University of Glasgow | School of Computing Science

CSC3005 Laboratory/Tutorial 3: Data Model Overfitting and Multiple Regression

1. Data Model Overfitting

Assume a two-dimensional dataset containing 10,000 labeled instances, each of which is assigned to one of two classes, 0 or 1. Instances from each class are generated as follows:

1. Instances from class 1 are generated from a mixture of 4 Gaussian distributions, cantered at [5,15], [15,15], and [15,5], [5, 5] with covariance of 2 with zero mean respectively.
2. Instances from class 0 are generated from a uniform distribution in a square region, whose sides have a length equals to 20.

Step 1: Create the data set of the above with 5000 set from class 0 and 5000 set from class 1 and plot it out. Hint: Use numpy.concatenate ( ) to combine dataset and numpy.random.multivariate_normal to create gaussian distribution data. **X=** data , **Y** = class

https://numpy.org/doc/stable/reference/generated/numpy.concatenate.html?highlight=numpy.concatenate#numpy.concatenate

https://numpy.org/doc/stable/reference/random/generated/numpy.random.multivariate_normal.html?highlight=numpy.random.multivariate_normal#numpy.random.multivariate_normal

Step 2: Split the Data into Training and Test in the ratio of 70:30 using sklearn.model_selection.train_test_split() function.

https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.train_test_split.html?highlight=sklearn.model_selection%20train_test_split#sklearn.model_selection.train_test_split

Step 3: Create Decision Tree Classifier and test the performance for various tree depth

from maxdepths=[2,3,4,5,6,7,8,9,10,15,20,25,30,35,40,45,50[. Model fit the data from step 2 and compare the accuracy using sklearn.metrics.accuracy_score() learn in Tutorial/lab 2.

https://scikit-learn.org/stable/modules/generated/sklearn.metrics.accuracy_score.html?highlight=accuracy_score#sklearn.metrics.accuracy_score

Step 4 : Plot the performance of training accuracy and test accuracy versus the max depth. What can you observe?

2. Linear Regression

Create a random 1-dimensional vector of predictor variables, x, from a uniform distribution of size 100. The response variable y has a linear relationship with x according to the following equation: $y = w_1 x + w_0 + \epsilon \rightarrow y = -5x + 5 + \epsilon$ where ϵ corresponds to random noise sampled from a Gaussian distribution with mean 0 and standard deviation of 2

Step 1: Create the data set of 100 for the above equation that is corrupted by the gaussian noise of mean 0 and standard deviation of 2.

Step 2: Perform the linear regression by estimating the regression coefficient and the y intercept with following procedure

a. Split the Data in Step 1 into Training and Test Set in the ratio of 70 and 30

   The number of training set =70 while number of test set=30. In Summary, split the original data X and Y into X_train, Y_train and X_test and Y_test data set.

b. Fit Regression Model with Training Set

   Import sk.learn,linear_model and call Linear Regression and fit method model fit X_train and Y_train data.

   https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LinearRegression.html?highlight=linear_model.linearregression#sklearn.linear_model.LinearRegression https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LinearRegression.html?highlight=linear_model.linearregression#sklearn.linear_model.LinearRegression.fit

c. Apply the trained model obtained in step (b) to the test set

Use the predict() method with the X_test data to obtain the Y_pred_test

https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LinearRegression.html?highlight=linear_model.linearregression#sklearn.linear_model.LinearRegression.predict

d. Evaluate the performance on test set output using RMSE and R^2

Import the mean_squared_error and r2_score from sklearn.metrics.mean_squared_error () and sklearn.metrics.r2_score. Compare both error between the Y_test and Y_pred_test data set.

https://scikit-learn.org/stable/modules/generated/sklearn.metrics.mean_squared_error.html?highlight=sklearn.metrics#sklearn.metrics.mean_squared_error

https://scikit-learn.org/stable/modules/generated/sklearn.metrics.r2_score.html?highlight=sklearn.metrics#sklearn.metrics.r2_score

Plot the measured values of y versus the predicted values of y (Y_test vs Y_pred_test)

e. Post Processing: Visualization

Plot predicted regression line against the backdrop of test set (X_test and Y_test).

Use the linearRegression.coef_[0] and linearRegression.intercept_[0] to show both the regression coefficient , w_1 and y intercept, w_0

3. Explore the effect among features that are correlated

The presence of correlated attributes can affect the performance of regression models. We create 4 additional variables, x_2 , x_3 , x_4 , and x_5 that are strongly correlated with the previous variable x created in Question 2 with the following relationship. That is we are going to create $y = w_1 x + w_2 x_2 + w_3 x_3 + w_4 x_4 + w_5 x_5$

1) $x_2 = 0.5x$ + gaussian noise of zero mean and std variation of 0.05

2) $x_3 = 0.5x_2$+ gaussian noise of zero mean and std variation of 0.01

3) $x_4 = 0.5x_3$+ gaussian noise of zero mean and std variation of 0.01

4) $x_5 = 0.5x_4$+ gaussian noise of zero mean and std variation of 0.01

We then fit y against the predictor variables and compare their training and test set errors.

Step 1: Create the 4 additional features x_2 to x_5

Step 2: Plot the pairwise correlation among each other for

1)Between x and x_2

2)Between x_2 and x_3

3)Between x_3 and x_4

4)Between x_4 and x_5

Use the numpy.corrcoef to find correlation between 2 set of data and use numpy.column_stack to stack two set of data together.

https://numpy.org/doc/stable/reference/generated/numpy.corrcoef.html?highlight=numpy.corrcoef#numpy.corrcoef

https://numpy.org/doc/stable/reference/generated/numpy.column_stack.html?highlight=numpy.column_stack#numpy.column_stack

Step 3: Create 4 set of training and test set each with ratio of 70:30

1)X_train2 = [70 set of x, 70 set of x_2]

1)X_test2 = [30 set of x, 30 set of x_2]

2)X_train3 = [70 set of x, 70 set of x_2, 70 set of x_3]

2)X_test3 = [30 set of x, 30 set of x_2, 30 set of x_3]

3)X_train4 = [70 set of x, 70 set of x_2, 70 set of x_3, 70 set of x_4]

3)X_test4 = [30 set of x, 30 set of x_2, 30 set of x_3, 30 set of x_4]

4)X_train5 = [70 set of x, 70 set of x_2, 70 set of x_3, 70 set of x_4, 70 set of x_5]

4)X_test5 = [30 set of x, 30 set of x_2, 30 set of x_3, 30 set of x_4, 30 set of x_5]

The first pair, X_train2 and X_test2 have 2 correlated predictor variables, x and x_2.

The second pair, X_train3 and X_test3 have 3 correlated predictor variables, x, x_2, and x_3.

The third pair have 4 correlated variables, x, x_2, x_3, and x_4

The last pair have 5 correlated variables, x, x_2, x_3, x_4, and x_5.

Use numpy.column_stack() to stack different data

https://numpy.org/doc/stable/reference/generated/numpy.column_stack.html?highlight=numpy.column_stack#numpy.column_stack

Step 4: Train the 4 new regression models based on the 4 pairs of training and test data created in the step 3.Run LinearRegression and fit method , similarly in Question 2 step 2(b) for

1)X_train2 and Y_train

2)X_train3 and Y_train

3)X_train4 and Y_train

4)X_train5 and Y_train

Step 5: Apply created regression model to both training and test data sets, using the predict() function, similarly In Question 2 step 2(c)

Predict the output training output and test output

1)Y_pred_train, Y_pred_train2,Y_pred_train3,Y_pred_train4,Y_pred_train5

2)Y_pred_test,Y_pred_test2,Y_pred_test3,Y_pred_test4,Y_pred_test5

with the respective training and test of X namely

1)X_train, X_train2, X_train3, X_train4, X_train5

2)X_test, X_test2, X_test3, X_test4, X_test5

That is to say, we will have 5 predicted value of equation namely

Model 0 : $y = w_1 x + w_o$

Model 1: $y = w_1 x + w_2 x_2 + w_o$

Model 2: $y = w_1 x + w_2 x_2 + w_3 x_3 + w_o$

Model 3: $y = w_1 x + w_2 x_2 + w_3 x_3 + w_4 x_4 + w_0$

Model 4: $y = w_1 x + w_2 x_2 + w_3 x_3 + w_4 x_4 + w_5 x_5$

Step 6: Postprocessing-Visualization

For postprocessing, compute both the training and test errors of all models. Show the resulting model and the sum of the absolute weights of the regression coefficients of all these models, i.e., $\sum_{j=0}^{k} |w_j|$ where k is the number of predictor features.

1)obtain RMSE error between Y_train and Y_pred_train

2)obtain RMSE error between Y_test and Y_pred_test

3)obtain absolute sum of the weights as shown above in the formulae

Plot the sum of absolute weights versus the error rate for both training and test error. What can you observe from the plot?

4. Regularization

i)Ridge regression is a variant of MLR designed to fit a linear model to the dataset by minimizing the following regularized least-square loss function:

$$L_{ridge}\left(y, f(\mathbf{X}, \mathbf{w})\right) = \sum_{i=1}^{N} \|y_i - \mathbf{X}_i\mathbf{w} - w_0\|^2 + \alpha[\|\boldsymbol{w}\|^2 + w_0^2]$$

where α is the hyperparameter for ridge regression. Note that the ridge regression model reduces to MLR when α=0. By increasing the value of α, we can control the complexity of the model

Step 1: Train the Ridge model

Import the linear_model.Ridge and create and Ridge object with α=0.5. For the model with the training data X_train5 and Y_train.

https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.Ridge.html?highlight=linear_model.ridge#sklearn.linear_model.Ridge

https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.Ridge.html?highlight=linear_model.ridge#sklearn.linear_model.Ridge.fit

Step 2: Predict the output with the trained Ridge model

Import the linear_model.Ridge and create and Ridge object with α=0.5. For the model with the training data X_train5 and Y_train

Use the predict() method to obtain the
i)Y_pred_train_ridge with X_train5 data
ii)Y_pred_test_ridge with X_test5 data

https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.Ridge.html?highlight=linear_model.ridge#sklearn.linear_model.Ridge.predict

Step 3: Performance Evaluation and show the results

Obtain
1) RMSE error between Y_train and Y_pred_train_ridge
2) RMSE error between Y_test and Y_pred_test_ridge
3) sum of the weight

Show all models' results and the sum of the absolute weights of the regression coefficients of all these models. What can you observe on the Ridge's performance as compared to the earlier model?

ii)Lasso regression: One of the limitations of ridge regression is that, although it was able to reduce the regression coefficients associated with the correlated attributes and reduce the effect of model overfitting, the resulting model is still not sparse. Another variation of MLR, called lasso regression, is designed to produce sparser models by imposing an ℓ1 regularization on the regression coefficients as shown below:

$$L_{lasso}\left(y, f(\mathbf{X}, \mathbf{w})\right) = \sum_{i=1}^{N} \left\|y_i - \mathbf{X}_i \boldsymbol{w} - w_0\right\|^2 + \alpha[\|\boldsymbol{w}\| + w_0]$$

Step : Repeat the whole process as in Ridge model

Import linear_model.lasso from sklearn and repeat the process similar to ridge regression. What can you observe on the lasso's performance as compared to the earlier model?

https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.Lasso.html?highlight=lasso#sklearn.linear_model.Lasso

5. Model Selection via Cross Validation

While both ridge and lasso regression methods can potentially alleviate the model overfitting problem, one of the challenges is how to select the appropriate hyperparameter value, α. Using k-fold cross-validation method to select the best hyperparameter of the model.

Step : Train RidgeCV model

Using the RidgeCV() function, we can train a model with k-fold cross-validation and select the best hyperparameter value.Set $k = 5$ and α range from 0.3 to 1 in step of 0.2. Import linear_model.RidgeCV() from sklearn and repeat the process similar to earlier ridge regression. What can you observe on the RidgeCV's performance as compared to the earlier model? What value of α has been selected by the system?

https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.RidgeCV.html?highlight=ridgecv#sklearn.linear_model.RidgeCV

6. **Basis Function Regression**

To adapt linear regression to nonlinear relationships between variables, the data can be transformed using basis functions. Using the PolynomialRegression pipeline in sklearn.preprocessing, the idea is to take the multidimensional linear model:

$$y = w_0 + w_1x_1 + w_2x_2 + w_3x_3 + w_4x_4 + w_5x_5 + \cdots$$

and build the $x_1.x_2, x_3$, and so on, from the single-dimensional input x. That is, let $x_n = f_n(x)$ where $f_n()$ is some function that transforms our data. For example, if $f_n(x) = x^n$, the model becomes a polynomial regression:

$$y = w_0 + w_1x_1 + w_2x^2 + w_3x^3 + w_4x^4 + w_5x^5 + \cdots$$

Step 1 :Import the polynomialFeatures class from sklearn.preprocessing library

1)Create 1 dimensional column array x of size 3 with values say 1 2 3 using np.array([1,2,3])
2)create an object called poly by using PolynominaFeatures (3, include_bias=False). This will set transform the any input x to 3 inputs with increasing raised power, x, x^2, x^3
3)fit.transform the input data x
4)Observe the output

https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.PolynomialFeatures.html?highlight=polynomialfeatures#sklearn.preprocessing.PolynomialFeatures

Step 2: Step 2: Make a 10th order polynomial model

1)from the sklearn.pipeline import make_pipeline
2)from the sklearn.linear_model import LinearRegression
3)Combine both PolynomialFeatures and LinearRegression class into poly_model by using make_pipeline(PolynomialFeatures,LinearRegression()). Use the polynonimal order equal to 10

https://scikit-learn.org/stable/modules/generated/sklearn.pipeline.make_pipeline.html?highlight=make_pipeline#sklearn.pipeline.make_pipeline

Step 3: Create the input data

1)create the 100 uniformly distrbuted value for the x axis that range from 0 to 20 using np.random.Randomstate().rand
2)create $y = \sin(x) + \varepsilon$ where ε =normal distrbuted noise with mean 0 and standard deviation of 1. Use randn function

Step 4: Build the model

1)build the model with the data x by first converting it to 100x1 matrix using $x[:, numpy.newaxis]$
2)create the model by feeding both matrix new x and the y data using fit() function

Step 5: Test the model and plot the original data(x,y) verus ($x, predicted\ y$)

1)create the 1000 linearly spaced values of xx axis values between 0 to 20 by using numpy.linspace() function. Called x_test
2)Feed the x_test values into the predict() function of the poly_model
3)plot the scatterplot of original x and y and overlay with the x_test and the predicted y values

What is your observation?