstrap AGGregatING)**

•Sample with replacement from your data set

•Learn a classifier for each bootstrap sample

•Average the results

**Bagging Results**

•Reduces overfitting (variance) without introducing too much bias

•Normally uses one type of classifier

•Decision trees are popular

•Not helping with linear models (if you average a linear model, you end up with a line)

•Easy to parallelize

**Random Forests extends Bagging of Decision Trees**

•Builds upon the idea of bagging (randomizing bootstrap)

•Each tree built from bootstrap sample

•Node splits calculated from random feature subsets

**Extension of bagging of decision trees:**

Generate bootstrap training samples, and a tree based on each, but also randomize set of predictors allowed each time a split is considered.

• Typically, # of random predictors chosen to be of order the square root of the total number of predictors.

• All trees are fully grown

• No pruning

• Two parameters

– Number of trees

– Number of feature

# sample code random forest regressor

rf = RandomForestRegressor(n\_estimators=1000,random\_state=61)

rf.fit(X\_train, y\_train)

pred = rf.predict(X\_test)

errors = abs(pred - y\_test)

df\_results = df.loc[X\_test.index, :]

df\_results['predOPS'] = pred

lr\_results(df,X\_test,y\_test,pred,path,'OPSpredictionsRF.csv',stats\_list,rf) # function for metrics

# sample code random classifier

rf\_cls = RandomForestClassifier(bootstrap=True, class\_weight=None, criterion='gini',

max\_depth=7, max\_features='auto', max\_leaf\_nodes=None,

min\_impurity\_decrease=0.0, min\_impurity\_split=None,

min\_samples\_leaf=1, min\_samples\_split=2,

min\_weight\_fraction\_leaf=0.0, n\_estimators=1000, n\_jobs=None,

oob\_score=False, random\_state=0, verbose=0, warm\_start=False)

rf\_cls.fit(X\_train,y\_train)

y\_pred = rf\_cls.predict(X\_test)

print(accuracy\_score(y\_pred,y\_test))

print(confusion\_matrix(y\_test,y\_pred))

print(classification\_report(y\_test,y\_pred))

**Knn**

**Logistic Regression**

**SVM**

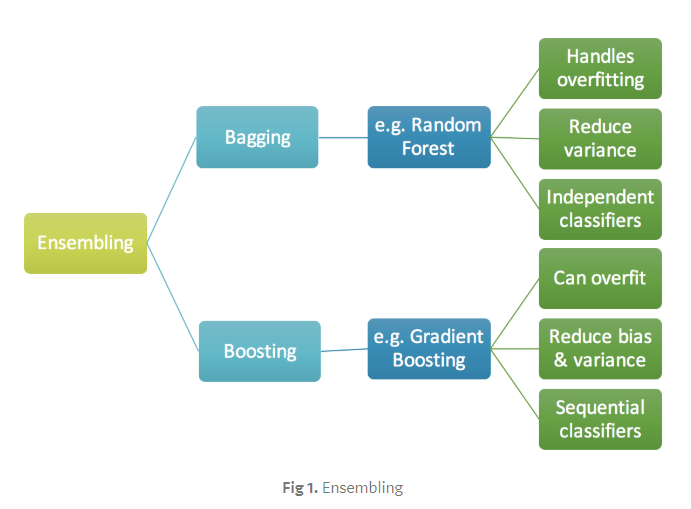
**Boosting**

**“Boosting is one of the most powerful learning ideas introduced in the last twenty years.”** Hastie-Tibshirani-Friedman, *The Elements of Statistical Learning: Data Mining, Inference, and Prediction*,

Springer (2009)

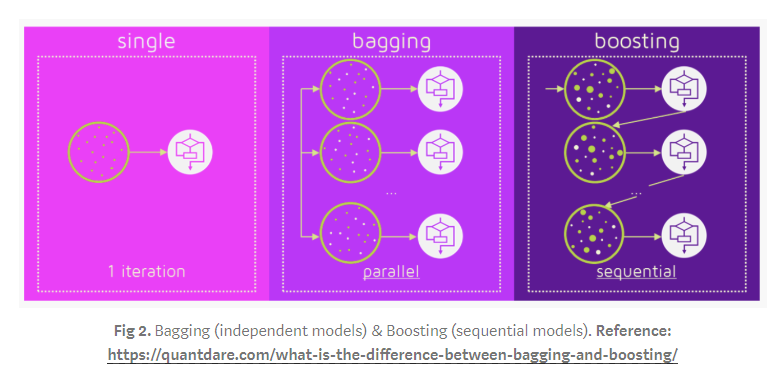
**Boosting**is an ensemble technique in which the predictors are not made independently, but sequentially.

Gradient boosting is a machine learning technique for regression and classification problems, which produces a prediction model in the form of an ensemble of weak prediction models, typically decision trees. (Wikipedia definition)



Algorithm:

• We first model data with simple models and analyze data for errors.  
• These errors signify data points that are difficult to fit by a simple model.  
• Then for later models, we particularly focus on those hard to fit data to get them right.  
• In the end, we combine all the predictors by giving some weights to each predictor.



#sample XGBoost Regressor

reg\_xgb = xgb.XGBRegressor(objective ='reg:squarederror',

colsample\_bytree=0.6,

learning\_rate=0.2,

max\_depth=3,

n\_estimators=60,

subsamples=0.6,

alpha=1,

gamma=0.001

)

reg\_xgb.fit(X\_train, y\_train)

y\_pred = reg\_xgb.predict(X\_test)

lr\_results(df,X\_test,y\_test,y\_pred,path,'OPSpredictionsXGB.csv',stats\_list,reg\_xgb)

#sample XGBoost Classifier

xgb\_cls = XGBClassifier()

xgb\_cls.fit(X\_train,y\_train)

y\_pred = xgb\_cls.predict(X\_test)

print(accuracy\_score(y\_pred,y\_test))

print(confusion\_matrix(y\_test,y\_pred))

print(classification\_report(y\_test,y\_pred))