**Regression**

**F-statistic**: shows Whether (not how much) the model’s features add to the reduced model, this is confirmed by its P-value. **Found in *fitted\_model.summary()***

* If the feature set in a model is a subset of the feature set of the other, then we can use F-test. The model with higher F statistic is superior to the other one.

**R-squared**:

- Quantifying the performance of a model on the training set.   
- Most common measure of goodness of fit in a linear regression model.   
- A proportion (between 0 and 1) that expresses how much variance in the outcome variable is explained by the explanatory variables in the model.   
- Getting a negative value for R-squared means that your model does very poorly in explaining the target.  
- Generally speaking, higher 𝑅2 values are better to a point. However, a very high 𝑅2 is a warning sign of overfitting.  
- Low 𝑅2 indicates that our model isn't explaining much information about the outcome, which means it will not give very good predictions.   
- No dataset is a perfect representation of reality, so a model that perfectly fits our data (𝑅2 of 1 or close to 1) is likely to be biased by quirks in the data and will perform less well on the test set.  
- Adjusted R-squared does the same job as R-squared, but it is adjusted according to the number of features included in the model.

Example: Model A’s *R*2 is 0.747: This means that model A explains 74.7% of the variance in the target, leaving 25.3% unexplained.

***\*If our model performs significantly worse on the test set compared to the training set, then we should suspect overfitting****.*

**Using information criteria**A common way of comparing different models and selecting the best one.

* Akaike Information Criterion (AIC) : 𝑛𝑙𝑛(𝑆𝑆𝐸)−𝑛𝑙𝑛(𝑛)+2𝑝
* Bayesian Information Criterion (BIC): 𝑛𝑙𝑛(𝑆𝑆𝐸)−𝑛𝑙𝑛(𝑛)+𝑝𝑙𝑛(𝑛)
* 𝑛: sample size, 𝑝: number of regression coefficients in the model (including the constant). 𝑙𝑛: natural logarithm.
* Both take into consideration the sum of the squared errors (SSE), the sample size, and the number of parameters.

For both AIC and BIC, the lower the value the better. Hence, we choose the model with the lowest AIC or BIC value. Although we can use either of the two criteria, AIC is usually criticized for its tendency to overfit. In contrast, BIC penalizes the number of parameters more severely than AIC and hence favors more parsimonious models.

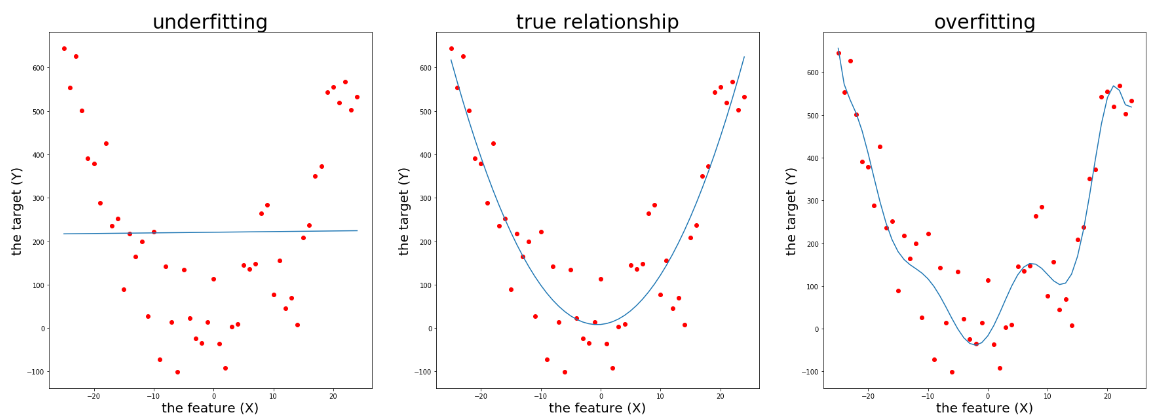
**Overfitting**

1. **Generalization**

One of the main challenges in machine learning is to design models that perform well not just on a training set but also on a test set or more generally on unseen observations. More concretely, we want:

1. our models to have a low error rate on the training set
2. the difference between the errors in the test set and in the training set to be also low

A model's ability to achieve that second requirement is called **generalization**, and the difference between the errors in the test and training sets is called the **generalization gap**. *A good model should generalize well, and its generalization gap should be as small as possible*.



***Generally, if our model is too complex then it will tend to overfit. Inversely, if our model is not complex enough, it will underfit the training set****.*

Keep in mind that a model's complexity is also referred to as its **capacity**. So if we find that our model is overfitting, we should reduce the complexity or capacity of the model. We can do this by eliminating some unimportant features from the model or by reducing the degree of the polynomial.

**Cost Function**

**OLS** cost function optimizes variance explained in the training set.

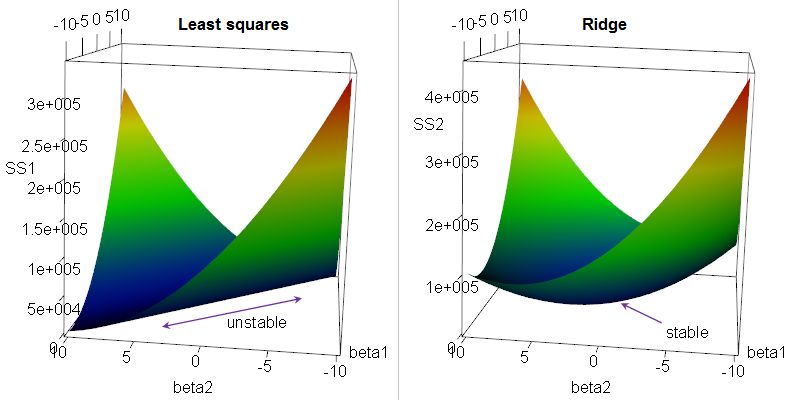
**Ridge**, **Lasso**, and **ElasticNet** each optimize variance explained in the test sets.

*In general, our goal is to make a model that tells us about the world (and not just our training sample) so Ridge, Lasso, and ElasticNet solutions are useful.*

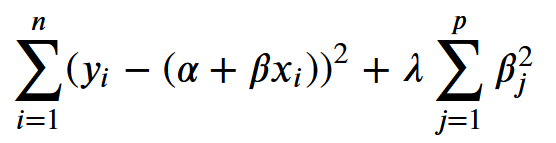
**B. Regularization**

To make a model more generalizable a test set, we sometimes adjust the model's learning objectives or loss functions. By doing this, we actually impose our preferences over potential solutions and force the model to choose one of our preferred solutions, assuming there isn't a nonpreferred solution that performs significantly better. In general, the term **regularization** refers to the process of modifying algorithms in order to lower the generalization gap without sacrificing training performance.

**B-1. Ridge regression**

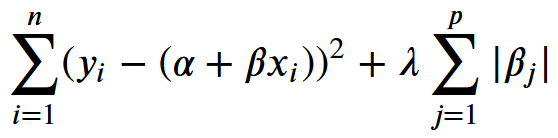


**1**: Ridge regression "regularizes" a model more than OLS does.



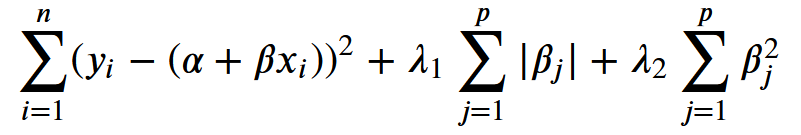
* + Also called "**L2 regularization**" (Based on sum of *squared* weights/coefficient estimates).
  + Imposes a penalty for large coefficients. It represents the sum of the square of all model coefficients (numbered 1 through *p*), multiplied by the regularization parameter λ**.**
  + As model’s complexity increases and features correlate with one another more and more (become \_multicollinear\_), coefficients arrived at by OLS become increasingly large. This is a sign that the model is incorporating too much variance from the training set – in other words, overfitting.
  + Ridge regression is a model variance minimizer.
  + *The best way to determine the λ parameter is cross-validation.*

**B-2. Lasso regression**



* Also called "**L1 regularization**" (based on the sum of the *absolute* weights).
* Tries to force small parameter estimates to be equal to zero, effectively dropping them from the model.
* Can prevent overfitting, and also works as an **embedded feature selection method**.
* Extremely handy when running thousands of predictors and need to optimize processor time, or to increase model understandability.
* *The best way to determine the λ parameter is cross-validation.*

**B-3. ElasticNet regression**



* When λ₁ is equal to zero, this loss function turns into the loss function of Ridge regression and when λ2 is equal to zero it becomes the loss function of Lasso regression. Hence both Lasso and Ridge regression are special cases of ElasticNet regression. Moreover, when both λ₁ and λ2 are equal to zero, Elasti5cNet becomes a linear regression model!
* In ElasticNet regression there are 2 hyperparameters to tune: 𝜆1 & 𝜆2. Sk-learn uses different terminology: alpha and l1\_ratio:

𝑎𝑙𝑝ℎ𝑎 = 𝜆1 + 𝜆2

𝑙1\_𝑟𝑎𝑡𝑖𝑜=𝜆1(𝜆1+𝜆2)

* + - * In other words:

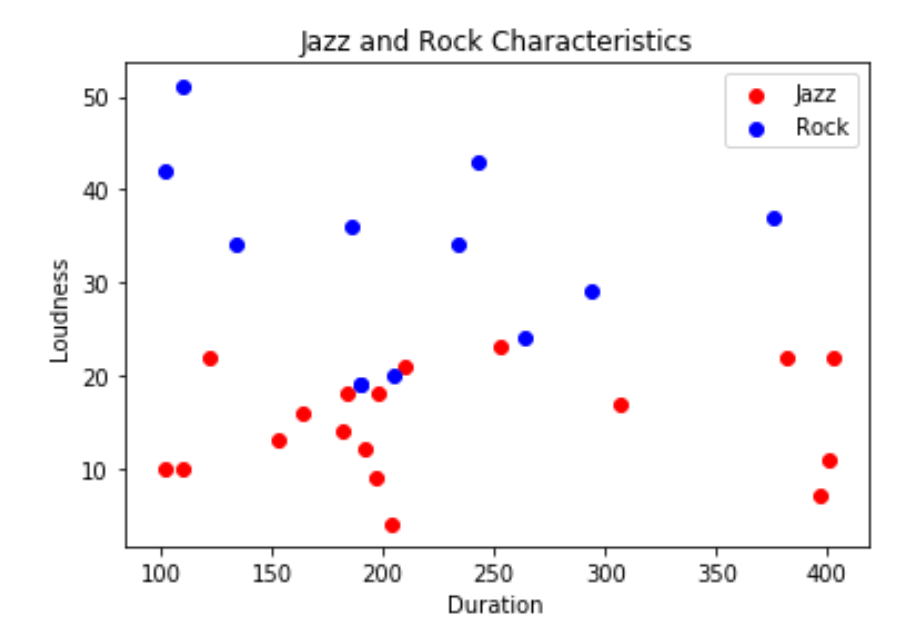
𝜆1=𝑎𝑙𝑝ℎ𝑎∗𝑙1\_𝑟𝑎𝑡𝑖𝑜

𝜆2=𝑎𝑙𝑝ℎ𝑎∗(1−𝑙1\_𝑟𝑎𝑡𝑖𝑜)

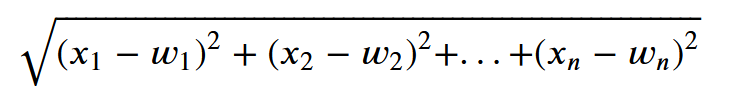
* + - * So in effect, they are equivalent!
* *As emphasized before, the best way to determine the λ parameter is cross-validation.*
* learning via probability 🡪 (naive Bayes)
* learning via errors 🡪 (regression)
* learning via similarity 🡪 KNN

**KNN**

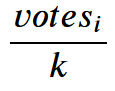
classifying a song as either "rock" or "jazz"



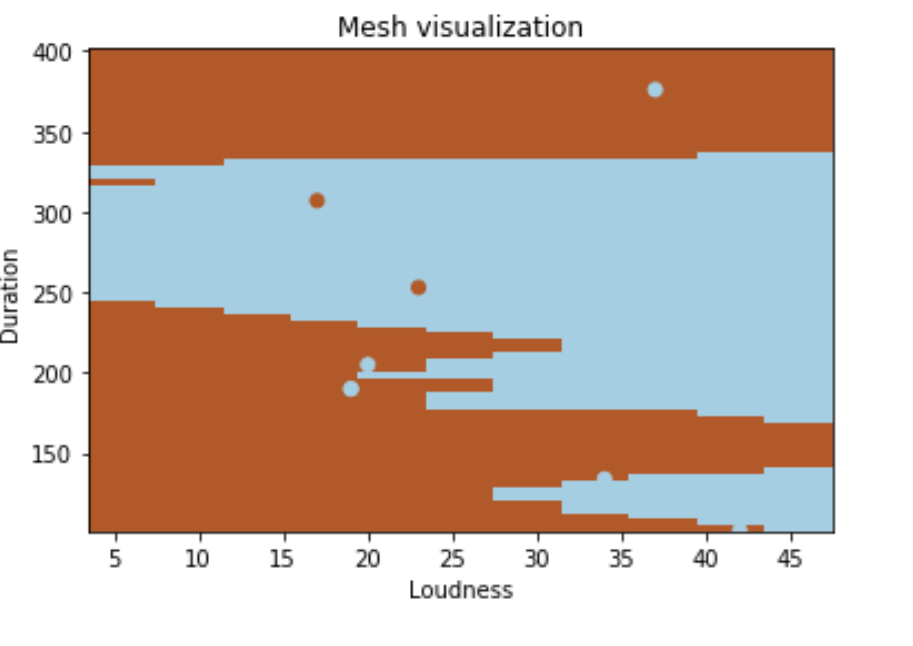
Euclidean distance



You can technically define any distance measure you want, and there are times where this customization may be valuable.

If each of the 𝑘 neighbors gets a vote on the outcome, then the probability of the test example being from any given class 𝑖 is: 

Visualizing KNN



**Distance and Normalizing**

**Normalization** is a way of taking seemingly incommensurate measures and making them comparable.

There are two main normalization techniques that are effective with KNN.

1. You can set the bounds of the data to 0 and 1, and then rescale every variable to be within those bounds. Every data point is measured in distance between max & min. Best if data shows a linear relationship. It is also best if there are known limits to the dataset, as those make for logical bounds for 0 and 1 for the rescaling.
2. You can also calculate how far each observation is from the mean, expressed in number of standard deviations: this is often called z-scores. Calculating z-scores and using them as your basis for measuring distance works for continuous data and puts everything in terms of how far from the mean (or "abnormal") it is.

You'll have to use your intuition to see which makes the most sense. Mixing them, while possible, is usually a dangerous proposition.

**Weighting**

Functionally this will weight by the inverse of distance, so that closer datapoints (with a low distance) have a higher weight than further ones. In SKLearn there is an optional weights parameter.

**Choosing K**

Choosing 𝑘 is a tradeoff.

* The larger the 𝑘 the more observations getting a vote in the prediction (smoothed).
* Smaller 𝑘 will pick up more subtle deviations, but these deviations could be just randomness (overfitting). Add in weighting and that's an additional dimension to this entire conversation.

The best technique is probably to try multiple models and use your validation techniques to see which is best. In particular, k-fold cross validation is a great way to see how your KNN model is performing