Machine Learning Assignment 5 Hands on with Regularization and Tree-based Ensemble

(40 pts)

For this problem, we are going to use the housing dataset from this kaggle competition.
The dataset has 80 variables for each house describing the lot shape, size, neighborhood, number of rooms, number of bathrooms, etc. The goal is to predict the house sale price from these variables.

Download and unzip the attached data. It consists of two files: 1- housing.csv which is the dataset and 2- data_description.txt which describes the variables in the data. Read this description to understand what each variable means.

Remove the first column, as it is a unique identifier and not used in predicting sale house price.

Section 1. Data Cleaning

1. Take a summary of the data and explore the result. How many categorical and numerical variables are there in the dataset?
2. (1pt) Which columns have missing values and **what percentage of those columns** have NAs?
3. (1pt)Read the data description carefully. For some of the variables, such as PoolQC, FirePlaceQU, Fence, etc. NA means not applicable rather than missing at random. For instance, a house that does not have a pool gets NA for PoolQC. For those variables for which NA means not applicable, you can replace NA with zero (if that variable is numeric) or replace it with a new category/level, for instance, “notApplicable” or “None” if that variable is categorical.
4. (1pt) After replacing not applicable NAs with appropriate values, find out which columns (if any) still have NAs and what percentage of each column is missing.
5. (1pt) **what percentage of rows** in the dataset have one or more missing values?

Section 2. Data Exploration

6. (1pt) plot the histogram of SalePrice. Interpret the histogram. Is SalePrice variable skewed?

7. (3 pt) Use appropriate plots and test statistics to find out which variables are associated with SalePrice. Remove variables that show no association or very weak association with salesprice. **Note: the t-test or one.way test for some variables may throw an error if there are not enough observations in a group. You can ignore this error at this point. Later we will find and eliminate groups with little variance.**

Dealing with Missing Values:

If a large percentage of rows still have missing values, omitting those rows will cause loss of information. We are going to try to impute the missing values. Several methods for missing data imputation exists. The

most simple imputation is to replace the missing values with the mean(or mode) of their columns. Another method would be to use other columns to predict the column with missing values This is called **multiple imputation** and it assumes that a missing value in one attribute can be predicted from the data in other attributes. The simplest form of multiple imputation is to use **knn imputation.** In knn imputation, we find the k complete data points closest to the data point with missing value (incomplete row) and take their average to fill the missing value.

To avoid data leakage, the nearest neighbors must be computed based only on the training data instead of the entire dataset. In other words, to impute the missing values in training, validation, and test data we should only use the training data to find the nearest neighbors.

Data leakage is when information outside of the training data is used in creating a machine learning model. This causes the model to overfit and not generalize well to future data.

When doing cross validation, the imputation must be computed based only on the training data on each fold (excluding the validation data). Luckily, caret's train method in R streamlines this process. you can use the option **preProc="knnImpute"** to do knn imputation and the option **na.action=na.pass** to allow the NA values to be passed to the model to be imputed:

```
train( …, na.action=na.pass, preProc="knnImpute",…)
```

You will do this in the next section when you train various machine learning models.

KnnImpute scales and centers the numeric features and uses Euclidian distance to compute the nearest neighbors. Since Euclidean distance is not meaningful for categorical features, it will ignore categorical features if present in the dataset and does not impute them.

8. (2 pt) Examine the columns with missing values to see if any of them are categorical. Use caret's createDataPartition method to partition the dataset to 80% training and 20% testing. If a categorical column has missing values in train or test data, impute it with the **mode of that column** in the training data. It is important that the mode is computed based only on the training data only (instead of the entire dataset) to avoid data leakage.

Eliminating variables with little to no variance:

This dataset has several variables with a handful of unique values that occur with very low frequency. For instance, if you take a summary of the "Street" variable 99.5% of the samples have "Pave" street while only 0.04% of samples have "gravel". The concern here that when we split the data for cross validation, the training samples will all have "paved" street but some of the validation samples might have "gravel" street which create a discrepancy between training and validation sets. In addition, a handful of unique values may have an undue influence on the model. Therefore, it is better to identify and eliminate these variables with *near zero variance.*

Luckily caret's train method have a preprocessing option preProc="nzv" that can be used during cross validation to remove variables with zero or little variance in the training set.

Use preProc=c("knnImpute", "nzv") inside caret's train method in the models you create in the next section to eliminate variables with near zero variance and to impute missing values in numeric variables using knnImpute.

Section 3. Creating Predictive Models

After cleaning and exploring data, we are ready to build our machine learning models to predict the SalePrice of a house based on other variables. We are going to examine four categories of models: Regularized linear regression, Tree-based Ensemble models, SVM, and neural networks with drop out.

Section 3.1 Creating Regularized Linear Regression Models

8. (2pt) Set.seed(1) and train a **Lasso Linear Regression model** using "glmnet" and "caret" as explained in the lectures to predict the SalePrice. Use 10 fold cross validation and Tune the lambda parameter) Note: You do not need to worry about scaling your test or train data, glmnet will automatically do it for you.

Use preProc=c("knnImpute","nzv") and na.action=na.pass options inside the train method to let caret impute the missing values using knn based on the training data during cross validation.

9. (1pt) Get the coefficients for the best tuned model. Did Lasso shrink some of the coefficients to zero? If so, what does this mean? (1 pt)

10. (1pt) Get the predictions on the test data using the "predict" function with option na.action=na.pass. This will allow the NA values in the test data to be passed to the model and imputed using knn imputation based on the training data. Go ahead and run install.packages("RANN") to install the RANN package if you get an error stating that you need this package.

    Compute the RMSE on of the predictions.

11. (1 pt) set.seed(1) again and train a **Ridge linear regression model** using 10 fold cross validation and tune lambda as you did for lasso and compute the RMSE of this model on the test data. Use knn imputation similar to what you did for lasso.

12. (1 pt) set.seed(1) again and train an **Elastic net linear regression model** using 10 fold cross validation and tune lambda as you did before and tune alpha to be a sequence of 10 values between 0 and 1, that is: 0,0.1,0.2,….1 . Compute the RMSE of the tuned model on the test data Use knn imputation similar to what you did for the two previous models.

Section 3.2 Creating Tree-Ensemble and SVM Models

13. (2 pt) Set.seed(1) and Use Caret package with "rf" method to train a random forest model on the training data to predict the SalePrice. You can impute the missing values using knn similar to what you did for the previous models. Use 10-fold cross validation and let caret auto-tune the model. Use the model to predict the SalePrice for test data and compute RMSE. (**Note:** use **importance=T** in your train method so it computes the variable importance while building the model). **Be patient. This model may take a long time to train.**

(1 pt) User caret's varImp function to get the variable importance for the random forest model. Which variables were most predictive in the random forest model?

14. (1 pt) Set.seed(1) and Use Caret package with "gbm" method to train a Gradient Boosted Tree model on the training data. GBM needs minimum data preprocessing, you don't need to scale numeric features or encode the categorical variables. In addition, it can be trained directly on data with missing values without having to do imputation.

Use 10 fold cross validation and let caret auto-tune the mode.

Use the model to predict the SalePrice for the test data and compute RMSE.

15. (1 pt) Set.seed(1) and Use Caret package with "svmLinear" method to train a support vector machine model on the training data. Use preProc="knnImpute" to impute the missing values and scale data.

Use 10 fold cross validation and let caret auto-tune the model, **explain what is hyper-parameter "c"?** Use the model to predict the SalePrice for the test data and compute RMSE.

16. (1 pt) repeat the above steps but set train method to "svmRadial" to use radial basis function as kernel.

17. (2pt) Use "resamples" method to compare the cross validation RMSE of the seven models you created above (LASSO, RIDGE, elastic net, randomforest, gbm, svmlinear, and svmradial). **In a sentence or two, interpret the results.**

Section 3.3 Creating a Neural Network Model

18. Split the training data to train –validation set. (use 90% for training and 10% for validation)

19. Use knn imputation to impute the missing values in the train/validation/ and test data based on the training data. To do this, you can use the preProcess function in caret with method="knnImpute" on the training data. This will return a preprocessing model which can be used to transform data using the predict function. Call the predict function as follows to do knn imputation on train, test, and validation data based on the information on the training data:

```
library("RANN")

preproc <- preProcess(train, method="knnImpute") #train is your training
data

train.imputed <- predict(preproc, train)

test.imputed <- predict(preproc, test) #test is your test data

val_imputed <- predict(preproc, val)    #val is your validation data
```

The above code segment:

- ignores the categorical variables
- scales and centers the numeric variables in train, test, and validation sets based on the mean and standard deviation values computed from the training set
- imputes the numerical features in train, test, and validation data using knn imputation computed based only on the information in the training data.

20. (3 pt) Neural Networks cannot take factor variables and you must convert your categorical variables to numbers before training your neural network model. One-hot encode the categorical variables using one_hot function from the mltools package.. Set dropUnusedLevels=FALSE. This is to make sure that the train and test data will have the same number of features in case some levels of a categorical variable does not occur in training data but occurs in test or validation data.

The drawback of one hot encoding is that it creates a binary variable for each level of a categorical variable. This increases the dimensionality of the training data and makes it sparse, especially for categorical variables with a large number of levels. In our next and final lecture, we learn a more advanced method for encoding sparse categorical variables using embedding vectors but for now just use one-hot encoding.

21. (2pt) Since we are not using Caret to train neural networks, We will have to manually remove variables with little or no variance. To identify these variables in your training data, use the method "nearZeroVar" from caret package. This will return the indices of variables with little to no variance in the training data. Use these indices to remove these variables from train, validation, and test data. You can refer to his page (https://topepo.github.io/caret/pre-processing.html#nzv , section 3.2) for an example of identifying and removing near zero variance.

22. (5 pt) Create a Neural Network model **with at least two hidden layers** to predict the saleprice in 100K units. In other words, your target variables/labels should be **SalePrice/100000.** We scale down the sale price to avoid error gradients to get too large during backpropagation. If gradients are too large, they can make the model unstable and you end up having NAN for training or validation loss.

Use the training and validation set you created above. Add a drop out layer after each hidden layer to regularize your neural network model. Use tfruns package to tune your hyper-parameters including the drop out factors. You should include two flags for the drop out factors, one for each hidden layer. Display the table returned by trfuns.

23. (2 pts) Use view_run to look at your best model**. Note that the best model is the model with lowest validation loss**. What hyper-parameter combination is used in your best model. Does your best model still overfit?
24. (2 pt) Now that we tuned the hyperparameters, we don't need the validation data anymore and we can use ALL of the training data for training. Use all of your training data (that is, train + validation data) to train a model with the best combination of hyper-parameters you found in the previous step.

25. (2pt) Use your model above to predict the saleprice in 100K units for the test data. To get RMSE in the original scale, you should multiply your predictions and test labels by 100000 before computing RMSE.

26. .(1pt) Compare the RMSE of your lasso, ridge, elastic net, random forest, gbm, svm, and neural networks models on the test data. Which model did better on this dataset?

Format of the submission

The submission must be in two formats:

- **A .html file which contains the preview of your notebook. When you click on preview in R studio to preview an R notebook, an html file is created in the same directory as your notebook.**
- **An .rmd file of your notebook**
- **Any additional R script you used for creating your neural network models.**