

Lecture 2 : Supervised Learning

Scott Broderick

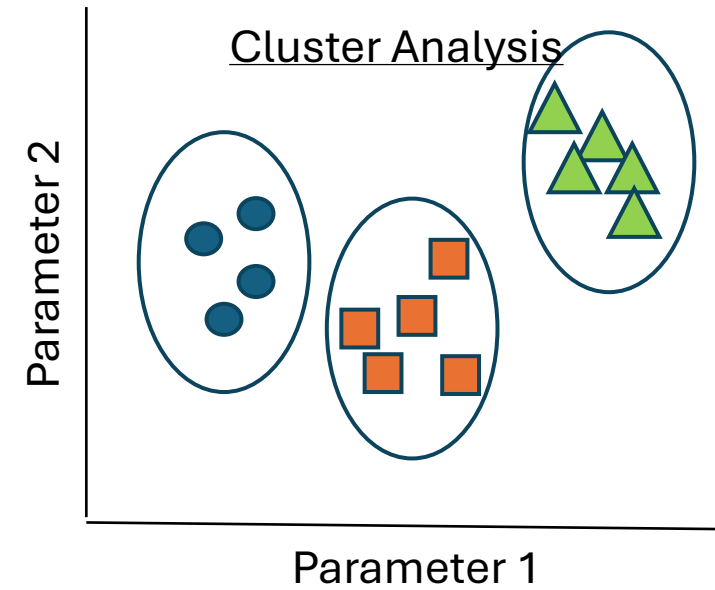
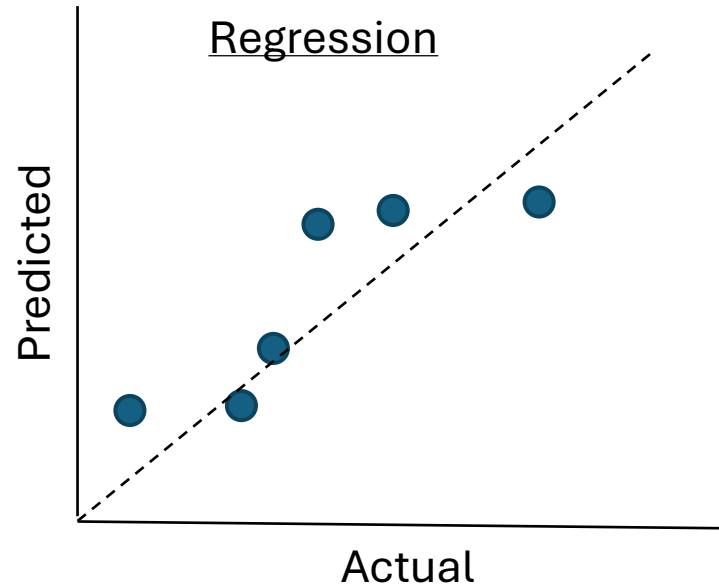
Department of Materials Design and Innovation
University at Buffalo

September 17, 2026

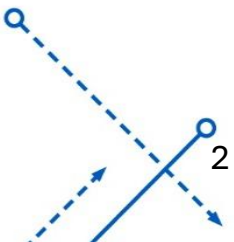
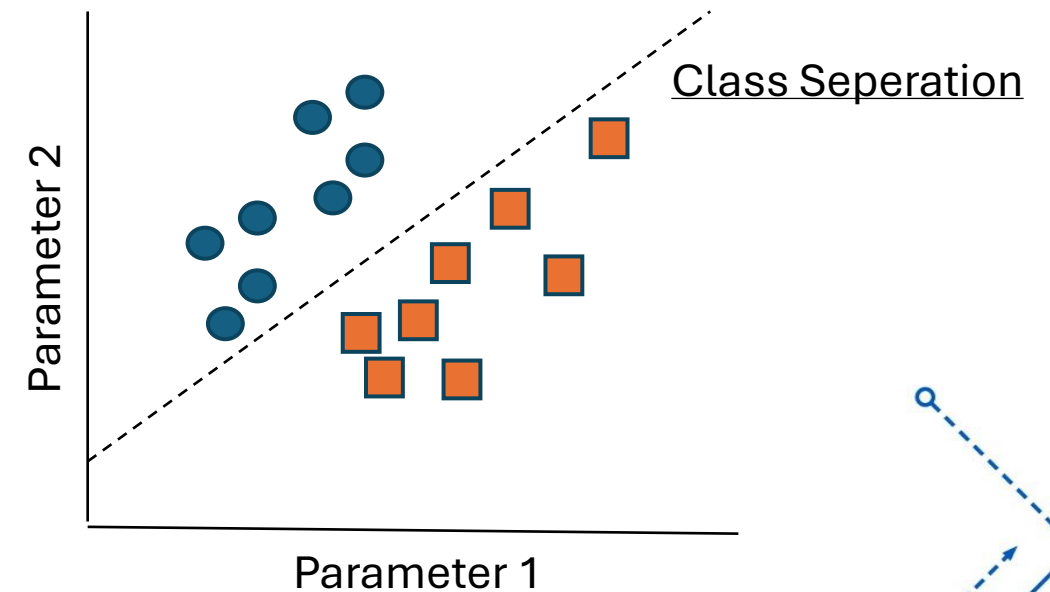
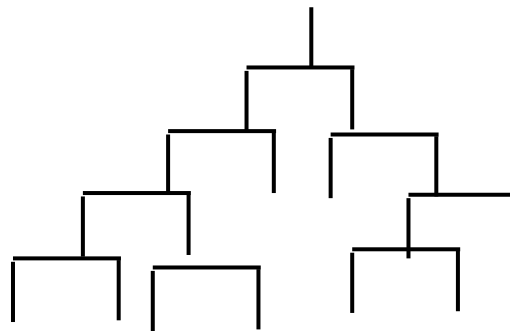
scottbro@buffalo.edu
mdi.buffalo.edu



Examples of Supervised Learning



Tree Based Models



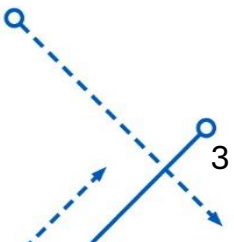
Data Dimensionality and Diversity

In selecting our input data (ie. the features we are using to train our models), we have consideration of both dimensionality and diversity.

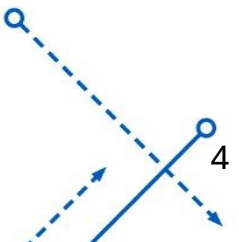
- For dimensionality, usually we could say the more data the better. However, we have to keep in mind issues with over-fitting. That is, especially when we have small number of property / response measurements.

--- Issues with over-fitting and selection of features depends on the method used (sometimes need to consider, sometimes we don't)

- For diversity, two features which are highly correlated do not contribute additional or useful added information. Therefore, we are interested in features which are diverse (ie. largely independent from other features).



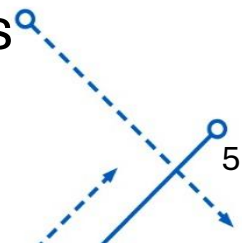
- Accuracy : how good is the model for what we know? ... the goal in this case is to get as good of a fit as we can... this is applying to both training and test sets
- Robustness : how good is the model for predicting what we don't know? This is largely dictated through comparison of training and test data (ie. based on underlying factors and not overfit).... Becomes particularly important with small data.
- Interpretability : can we explain the model? If looking at the features, which are most important, can we come up with some justification This is often a trade off with techniques giving better accuracy



Standardizing Data

- Typical approach is to subtract mean and divide by standard deviation But when should we standardize the data?
- In case of regression, we may develop a model that is :

$$P = c_1 * f_1 + c_2 * f_2 + \dots$$
 where P is property, c is coefficient, and f is feature....
- In this case, it doesn't matter if the features have been normalized The coefficient can take care of that So if we go from meter to centimeter, the model will be the same, just multiplying c by 100.
- If we have a situation with comparative data (ie. without the corresponding coefficients Eg. manifold learning) then we need to standardize. That is, changing unit from meter to centimeter doesn't change the data, but changes the results.

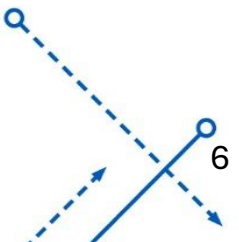


Probability

- When we will predict things, keep in mind that there is uncertainty and we are only predicting what we think is the highest probability result.... That is, we are not saying this is the result, but rather it is more likely to be the result.
- Bayes' Theorem : We are predicting the probability of an event given the knowledge of another event I.e. does having more information change our expectation of an event?

$$P(E_j | E) = \frac{P(E | E_j)P(E_j)}{\sum_{i \in \Lambda} P(E | E_i)P(E_i)}, \quad j \in \Lambda.$$

- Let $\Omega, \mathcal{F}, P$ be a probability space and
- Let $E_i : i \in \Lambda$ be a countable collection of mutually exclusive and exhaustive events with $P(E_i) > 0, i \in \Lambda$.
- Then, for any event $E \in \mathcal{F}$ with $P(E) > 0$



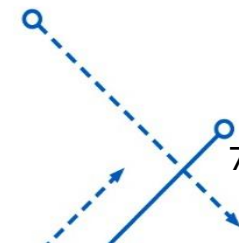
Probability

$$P(E_j | E) = \frac{P(E | E_j)P(E_j)}{\sum_{i \in \Lambda} P(E | E_i)P(E_i)}, \quad j \in \Lambda.$$

Classic example : we have a bowl B1 which has 4 white and 6 black balls in it. We also have another bowl B2 which has 6 white and 4 black balls in it.

We shake a die, and if we shake 1 or 2, we select a ball from bowl 1 (B1). Else, we select a ball from bowl 2 (B2).

- (i) If we pick a white ball, find the conditional probability that it came from B1.
- (ii) If we pick a white ball, find the conditional probability that it came from B2.



Probability

$$P(E_j | E) = \frac{P(E | E_j)P(E_j)}{\sum_{i \in \Lambda} P(E | E_i)P(E_i)}, \quad j \in \Lambda.$$

W : we select a white ball

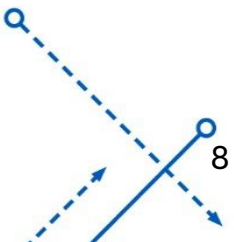
B1 : pick from bowl 1, and B2 : pick from bowl 2. These two events are exclusive and exhaustive.

$$P(E_1 | W) = \frac{P(W | E_1)P(E_1)}{P(W | E_1)P(E_1) + P(W | E_2)P(E_2)}$$

= $[4/10 \cdot 2/6] / [4/10 \cdot 2/6 + 6/10 \cdot 4/6] = 1/4$; And since they are mutually exclusive, (ii) has probability of $3/4$.

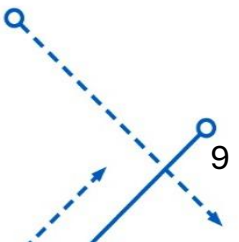
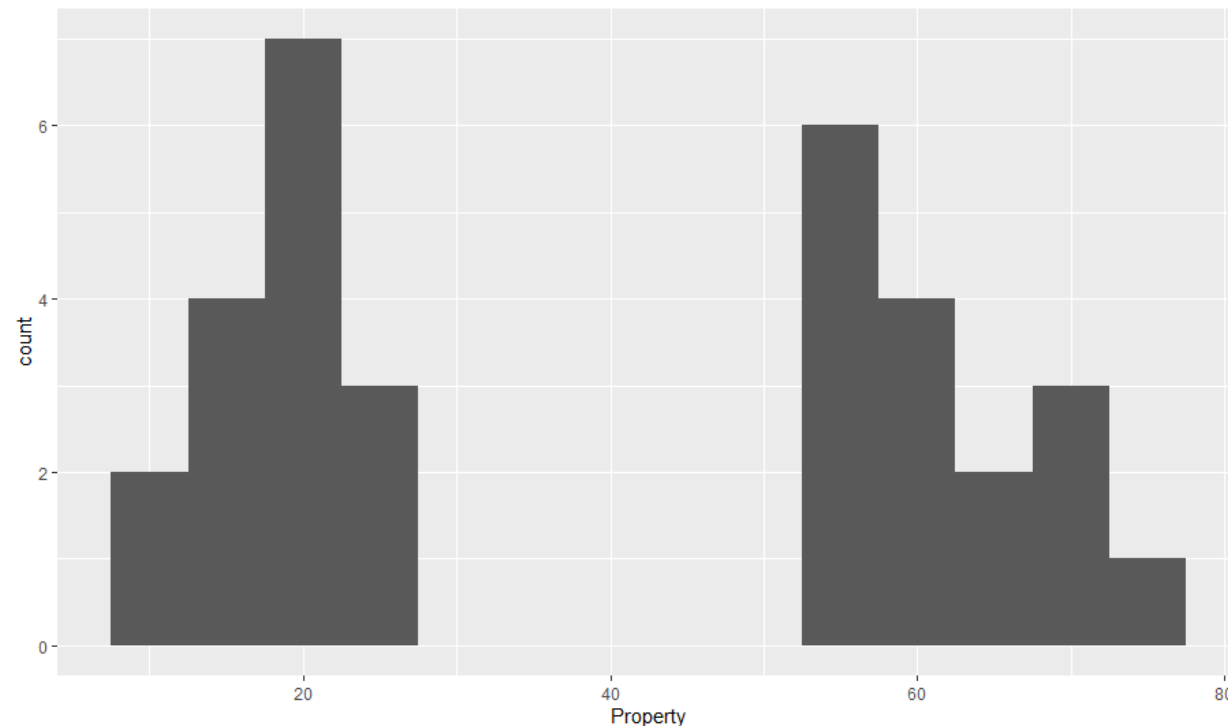
Thus : $P(E_1) = 1/3$, $P(E_1 | W) = 1/4$ and $P(E_2) = 2/3$ and $P(E_2 | W) = 3/4$

- (i) the probability of occurrence of event E_1 decreases in the presence of the information that the outcome will be an element of W ;
- (ii) the probability of occurrence of event E_2 increases in the presence of information that the outcome will be an element of W .



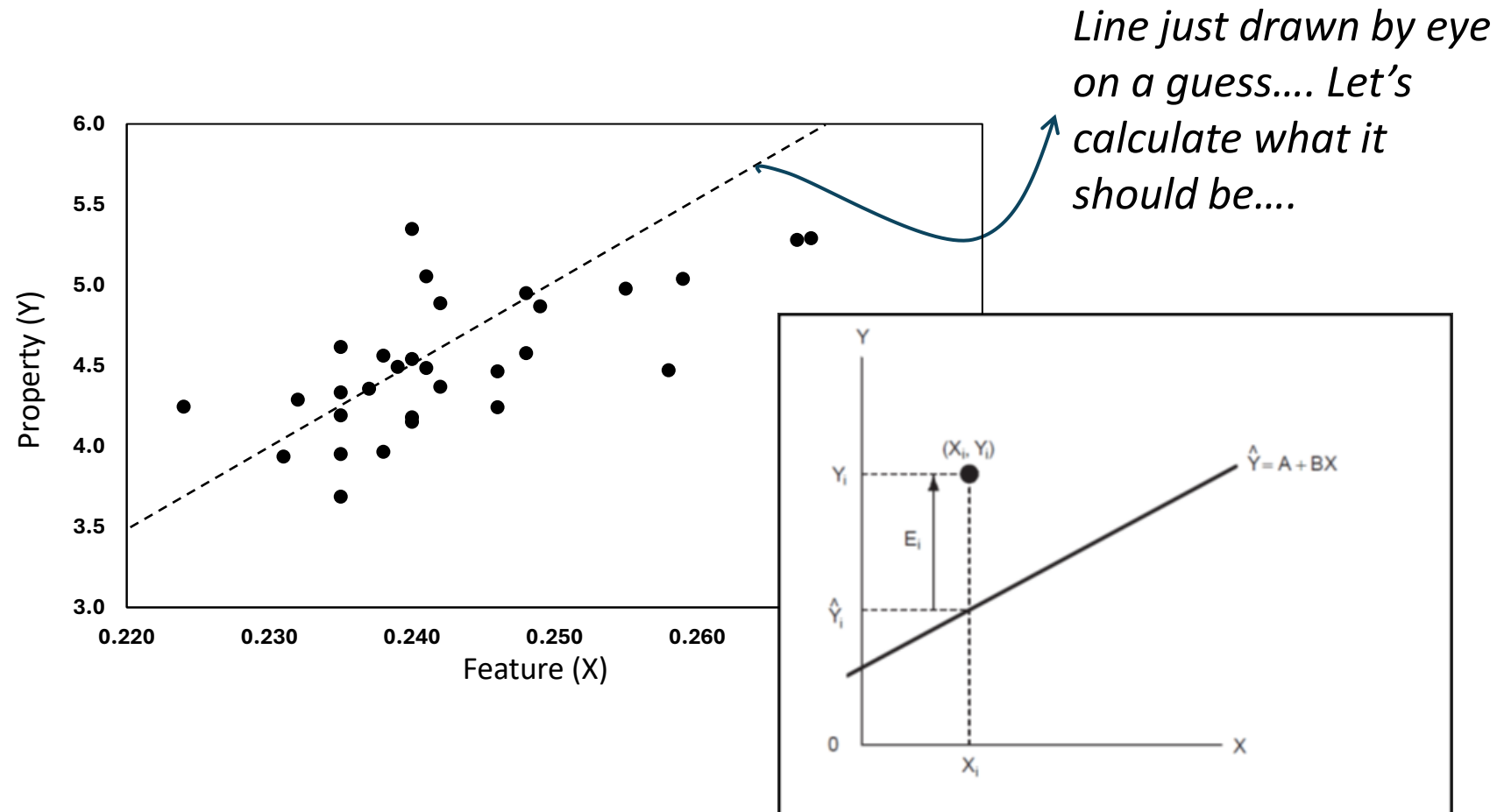
Data Distribution

- Non-normal distribution of data : Property does not follow distribution
- Can be a challenge Is it a data issue or is it that there are two different underlying effects that need to be modeled?



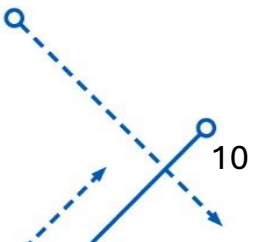
Least Squares Model

$$E_i = Y_i - \hat{Y}_i = Y_i - (A + BX_i)$$



A few assumptions in linear regression, among:

- Homoscedasticity : Error across all samples is the same.
- Property predicting can be linearly explained
- No auto-correlation : that is, the feature and the property are not the same, but with just a 'lag'



To easily calculate, we can use the following equations, with m as slope, b as intercept and equation of $y = mx + b$, with y being the predicted variable, and x is the predictor.

$$m = \frac{N \sum(xy) - \sum x \sum y}{N \sum(x^2) - (\sum x)^2}$$

$$b = \frac{\sum y - m \sum x}{N}$$

N is the number of measurements, xy is the product for a given measurement.

$$y = mx + b$$

$$m = \frac{N \sum(xy) - \sum x \sum y}{N \sum(x^2) - (\sum x)^2}$$

$$b = \frac{\sum y - m \sum x}{N}$$

For this toy data:

$N = 30$ (this is how many points are in our data)

Y is property

X is feature

$$\sum(xy) = 33.07$$

$$\sum X * \sum Y = 7.28 * 135.83 = 989.42$$

$$\sum(x^2) = 1.77158$$

$$(\sum x)^2 = 53.056$$

$$m = (30 * 33.07 - 989.42) / (30 * 1.77158 - 53.056)$$

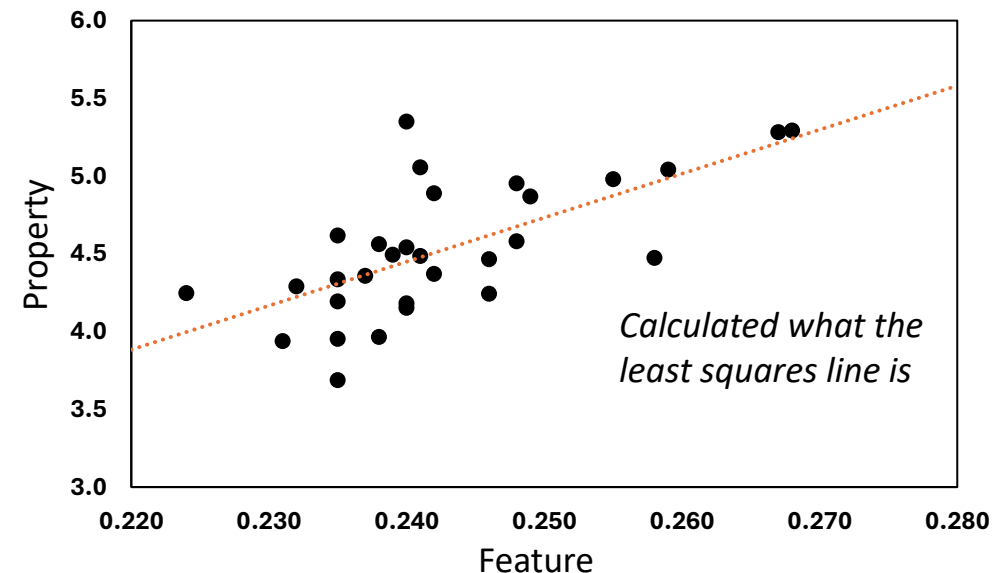
$$m = 28.29$$

$$b = (135.83 - 28.29 * 7.28) / 30$$

$$b = -2.34$$

Least squares equation :

$$\text{Property} = -2.34 + 28.29 * \text{Feature}$$



Mean squared error (MSE) :

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

Root mean squared error (RMSE) : $RMSE = \sqrt{MSE}$.

Note, this has the same form as a distance equation (ie. $\sqrt{(x-x_i)^2}$)....
So we can think of it as basically the collective distance (although scaled by number of samples) between actual and predicted.

$$R^2 = 1 - \frac{SSE}{TSS}$$

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

Sum of squared errors (SSE) : $SSE = \sum_{i=1}^n (y_i - \hat{y}_i)^2$,

$$SSE = n \times MSE$$

Total sum of squares (TSS) : $TSS = \sum_{i=1}^n (y_i - \bar{y})^2$, and $\bar{y} = \frac{1}{n} \sum_{i=1}^n y_i$.

And thus :

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

Cross Validation (CV)

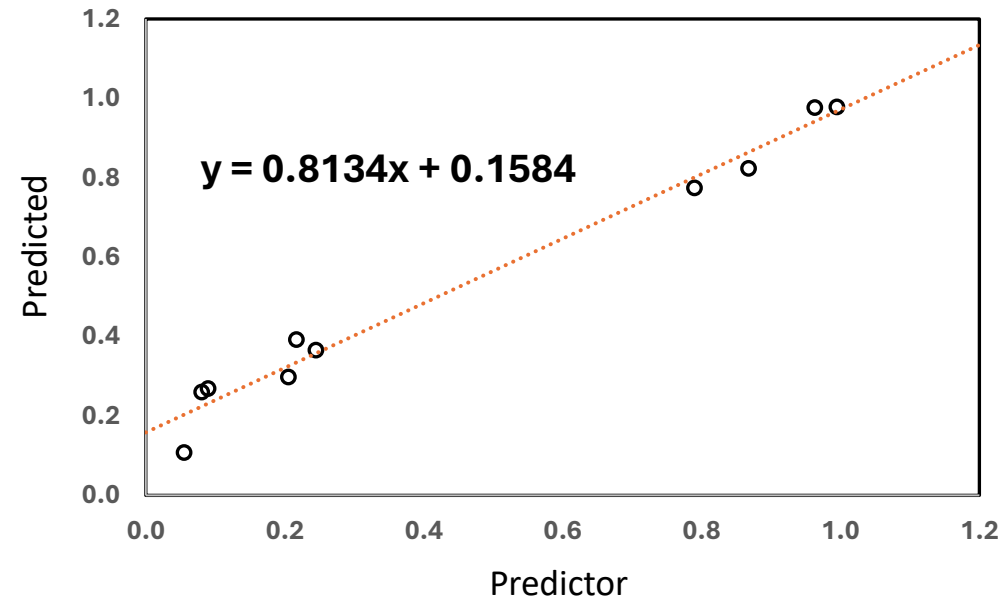
- We have built a model in this example trained on all of the available data. However, does it predict things we don't know with any accuracy?
- There are many ways for cross validation. But for now, let's follow the strategy of (i) randomly removing 20% of the data, (ii) train our model on the other 80% of the data, (iii) measure RMSE training from that, (iv) apply the model to the other 20% of the data and predict the property value, (v) measure RMSE CV between predicted and actual. Good practice would be to repeat this multiple times to ensure that the selection of training / test doesn't have an impact.
- If we have significantly higher value of RMSE testing than RMSE training, then our model is not good for predicting things we don't know. Either need more data, or the descriptors aren't sufficient and the model is overfit, or we need different descriptors, etc.

Calculation of Model Accuracy

$$y_i$$

Predictor	Predicted
0.055	0.108
0.080	0.261
0.090	0.269
0.205	0.298
0.217	0.393
0.245	0.366
0.789	0.774
0.867	0.823
0.963	0.977
0.994	0.979

$$\bar{y} = 0.525$$



$$\hat{y}_i$$

From Model:
0.203
0.224
0.231
0.325
0.335
0.358
0.801
0.864
0.942
0.967

Consider sensitivity to outlier.

Calculation of Model Accuracy

 y_i
 $\hat{y}_i$

Predictor	Predicted	From Model:
0.055	0.108	0.203
0.080	0.261	0.224
0.090	0.269	0.231
0.205	0.298	0.325
0.217	0.393	0.335
0.245	0.366	0.358
0.789	0.774	0.801
0.867	0.823	0.864
0.963	0.977	0.942
0.994	0.979	0.967

 $\bar{y} = 0.525$

Predicted	From Model:	SSE	TSS
0.108	0.203	0.009	0.103513
0.261	0.224	0.001	0.090646
0.269	0.231	0.001	0.08618
0.298	0.325	0.001	0.0399
0.393	0.335	0.003	0.036239
0.366	0.358	0.000	0.028007
0.774	0.801	0.001	0.076004
0.823	0.864	0.002	0.114979
0.977	0.942	0.001	0.173629
0.979	0.967	0.000	0.195736
		0.020	0.945

$$R^2 = 1 - \frac{SSE}{TSS}$$

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

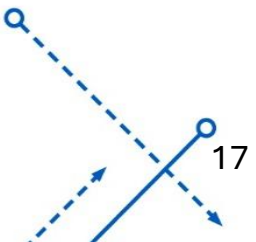
$$TSS = \sum_{i=1}^n (y_i - \bar{y})^2$$

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

$$R^2 = 1 - 0.02/0.945 = 97.9\%$$

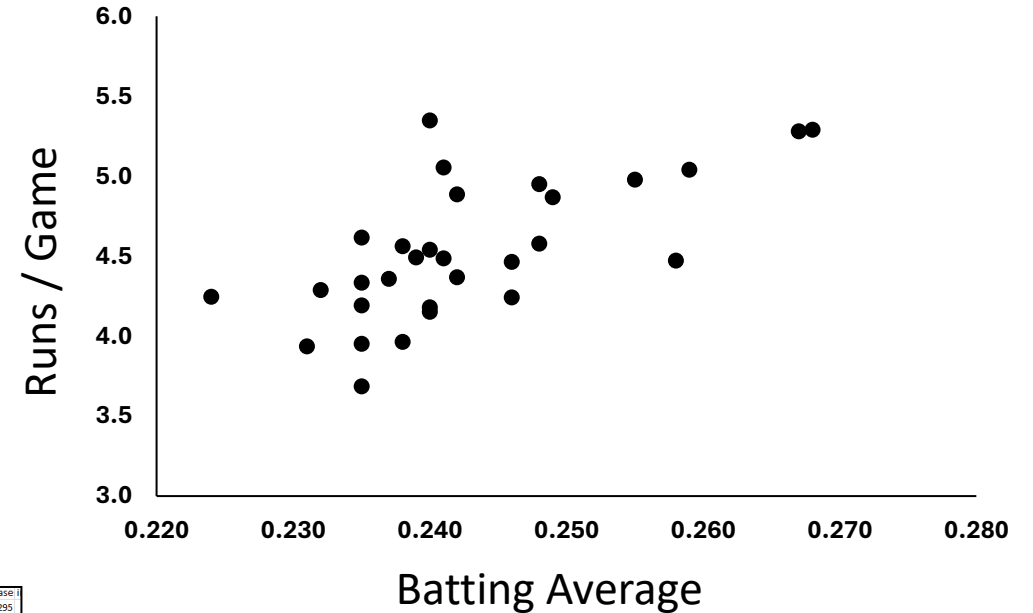
$$MSE = 0.02 / 10 = 0.002$$

$$RMSE = \sqrt{MSE} = 0.044$$



Multiple Linear Regression (MLR)

- We'll use some data on baseball statistics. We'll have our key criteria (target property) as runs / game
- As a starting point, we see that batting average is positively correlated to runs/game, although insufficient.

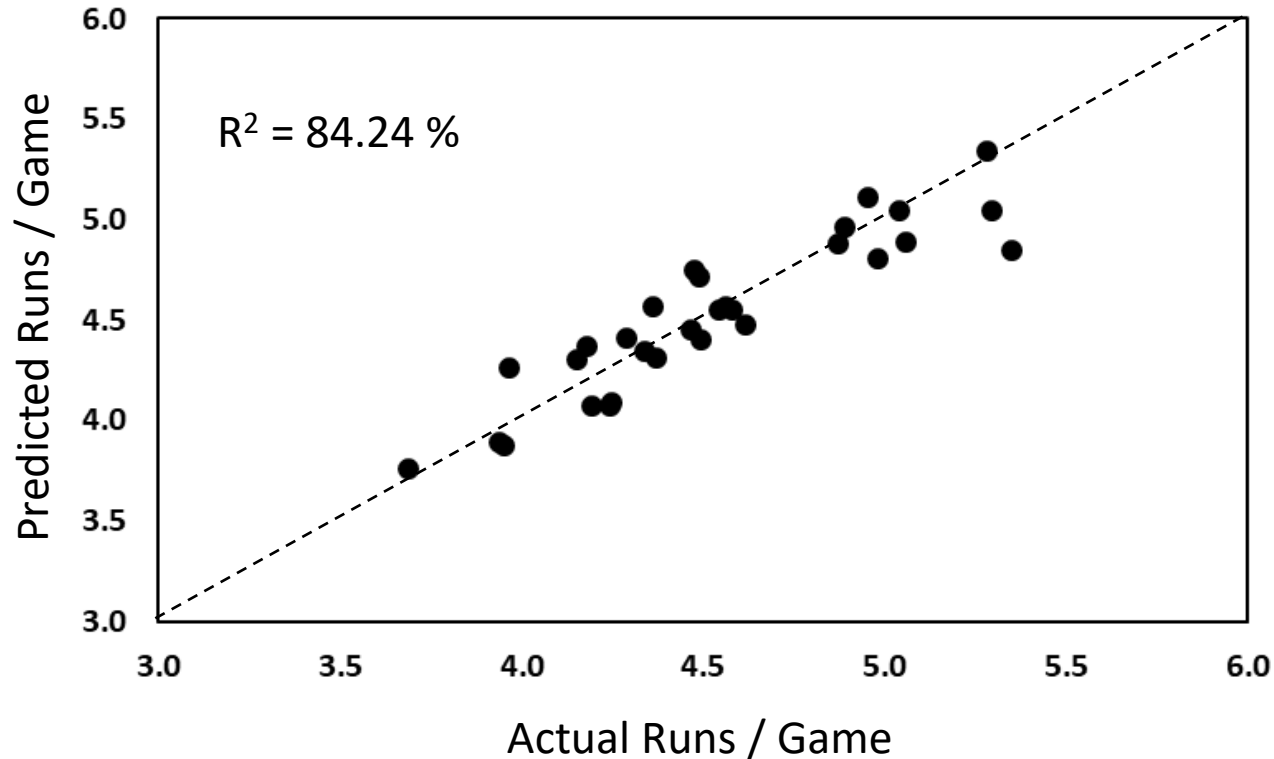


	Runs / Game	At Bats	Runs	Hits	Doubles	Triples	Home Run	Runs Batted	Strike Out	Stolen	Bases Caught	Stealing	Batting Av	On Base P	Slugging P	On Base P	Plate Appt	Hit by Pitch	Sacrifice B	Sacrifice F	Double P	Ground out	Total		
Toronto Blue Jays	5.28	146	4935	771	1319	251	13	233	743	453	1108	72	20	0.267	0.333	0.465	0.798	5481	48	10	33	105	1189	0.87	2295
Houston Astros	5.29	144	4962	762	1332	273	12	186	733	507	1087	48	14	0.268	0.34	0.441	0.781	5583	56	6	50	127	1209	0.87	2187
Boston Red Sox	5.04	148	5008	746	1298	300	21	198	707	462	1276	39	21	0.259	0.326	0.446	0.772	5576	55	10	39	86	1197	0.93	2234
San Francisco Giants	4.95	145	4888	718	1211	241	24	218	686	541	1326	61	12	0.248	0.328	0.441	0.769	5536	53	30	23	107	1157	0.93	2154
Chicago White Sox	4.98	144	4762	717	1214	237	21	171	681	518	1240	54	19	0.255	0.335	0.421	0.756	5406	71	20	35	119	1244	1.11	2006
Cincinnati Reds	4.87	145	4859	706	1209	260	11	157	678	492	1289	27	22	0.249	0.327	0.428	0.756	5512	94	30	36	117	1247	1.06	2082
Atlanta Braves	4.89	143	4746	699	1149	232	18	215	672	501	1275	55	18	0.242	0.32	0.434	0.754	5374	60	28	38	71	1147	0.93	2062
Washington Nationals	4.47	146	4823	653	1245	237	20	167	619	493	1168	53	25	0.258	0.335	0.419	0.754	5463	78	38	29	141	1399	1.3	2023
Los Angeles Dodgers	5.05	146	4903	738	1181	226	22	206	709	558	1289	56	14	0.241	0.328	0.422	0.75	5628	98	29	40	82	1157	0.86	2069
Tampa Bay Rays	5.35	146	4986	781	1198	260	30	203	736	517	1409	77	35	0.24	0.318	0.427	0.744	5607	63	6	34	61	1157	0.92	2127
Minnesota Twins	4.49	146	4899	655	1180	242	16	205	619	480	1265	46	13	0.241	0.314	0.422	0.737	5485	61	7	35	111	1246	1	2069
Colorado Rockies	4.58	145	4808	664	1192	242	32	167	635	438	1199	70	21	0.248	0.315	0.416	0.731	5376	52	44	34	89	1307	1.1	1999
New York Yankees	4.36	145	4770	632	1130	189	11	194	591	568	1334	58	13	0.237	0.323	0.403	0.726	5436	56	10	31	137	1184	1.02	1923
Philadelphia Phillies	4.54	144	4773	654	1144	228	23	177	624	510	1245	71	18	0.24	0.318	0.408	0.726	5417	55	41	36	95	1280	1.08	1949
Oakland Athletics	4.56	144	4785	657	1138	247	18	179	617	478	1204	82	15	0.238	0.316	0.409	0.725	5407	86	15	43	91	1202	0.93	1958
Los Angeles Angels	4.47	144	4835	643	1191	241	19	175	612	396	1259	69	23	0.246	0.309	0.413	0.722	5327	52	24	19	93	1193	0.97	1995
Milwaukee Brewers	4.62	146	4847	674	1141	232	17	176	639	524	1333	71	20	0.235	0.319	0.399	0.718	5515	84	24	33	88	1227	1.02	1935
San Diego Padres	4.49	144	4762	647	1137	230	19	161	615	532	1190	102	34	0.239	0.32	0.396	0.717	5422	57	28	38	108	1306	1.09	1888
Baltimore Orioles	4.18	144	4828	602	1160	243	14	178	577	406	1273	51	20	0.24	0.305	0.407	0.712	5334	56	13	29	92	1150	0.89	1965
Detroit Tigers	4.37	146	4849	638	1172	211	34	167	616	440	1383	72	23	0.242	0.307	0.403	0.709	5386	35	16	44	96	1137	0.93	1952
St. Louis Cardinals	4.15	144	4768	598	1143	227	20	163	577	430	1191	76	21	0.24	0.311	0.398	0.709	5343	76	33	34	85	1170	0.88	1899
Chicago Cubs	4.29	145	4746	622	1102	196	24	190	592	430	1428	64	33	0.232	0.305	0.404	0.709	5316	79	34	25	113	1170	1.06	1916
New York Mets	3.97	146	4686	579	1117	204	12	161	549	456	1245	48	24	0.238	0.316	0.39	0.706	5383	88	31	22	109	1202	1.02	1828
Cleveland Indians	4.34	143	4711	620	1106	217	20	180	596	413	1229	96	13	0.235	0.302	0.404	0.706	5327	53	19	29	92	1221	1.01	1903
Kansas City Royals	4.24	144	4833	611	1189	211	24	151	577	374	1140	109	28	0.246	0.304	0.393	0.697	5336	48	29	45	90	1271	0.97	1901
Arizona Diamondbacks	4.19	145	4913	608	1157	272	28	129	578	484	1313	36	12	0.235	0.309	0.381	0.69	5506	51	30	24	87	1212	0.94	1872
Seattle Mariners	4.25	146	4813	620	1077	200	8	181	597	483	1371	59	21	0.224	0.301	0.381	0.682	5399	60	8	33	80	1116	0.87	1836
Miami Marlins	3.95	146	4843	577	1138	211	20	143	551	411	1408	101	25	0.235	0.301	0.375	0.677	5378	61	29	28	82	1224	1.08	1818
Texas Rangers	3.94	144	4810	567	1113	202	20	155	541	387	1236	88	25	0.231	0.295	0.378	0.673	5297	57	15	28	100	1285	1.05	1820
Pittsburgh Pirates	3.69	144	4745	531	1115	218	25	113	499	458	1170	49	23	0.235	0.307	0.363	0.67	5305	45	26	30	84	1283	1.04	1722

Data taken from mlb.com

[30 x 26]

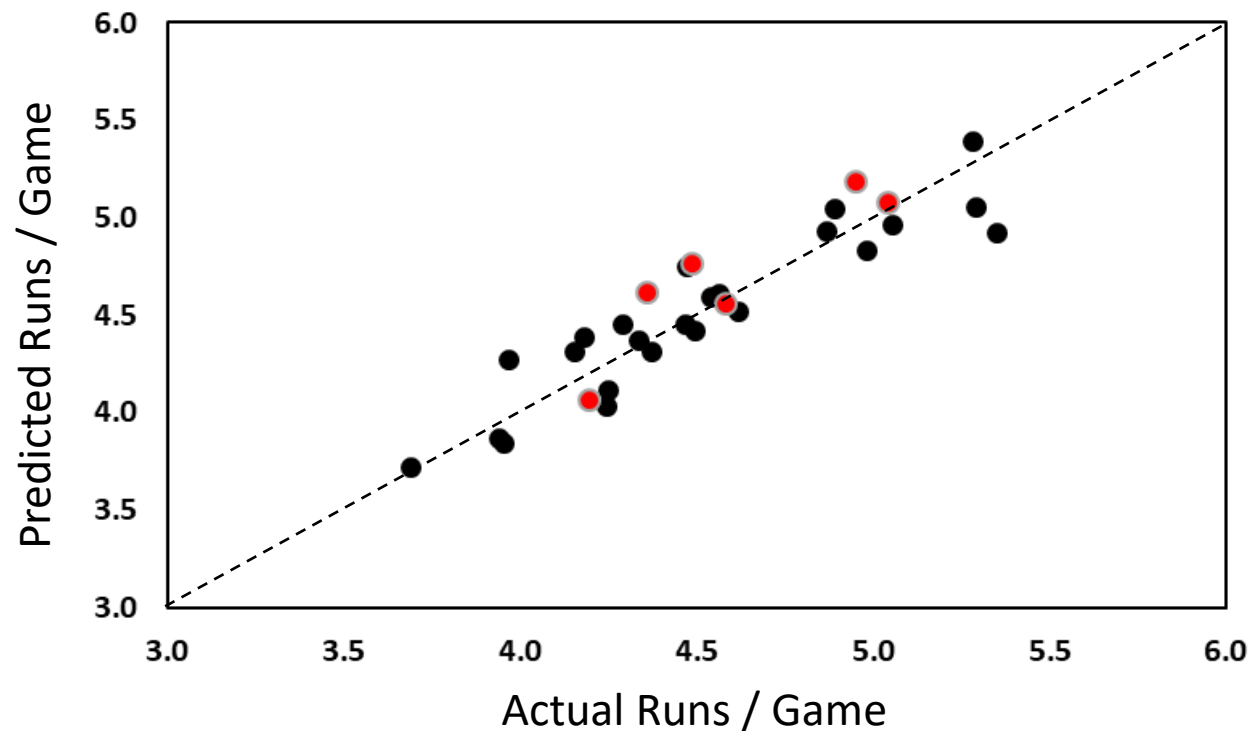
In MLR we are not assuming any correlation between features. Also, consider role of outliers.



We have added in two additional features : On-base percentage and Slugging percentage (So now three terms).

$$\text{Runs / Game} = -10.83 * \text{Batting Avg} + 12.58 * \text{On-base pct} + 15.70 * \text{Slugging pct} - 3.25$$

- Significant improvement in model (which it should have, as there are more terms to fit to ; addition of terms should not decrease accuracy).
- Added terms leads to better chance for over-fitting though. Let's test.

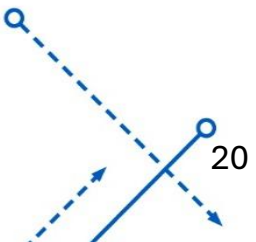


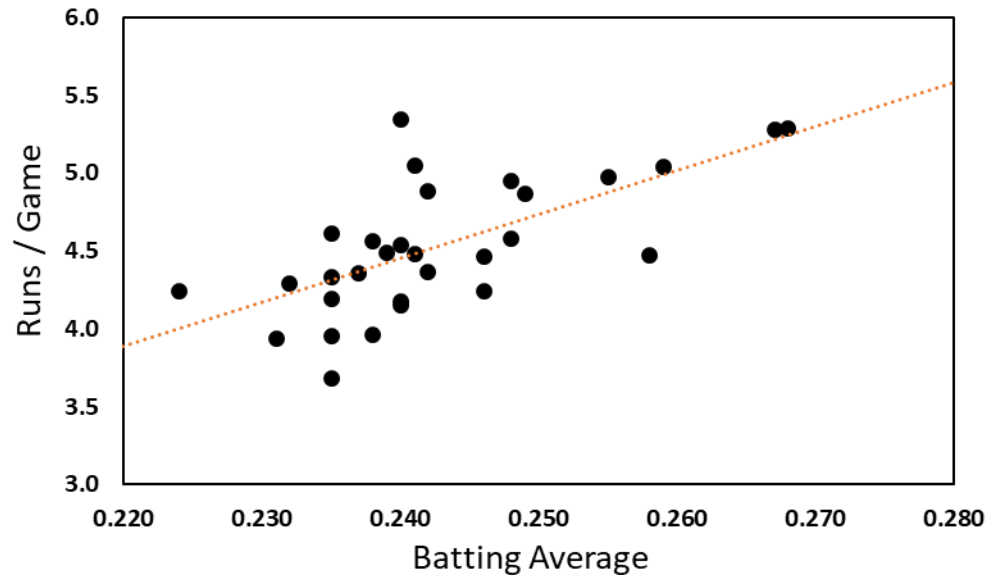
$$\text{Runs / Game} = -15.34 * \text{Batting Avg} + 14.30 * \text{On-base pct} + 17.54 * \text{Slugging pct} - 3.43$$

$$\text{From before : Runs / Game} = -10.83 * \text{Batting Avg} + 12.58 * \text{On-base pct} + 15.70 * \text{Slugging pct} - 3.25 \dots$$

So similar, which is what we want.

Training and testing accuracy are similar, which suggests that the model is good for predicting also (at least in terms of interpolation)





$$\text{Runs / Game} = -10.83 * \text{Batting Avg} + 12.58 * \text{On-base pct} + 15.70 * \text{Slugging pct} - 3.25$$

- We can see an issue with this..... Batting average and runs/game are negatively correlated in the model. But we know there is some correlation.... Model based on fitting and not real relationships.
- MLR operates as if the terms are all independent (ie. uncorrelated). Basically, treating all terms except for one as constant, and adjusting that parameter, and then fixing it and adjusting to next parameter and so on, until accuracy is maximized
- Clearly though we have correlation in input.

Principal Component Regression (PCR)

First, do PCA on batting average, on-base pct. and slugging pct.

(In classification case, wouldn't do PCA on such small data, but let's see if there is any benefit for predictive situation)

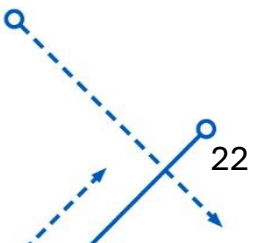
- (1) Normalize data (so that we aren't modeling on background – ie. only signal impacting model) --- this step isn't necessary in MLR (and isn't needed in PCR if we use all PCs).
- (2) Select number of PCs
- (3) Extract scores matrix (data of form of team vs. PCs)
- (4) Build multiple linear regression between runs/game and scores values

PC1 : 84.73%

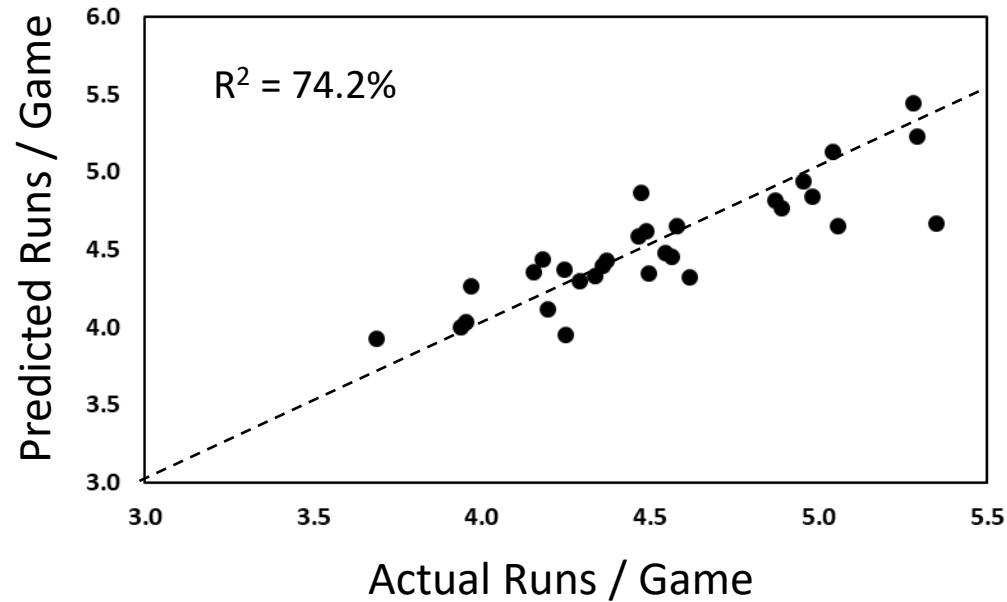
PC2 : 8.42%

PC3 : 6.86%

PCs are all orthogonal to each other ie. no correlations between them



PCR Result



If we select 2 PCs (as captures over 90% of variance and also has elbow in scree plot) we get this result.

$$\text{Runs / Game} = -.228 * \text{PC1} + 0.147 * \text{PC2} + 4.53$$

- Note: Accuracy is worse than MLR model (ie. have gone from R^2 of 84.2% to 74.2%).
- So clearly this is worse, right ? Maybe not.
- Note, makes sense it would be lower R^2 because we have included less information (ie. left out 6.86%)

What are the PCs capturing? Let's look at the loadings matrix (remember that batting avg, on-base pct and slugging pct refer to the normalized values):

$$PC1 = -0.583 * \text{batting avg} + 0.013 * \text{on-base pct} - 0.812 * \text{slugging pct}$$

$$PC2 = -0.574 * \text{batting avg} - 0.714 * \text{on-base pct} + 0.401 * \text{slugging pct}$$

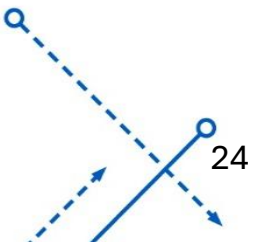
$$\text{Runs / Game} = -.228 * PC1 + 0.147 * PC2 + 4.53$$

So we can now estimate the relative weights on the model of the terms by plugging the PC equations into the model equation
(eg. $-0.583 * -.228 - 0.574 * 0.147$ for batting average)

Relative weights on runs/game:

Batting avg : 0.135 ; On-base pct : 0.026 ; Slugging pct : 0.234

Makes more sense, even with lower R^2 !!



Our target property is Runs/Game, and our features are batting average, on-base percentage and slugging percentage.

$$\text{Runs / Game} = -0.228 * \text{PC1} + 0.147 * \text{PC2} + 4.53$$

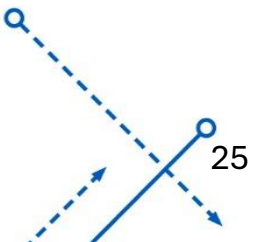
$$\text{PC1} = -0.583 * \text{batting avg} - 0.574 * \text{on-base pct} - 0.575 * \text{slugging pct}$$

$$\text{PC2} = 0.013 * \text{batting avg} - 0.714 * \text{on-base pct} + 0.700 * \text{slugging pct}$$

$$\text{Runs / Game} = -.228 * (-0.583 * \text{batting avg} - 0.574 * \text{on-base pct} - 0.575 * \text{slugging pct}) + 0.147 * (0.013 * \text{batting avg} - 0.714 * \text{on-base pct} + 0.700 * \text{slugging pct}) + 4.55$$

$$\underline{\text{Runs / Game} = 0.135 * \text{BA} + 0.026 * \text{OBP} + 0.234 * \text{SLG} + 4.55}$$

But still standardized feature input



Final PCR Model Equation

$$\text{Runs / Game} = 0.135 * \text{BA} + 0.026 * \text{OBP} + 0.234 * \text{SLG} + 4.55$$

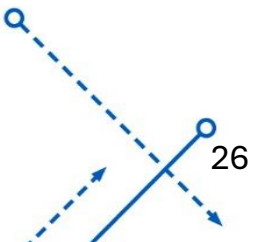
Batting avg : mean .243, stdev .0102

On-base pct : mean .316, stdev .0117

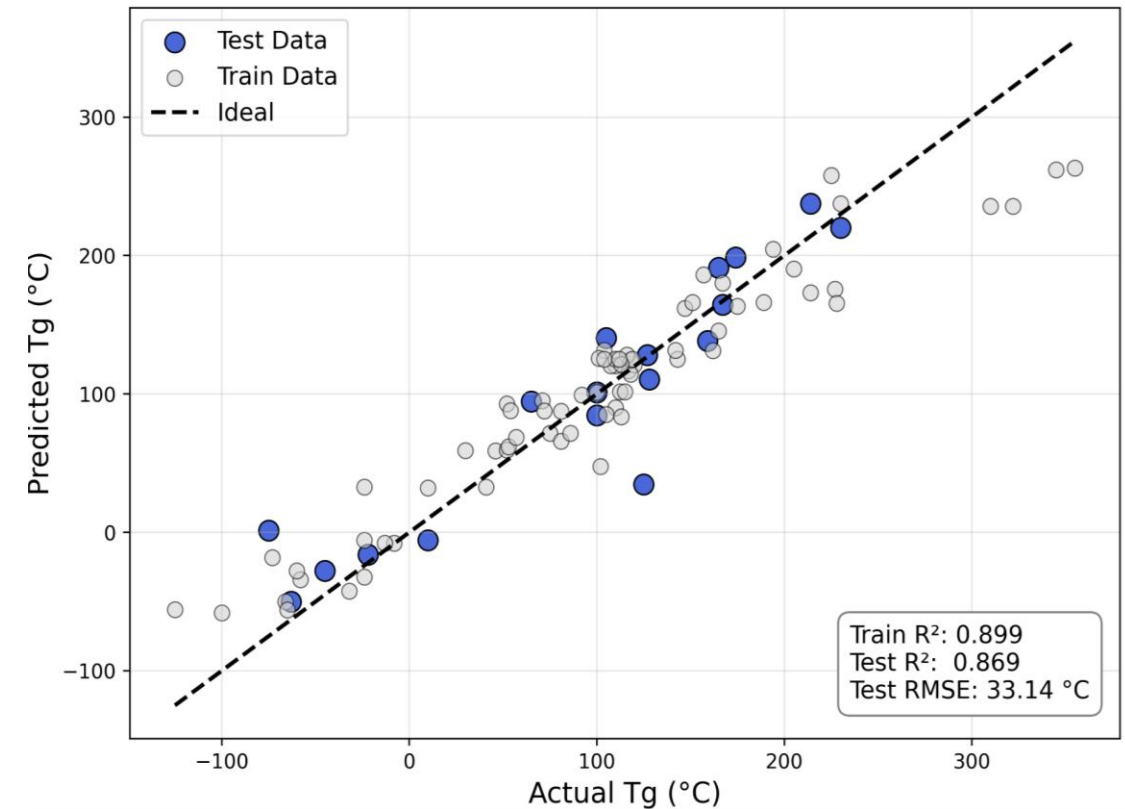
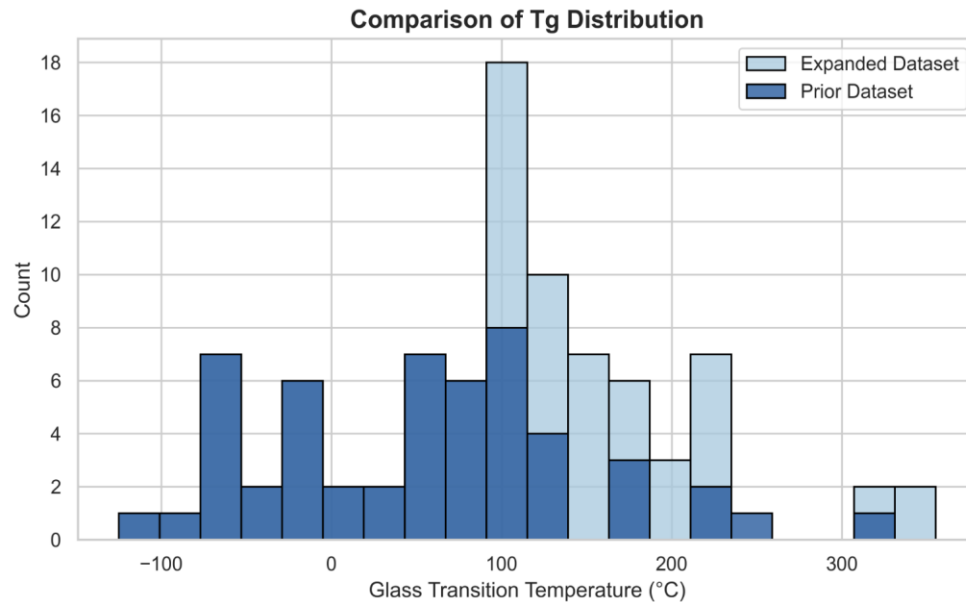
Slugging pct : mean .4096, stdev .0231

$$\text{Runs / Game} = 0.135 * ((\text{BA} - .243) / .0102) + 0.026 * ((\text{OBP} - .316) / .0117) + 0.234 * ((\text{SLG} - .4096) / .0231) + 4.55$$

$$\text{Runs / Game} = 13.24 * \text{BA} + 2.22 * \text{OBP} + 10.13 * \text{SLG} - 3.52$$



Developing Model for Tg from Topological Descriptors : Importance of Data Distribution

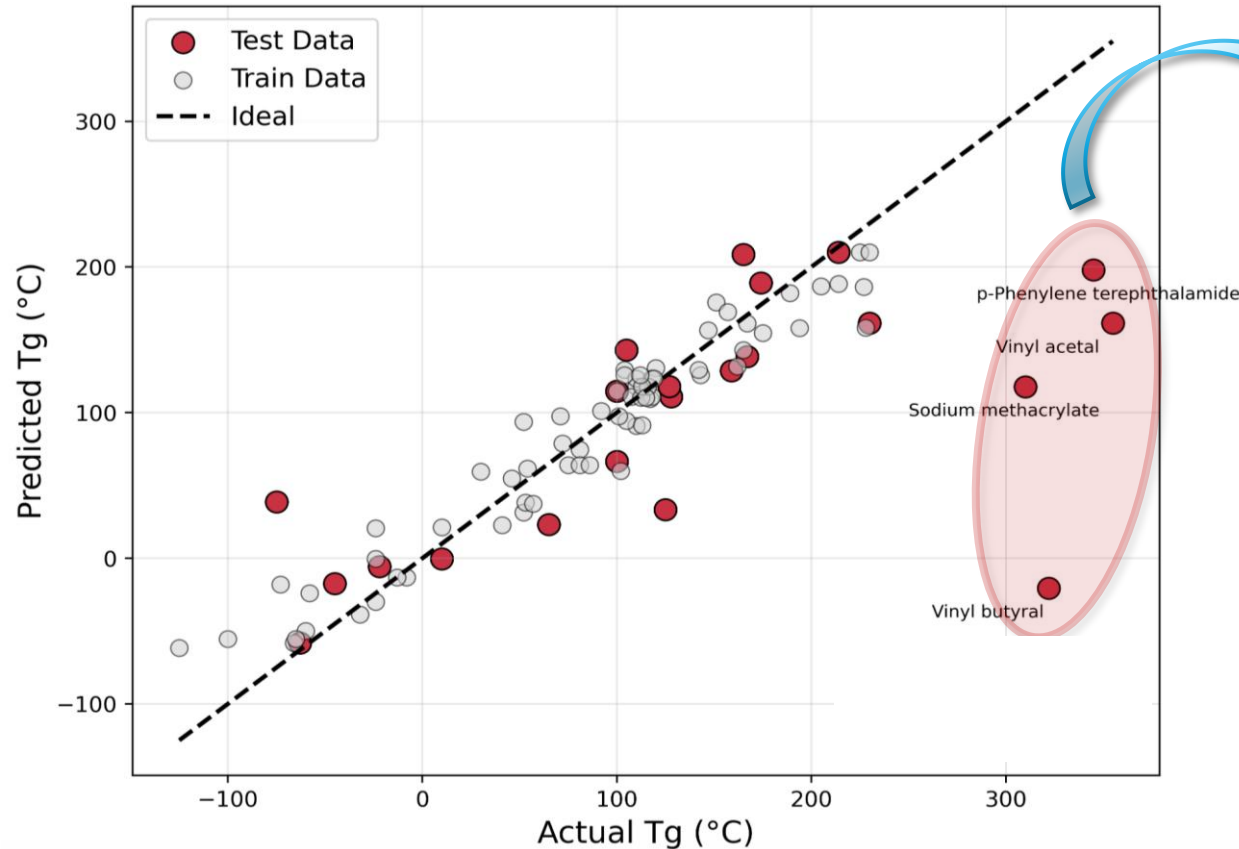


Using initial dataset, had model (not shown) with Training $R^2 = 0.883$, but a test R^2 of 0.343

Added targeted data for more normal distribution.
Training $R^2 = 0.899$ and test R^2 of 0.869

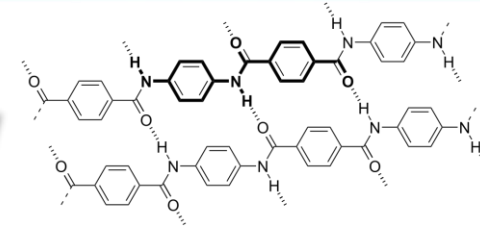
- Significance of features consistent with the correlation analyses

Model Interpretation: Insights into High-Tg Polymers



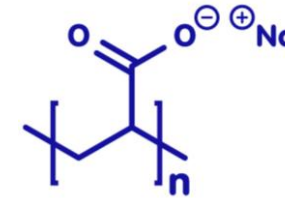
Several high-Tg polymers are systematically **under-predicted** by the model.

Identify information necessary for high Tg design



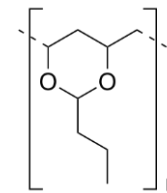
Poly (P-Phenylene terephthalamide)

Strong **interchain hydrogen bonding**, **aromatic stacking interactions** and **conjugated backbone**, leading to high Tg not fully represented by the descriptor set.



Poly (sodium methacrylate)

Contains **ionic side groups** that enhance intermolecular interactions and thermal resistance, which are not reflected in the current topological features.



Poly (Vinyl acetal / Vinyl butyral)

Contain acetal (**1,3-dioxane-like**) ring structures that introduce **local conformational constraints**, which are not fully captured by current topological descriptors.

Regularization Approaches

- Regularization tunes the function by adding an additional penalty term in the error function. This term penalizes large weights and thereby avoids the impact from outliers.
- This is also referred to as a shrinkage method.
- To this point, the goal of our regression has been to minimize the sum of squared residuals (RSS): $RSS = \sum (y_i - \hat{y}_i)^2$ to have the actual values as close as we can to the predicted values.
- With regularization approaches, we instead seek to minimize the following :

$$RSS + \lambda \sum \beta_j^2$$

β is the coefficients of our regression model (ie. $Y = \beta_i X_i + b$), and λ is the shrinkage penalty. We will select λ to minimize RMSE. If $\lambda = 0$, then it is same as ordinary least squares.

Ridge, LASSO and Elastic Nets

$$\text{Loss} = \text{RSS} + \lambda \sum \beta_j^2$$

This equation is for **Ridge regression**, where we penalize the model by square of coefficient values.

$$\text{Loss} = \text{RSS} + \lambda \sum |\beta_j|$$

This equation is for **LASSO** (Least Absolute Shrinkage and Selection Operator) regression. This instead penalizes the model by the absolute values of the coefficient values.

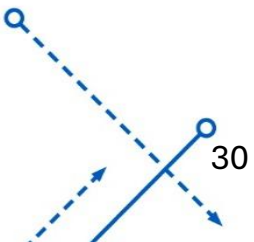
Or we can combine the two. This is the **Elastic Nets** regression approach :

$$\text{Loss} = \text{RSS} + \lambda(((1 - \alpha)/2) \sum \beta_j^2 + \alpha \sum |\beta_j|)$$

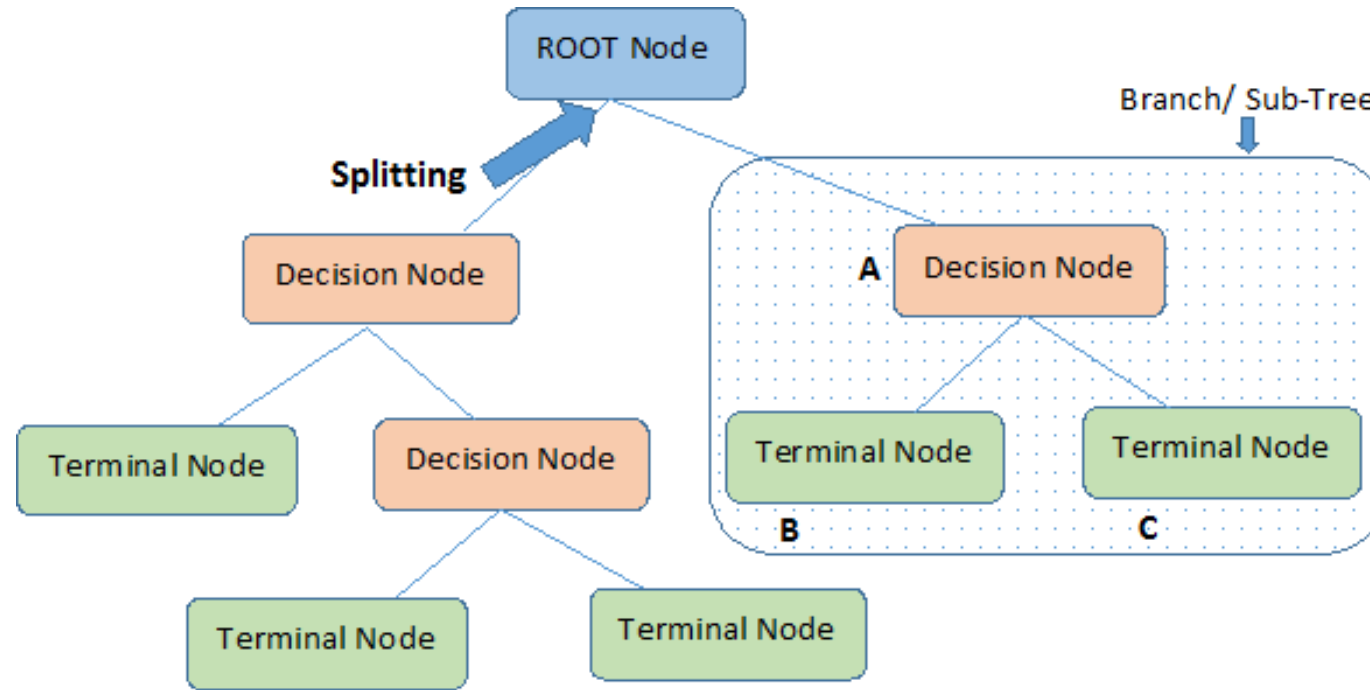
where α is the mixing parameter between ridge ($\alpha = 0$) and lasso ($\alpha = 1$).

So we have two parameters to tune :

λ for all three regression approaches, and α for elastic nets.



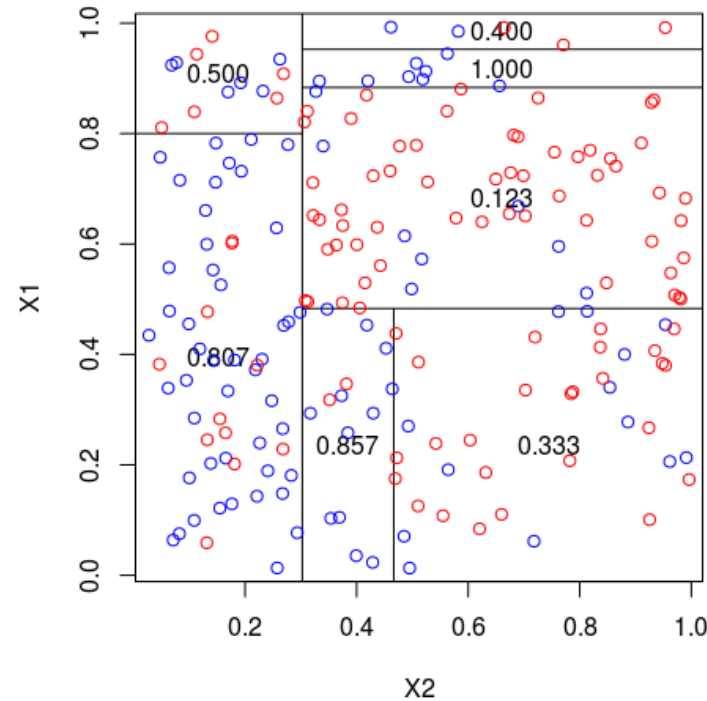
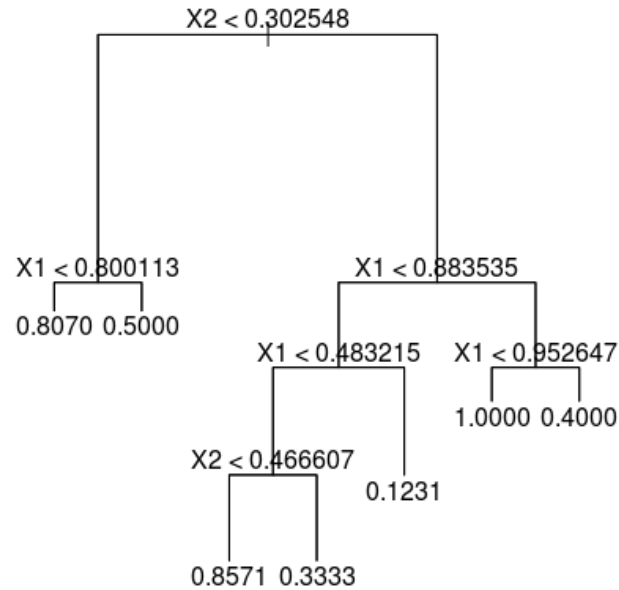
Decision Trees



Note:- A is parent node of B and C.

- The nodes are based on maximizing information. A goal is to minimize the number of nodes (same idea as with regressions where we want to avoid over-fitting).
- In order to work for now, final property or response needs to be categorical.
- A downside is that it is dependent on the data, and therefore provides result based on that but not necessarily universal rule.

Decision Trees



- Alternate visualizations : If we have a binary classification, it can be viewed as specific cut-offs within a 2D map. Of course, this becomes challenging when we go beyond 2D to a high-dimensional descriptor space.
- Also can see how this corresponds potentially with over-fitting. That is, if we have more and more lines, we can get higher accuracy, but without much meaning.

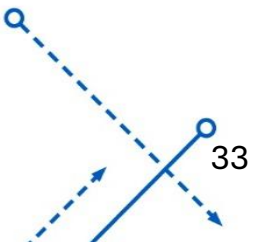
Classification Criteria

We will discuss here Shannon entropy criterion for our example. There are multiple criteria, which we will discuss more later, but this is one which seeks to maximize information cut-off to maximize separations of classes.

$$Entropy(attribute) = H = -\sum_{i=1}^m p_i \log_2(p_i)$$

Entropy is the measure of randomness (impurity). An attribute (or feature or descriptor) which minimizes the entropy reflects the least randomness or impurity in the partition.

p_i Probability that an arbitrary attribute belongs to class (C_i)
 m # of distinct classes (in the present case $m=2$)



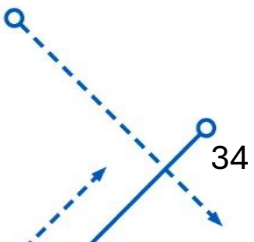
$$Entropy(attribute) = H = -\sum_{i=1}^m p_i \log_2(p_i)$$

For the data we'll use for this example, we have a total of 355 compounds and 227 forms perovskite and 128 do not form perovskite.

$$p(yes) = \frac{227}{355} \quad p(no) = \frac{128}{355}$$

Therefore Entropy (in bits) would be

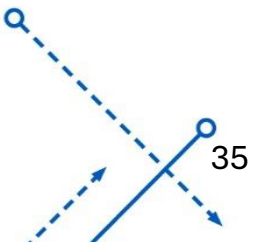
$$H(class) = -\frac{227}{355} \log_2\left(\frac{227}{355}\right) - \frac{128}{355} \log_2\left(\frac{128}{355}\right)$$



Consider an attribute now. If we have an attribute A with 3 distinct values (say low, medium and high), the entropy of this attribute should be calculated as the second step.

Consider we have out of 355 compounds, 100 with *low*, 150 with *medium* and 105 with *high* value for the attribute A. Also consider, the *low* category in turn has 45 ABO₃ compounds, *medium* has 100 compounds and the *high* has about 75 compounds forming perovskite, then the entropy of attribute A would be,

$$\begin{aligned} Entropy(A) = & \frac{100}{355} \left\{ -\frac{45}{100} \log_2 \left(\frac{45}{100} \right) - -\frac{55}{100} \log_2 \left(\frac{55}{100} \right) \right\} \\ & + \frac{150}{355} \left\{ -\frac{100}{150} \log_2 \left(\frac{100}{150} \right) - -\frac{50}{150} \log_2 \left(\frac{50}{150} \right) \right\} \\ & + \frac{105}{355} \left\{ -\frac{75}{105} \log_2 \left(\frac{75}{105} \right) - -\frac{30}{105} \log_2 \left(\frac{30}{105} \right) \right\} \end{aligned}$$



In order to measure the gain (worth) of attribute in splitting the data into pure classes, the entropy of A is subtracted from the entropy of the class attribute.

$$Gain(A) = Entropy(class) - Entropy(A)$$

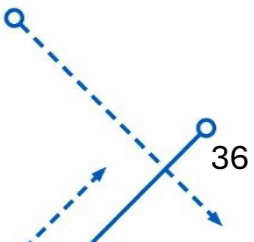
“Best classification” is obtained from an attribute which has the highest Gain value.

Information gain is unstable in some cases so, to counter that Gain ratio is used as a splitting criterion.

$$Gainratio(A) = \frac{Gain(A)}{SplitInfo(A)}$$

By considering the same attribute A, the SplitInfo can be calculated as,

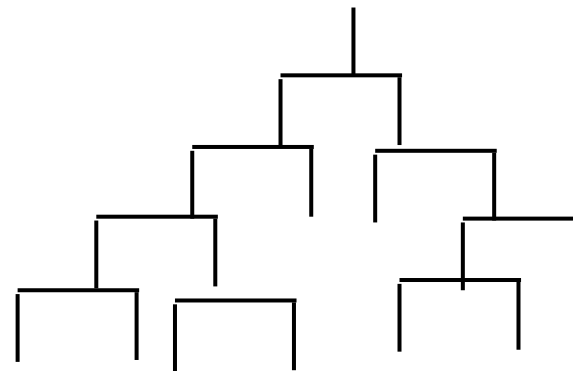
$$SplitInfo(A) = -\frac{100}{355} \log_2 \left(\frac{100}{355} \right) - \frac{150}{355} \log_2 \left(\frac{150}{355} \right) - \frac{105}{355} \log_2 \left(\frac{105}{355} \right)$$



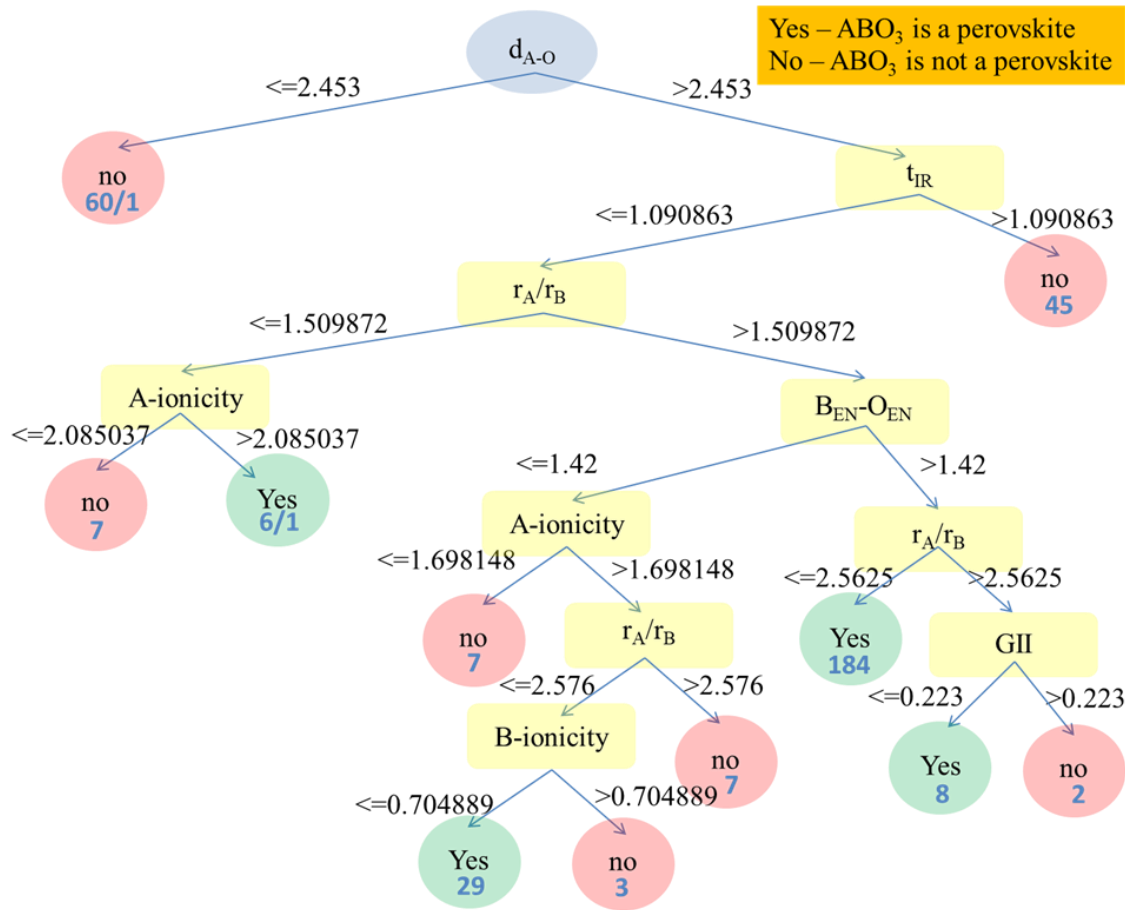
- This example had an attribute with three classes. But assume we have many potential values for many attributes... this becomes a more complex challenge as there are nearly unlimited possibilities.
- This is the logic though i.e. go through all of the potential possibilities, and determine the particular attribute and the corresponding attribute value which maximizes the gain ratio.

$$\text{Gainratio}(A) = \frac{\text{Gain}(A)}{\text{SplitInfo}(A)}$$

- Once the first node is defined, we keep repeating the process on the new subsets in the data.



Example Result



Model evaluation: 10-fold cross validation

Accuracy: 94%

Post-pruning operation (this is done to avoid over-fitting the data) – effectively this means we remove lower 'branches' and track how much the results change

Adjustable Parameters (potential source of bias)

-minsplit: Set the minimum number of observations in the node before the algorithm perform a split

-minbucket: Set the minimum number of observations in the final node i.e. the leaf

-maxdepth: Set the maximum depth of any node of the final tree. The root node is treated a depth 0

Confusion matrix :

		Actual	
		Y	N
Predicted	Y	True Positive (TP)	False Positive (FP)
	N	False Negative (FN)	True Negative (TN)

$$\text{Accuracy} = (TP + TN) / (TP + TN + FP + FN)$$

$$\text{Sensitivity} = TP / (TP + FN)$$

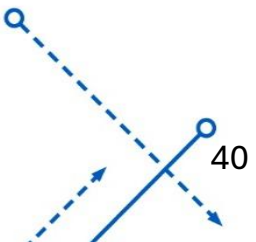
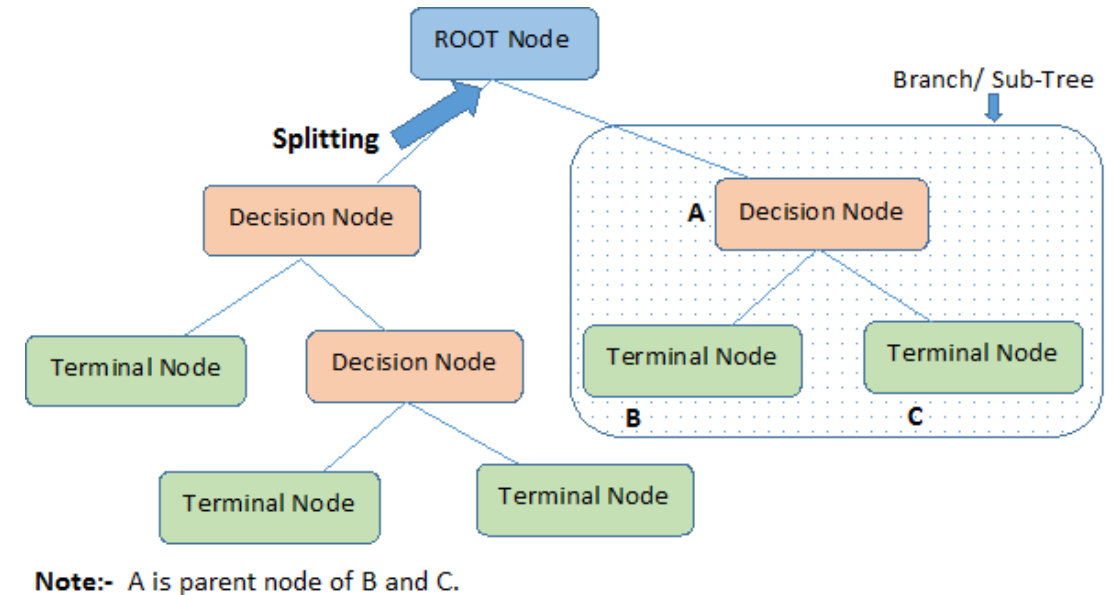
$$\text{Specificity} = TN / (TN + FP)$$

Quantitative Prediction Instead of Class

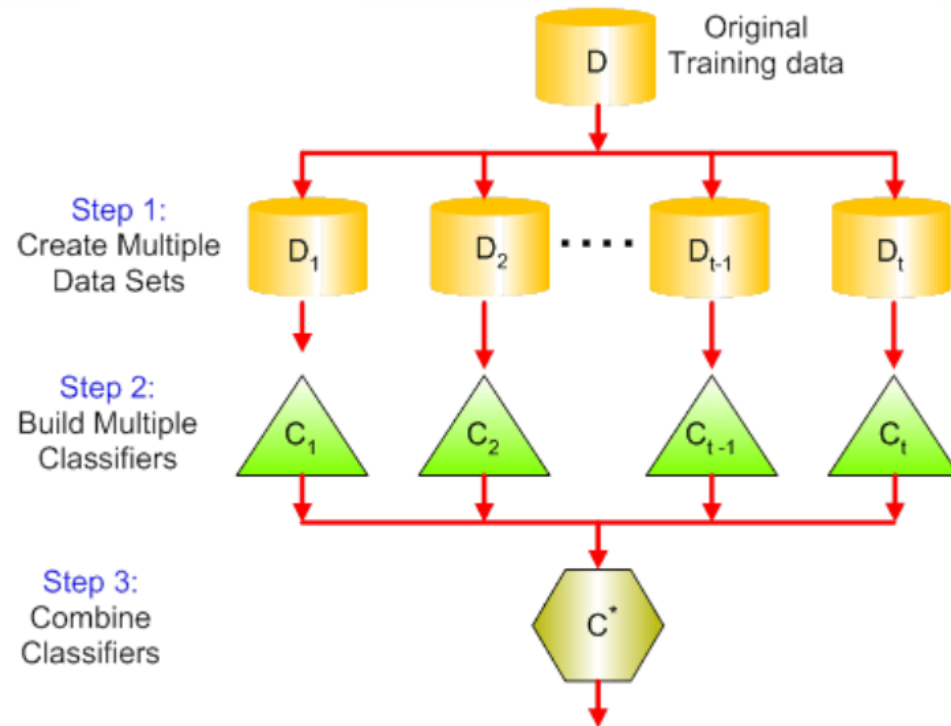
Often used for predicting a value. The objective in this case is instead of saying we want the feature and cutoff which most separates the classes, we instead have the cutoff which results in the least variance within the values of the two classes.

This process is then continually repeated until instead of having a yes or no prediction, we instead have an average value for all remaining entries.

Also, can think about the impact of outliers versus in MLR.



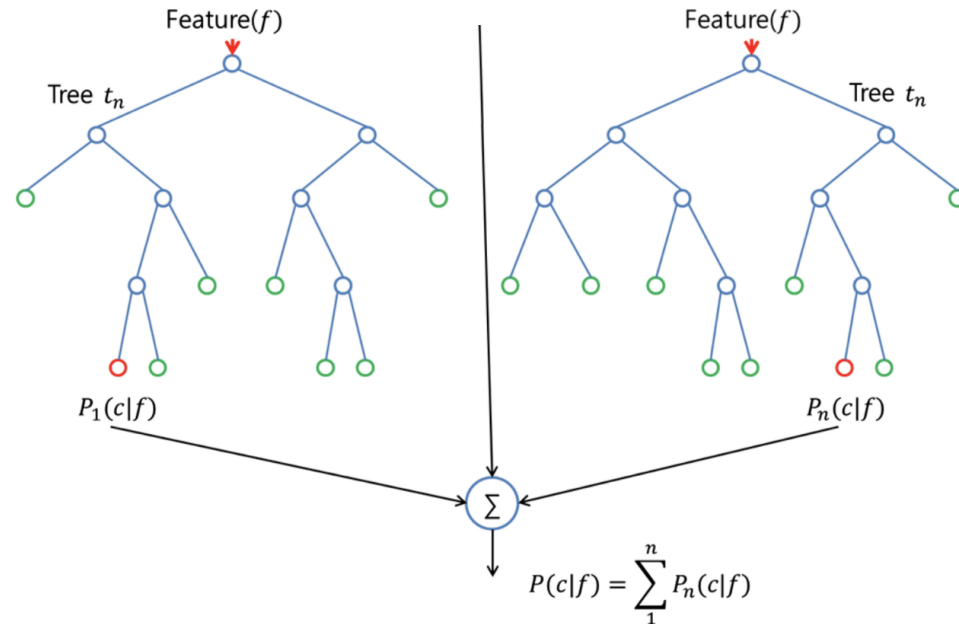
Bagging



Con : It is less interpretable ... that is, since it is the combination of many models, we don't have a single tree diagram anymore, thus losing a benefit of the approach.

- To apply bagging to regression/classification trees, simply construct B regression/classification trees using B bootstrapped training sets, and average the resulting predictions.
- These trees are not pruned, and therefore each individual tree has high variance, but low bias. Averaging these B trees reduces the variance.

Random Forest



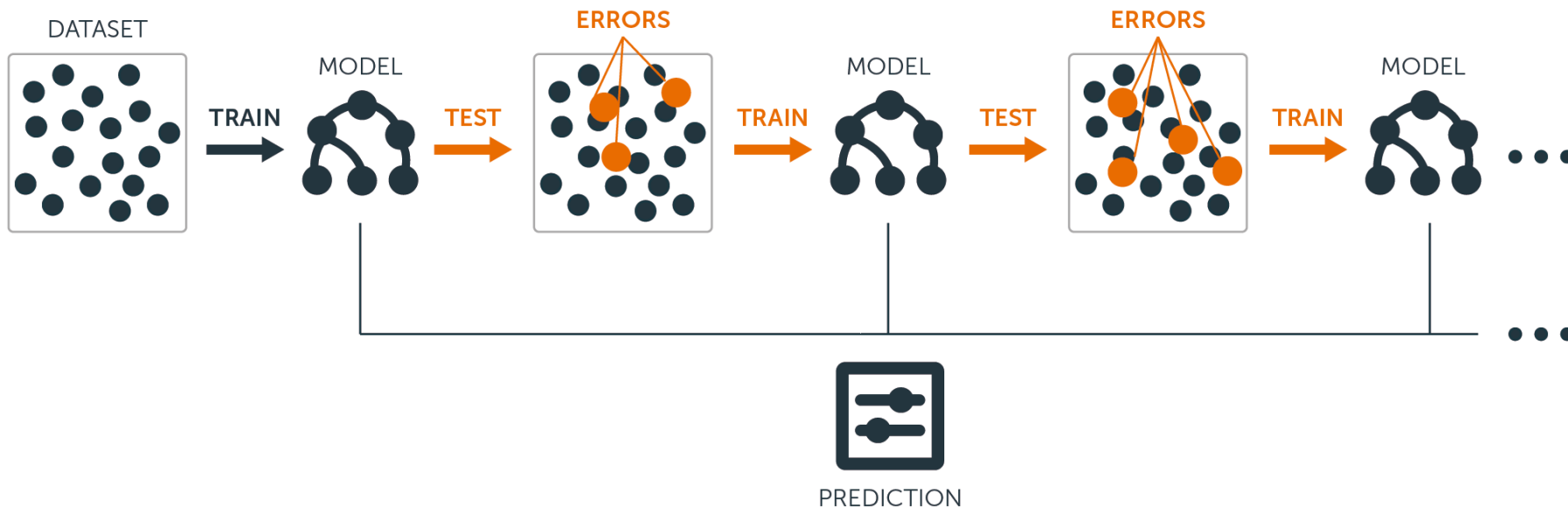
- As in bagging, you build a number of decision trees on bootstrapped training samples.
- But when building these decision trees, each time a split in a tree is considered, a random sample of m predictors is chosen as split candidates from the full set of p predictors.
- The split is allowed to use only one of those m predictors. This is the main difference between random forests and bagging; in bagging, $m = p$.

Parameters to tune : the number of predictors to randomly select at each split, the number of trees, and the minimum leaf node size.

Predicted property value is then the average across all of the trees.

Boosting

- Bagging involved creating multiple copies of the original training dataset and fitting a separate decision tree to each copy, and then combining all of the trees in order to create a single predictive model. Notably, each tree is built on a bootstrapped dataset, independent of the other trees.
- Boosting works in a similar way, except that the trees are grown sequentially: each tree is grown using information from previously grown trees. Boosting does not involve bootstrap sampling; instead, each tree is fitted on a modified version of the original dataset.



Boosting is very useful when you have a lot of data and you expect the decision trees to be very complex.

Feature	Random Forest	XGBoost (Extreme Gradient Boosting)
Ensemble Method	Bagging (Bootstrap Aggregating)	Boosting (Gradient Boosting)
Tree Architecture	Parallel : Trees are grown independently at the same time.	Sequential : Trees are grown one after another, learning from prior mistakes.
Primary Goal	Reduces variance (prevents overfitting).	Reduces bias and variance simultaneously.
Hyperparameter Tuning	Minimal. Works very well out-of-the-box.	High. Requires careful tuning of learning rates, depth, and regularization.
Missing Values	Requires preprocessing or imputation.	Native handling . Automatically learns how to route missing data.
Regularization	None built-in (relies on random sampling).	Built-in L1 (α) and L2 (λ) regularization to control overfitting.

Summary

- Last time we discussed unsupervised learning methods. This time we discussed supervised learning methods.
- In particular : linear regressions, random forest, and boosting.
- Highlight the benefits and negatives to each, including handling different data sizes, the potential for human biasing, over-fitting and interpretability.