**Week 1**

* Machine learning allows computers to learn from data with traditional statistical modeling. We use our understanding of the underlying data generating process to develop a predictive model. But sometimes in machine learning, it’s too complicated to express why are we seeing the data we have and still try to approximate the equation from the data.
* Supervised machine learning where the outcome is known, and that outcome is your dependent variable that you’re trying to predict or understand.
* Parameters v. Hyperparameters which are decisions we must make to build the “right” model.
* Train v test sets are crucial so that our model does not use an overly specific criteria when making predictions. We want the model to generalize well to data it hasn’t seen, meaning that we don’t want the quality of our predictions depend much on which dataset we feed to the model.
* Loss functions define just some quantitative score for how good our predictions are to the actual values. Often, we want to minimize the error between y and our prediction.

*Interpretation:*

* Train a model to find insights from the data. The betas will help us understand which features provide the most value in predicting our outcome variable because those features affect our outcome the most.
* Thus, we are choosing a model with high interpretability. For example, what’s driving sales rather than predicting future sales. Or marketing budget’s effectiveness to understand movie revenue.

*Prediction:*

* Prediction approach compares our predicted values and the actual values. We want to choose a model that minimizes the sum of squared residuals and generalizes well to unseen data.
* All we care about is the best prediction such as customer churn likelihood, default, or next purchase amount or when it will happen.
* A great way to understand how good our model is at making predictions is to plot Actual v Predicted values and a diagonal line with a slope of 1 and intercept 0. Thus, the closer the points to the diagonal line the more accurate your prediction.

*Two Types of Supervised Learning*

* When the outcome is continuous (numerical), regression is an ideal choice. But when the outcome is categorical, outcome is a category.
* For example, support we start out with movie revenue data and choose a model to fit whether a linear regression, polynomial regression, time series model, and so on. You fit the data with outcomes to learn the parameters. Finally, use the models and the fitted parameters to predict movie revenues for movies whose revenues we do not know.
* Process for categorical data is exactly the same except your outcome variable is categorical.

*Linear Regression*

* Minimize the cost function, the sum of squared residuals to estimate the parameters.
* SSE, TSS, 1-(SSE/TSS), adjusted R^2.

*Insights from the Lab:*

* **Determining Normality**. The lab covers three main methods to making your target variable normally distributed. Along with visual aid, you can use the normaltest function from scipy.stats.mstats library. If the p-value > 0.05, you fail to reject the null hypothesis that distribution is normal.
* The three common transformations to try and get y to be normally distributed are natural log transformation, taking the square root of your variable, and the box-cox method.
* The first two methods are straightforward: apply them to the dependent variable and do the D’Agostino K^2 test.
* The box cox method is a generalization of the square root function. Its objective is to get distributions as close to a normal distribution as possible.
  + *,* where lambda is the best exponent determined by the boxcox function from the scipy.stats library.
  + The function returns a tuple of length 2. The first element is a numpy array consisting of the transformed dataset. The second element is lambda, the best exponent per the function.
* The lab selected a polynomial regression model. We cover that in weeks 2 and 3.
* Because we are using polynomial regression, we must transform the independent variables. This transformation will result in a data frame that consists all of the original variables, the square of each variable, and the interaction between each and unique variable. Thus, the transformed X will have a lot more columns.
* Split the data into training and testing sets: x\_train and x\_test
* And then you use a StandardScaler () object to standardize both x\_train and x\_test; for each column’s value, subtract the mean and divide by the standard deviation so that all the variables are now on the same scale:
* Use the boxcox function on y\_train set and fit the linear regression object of x\_train\_s and y\_train. And then make predictions on x\_test\_s to get y\_pred\_c.
* Remember that y\_pred\_c is transformed because we trained the model using a normalized y. Thus, use the inv\_boxcox function to transform y\_pred\_c back to the original scale of the dependent variable and check how good the predictions are compared to the test set.

**Week 2**

**Training and Test Set**

* Use the training data to learn the parameters and use the test data to see how the model performs on unseen data.
* From sklearn.model\_selection import train\_test\_split, define a test\_size and pass the data into train\_test\_split.
* You know the rest.
* There is ShuffeSplit where you will have four different train and test splits so it could be the last 30%, the middle 30% or random 30%.
* Stratified Split allows you to maintain the proportions of the different categories in training and test data to avoid bias. Here is another way to understand Stratified Split. The training and test sets are subsamples of the original data. Assuming that your original data is representative of the population, you want your splits to also be representative of the population. Stratified split allows you to do just that.

**Cross Validation**

* Say you split the data into 4 folds. In this case, there will be 4 iterations where 3/4th of your data will be used to train your model, and the remaining 1/4th will be used to test your model.
* Every iteration will yield four different scores about how well the model did on the validation set, such as a mean squared error. Then we will take the average of all these validations set scores to understand how well the model performed on holdout sets.

***Model Complexity v. Error.***

* As the model becomes more complex, the training error will decrease.
* But a complex model will decrease your validation error score only up to a certain point. After such point, the model doesn’t generalize well to your test data.

***Different Types of Cross Validation***

* Leave One Out Cross Validation: This time, k will equal to the number of rows minus 1 where for every single train set, split just a single row and train on the remaining rows. You’ll get many more test sets as the algorithm is applied once for each instance. However, the algorithm will take a lot longer.
* Stratified Cross Validation: Ensures that our train test splits maintain the proportions of different categories from the original data.

**Week 3**

* You want the training and test error to be small. Training error will always decrease.
* ***Chart

  Description automatically generated***
* ***Graphical user interface, diagram

  Description automatically generated***
* So, something like cross validation will refer to our tendency to have high or low variance as well as high or low bias given our hold-out sets.
* Three essential ways models can produce errors in their predictions.
  + The model choice is wrong, and we are consistently getting the wrong prediction (Bias). Happens because we are not controlling for a variable or we are making overly simplistic assumptions about the relationship between the response and the predictors.
  + The model is unstable and has high variance because it is overfitting the training data. Flexible models that follow the observations very closely have high variance because small changes in the training data results in large changes in f-hat.
  + Irreducible error which will always be the reason why we won’t have perfect predictions. But this error is the least of our concern as we are trying to find the underlying relationship and not model the noise.

***Regularization and Model Selection***

* Lambda allows us to manage complexity parameter. Higher parameter means a simpler model.

***Ridge Regression***

Text

Description automatically generated

* Now we’re not only trying to reduce error but also ensuring that our model is not too complex.
* Now scale matters a lot because Ridge penalized large coefficients. Suppose you’re using the number of stores and price of an item to predict sales. If the price is between $8 and $10 and the number of stores is between 10,000 and 20,000 stores, the coefficient for price of $1 change will be much larger, assuming both variables and the response have the same covariance, the variance of price will be smaller.
* Thus, different features should be on the same scale.
* We can select the best regularization strength lambda via cross-validation and grid search.

**Regularization Details**

Text, letter

Description automatically generated

* Suppose you only 2 regression coefficients. Lasso’s equation indicates that its coefficient estimates have the smallest RSS out of all points that lie within the diamond defined by |β1| + |β2| ≤ s.
* Ridge’s equation indicates that its coefficient estimates have the smallest RSS out of all points that lie within the circle defines by .
* When we perform the Lasso, we are trying to find the set of coefficient estimates that lead to the smallest RSS subject to the constraint that there is a budget s for how large can be.
* Similar idea applied to Ridge regression.
* When the budget is very large, the OLS least square solution will fall within the budget and the lasso formula will yield the least square solution.

**Diagram

Description automatically generated**

* If we weren’t constrained by regularization, the optimal solution would be the beta hat in the middle.
* All the points on a given ellipse will share a common RSS.
* Since ridge regression has a circular constraint with no sharp points, this intersection will not generally occur on an axis, and so the ridge regression coefficient estimates will be exclusively non-zero. However, the lasso constraint has corners at each of the axes, and so the ellipse will often intersect the constraint region at an axis. When this occurs, one of the coefficients will equal zero.
* These ideas hold in higher dimensions.