Number of Article Mashable Shares Prediction

Data Set:Online News Popularity Dataset

EE 559 - Machine Learning I: Supervised Methods
(including Mathematical Pattern Recognition)

Date of submission: May 07, 2021

Table of Contents

1. Abstract

With the increasing usage of social media, now we can utilize plenty of data to predict how popular any article will be if they posted on some websites. The goal of this project is to develop a machine learning system that operates on the given real-world dataset with a set of features and predict the number of article mashable shares using the given data set from UCI machine learning website.

In the following report, we introduced 8 models (Lasso, Ridge, SVR, KNN Regressor, Gradient Boosting Machine, Random Forest Regressor, Multi-layer Perceptron (MLP) and Stochastic Gradient Descent (SGD) to make predictions and compared their performance with the baseline system (Linear Regression) and trivial system (system that will always output the mean value of the target values). The metrics here we used to evaluate the model performance are pMAE, pMSE and mR2 scores due to the sparsity (the imbalanced distribution) of the target values.

Based on the model we selected to train, we used validation techniques to conduct the hyper-parameters tuning so that the model could be well trained and provide better outcomes. Among the 8 models we selected to train, a few of them did make excellent predictions with proper data preprocessing. We made analysis on why some of the rest of the models provided less satisfying predictions.

Among all of the models, we decided to choose KNN Regressor as our final model. Compared with the baseline model which implements the Linear Regression, pMAE, pMSE and mR2 score are all significantly improved on the test data set.

Finally, we made a conclusion and provided insights on what future works can be done.

2. Introduction to the Dataset

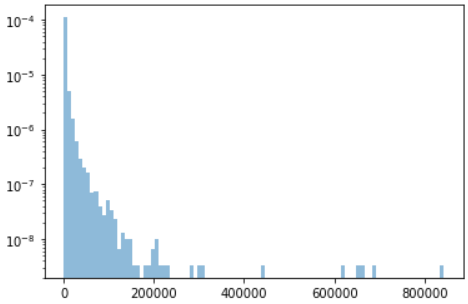
2.1 Problem Statement and Goals

Data set given is not raw. This processed dataset has 58 float-type predictive features and there are no null values. The only one output (target) is an integer value representing the number of shares of the article. Among the 58 features, there are 2 nominal features (Day of the week and Article category), they have already been possessed as numerical features using one-hot encoding. We can see that the number of shares has the biggest range which is : 843300.

Min-max value and range of some features are as follows.

features	Self reference shares	Number of words in the article	Number of non-stop unique words	Number of unique words	Number of hrefs	Number of images
min value	0	0	0	0	0	0
max value	843300	8474	850	701	304	128
range	843300	8474	850	701	304	128
features	**Number of self hrefs**	**Number of videos**	**Number of worlds in the title**	**Number of keywords**	**Average token length**	
min value	0	0	2	1	0	
max value	116	91	23	10	8.04	
range	116	91	21	9	8.04	

If we take a look at the output value of our training data set and plot using log scale, we can see that the distribution of the target value is not in balanced distribution.



The goal of this project is to predict the shares of each article based on a set of features provided.

3. Data Preprocessing

In total, we have 58 features with 35644 data points. The main challenge here is imbalanced distribution of the feature space. We won't consider the degree of freedom is too large for most of our models. But we want to consider whether processed nominal data is meaningful for the total shares prediction.

3.1 Log Scaling

By observing the min, max of each feature space, we are aware that some features can have a large range of distribution; max value can sometimes be 1000 greater than the min value. And by observing the 25%, 50% and 75% quartile of the features, we can see an imbalanced distribution of some features even though their max value minus min value are not very large or even not large at all.

In order to smooth the distribution and avoid some "outlier" feature value to dominate our model, log scaling was performed. The process is as followed:

1. Observe the distribution of each feature by calculating the (max-min) value.
2. Define the Threshold by checking whether such a feature is imbalanced.
3. Select feature that is not processed nominal features and has (max-min) value over Threshold.
4. Perform the log scaling on the entire feature column.

In the process of choosing the threshold, we initially chose 1000 as our threshold. However, we found this threshold won't include other features without large range but has imbalanced distribution, for example, some features have min value 0, max value 95, 25% quartile 0, 50% quartile 20 and 75% quartile 35 (we consider such feature distribution as imbalanced). Then we adjusted the threshold to 90 so that either feature with a large range or imbalanced distribution will fall into our log scaling process.

Applying our rule of selecting features performing log scaling, we now have our features need to do log scaling:

['n_tokens_content', 'n_unique_tokens', 'n_non_stop_words', 'n_non_stop_unique_tokens', 'num_hrefs', 'num_self_hrefs', 'num_imgs', 'num_videos', 'kw_min_min', 'kw_max_min', 'kw_avg_min', 'kw_min_max', 'kw_max_max', 'kw_avg_max', 'kw_min_avg', 'kw_max_avg', 'kw_avg_avg', 'self_reference_min_shares', 'self_reference_max_shares', 'Self_reference_avg_sharess']

Total features do log scaling = 20

The formula to perform the log scaling is $log(X_i + n)$. We defined a more specific rule for n based on the min value of feature X_i.

1. If min $(X_i) = 0$, n = 1 so that min $(log(X_i + n))$ match the min (original X_i) = 0.
2. If min $(X_i) < 0$, from the training set, we can see some X_i can have min value -1, so we set n = min (X_i) + 0.37 so that after log scaling, min $(log(X_i + n))$ will not become very small but still a negative value.
3. If min $(X_i) > 0$, just do the normal $log(X_i)$.

Next, we considered the range of the target value. We found the range of the target value is 843299, which can also be viewed as a large range output. Then we do log scaling on the target value to eliminate the outliers influences during the model training.

After performing the log scaling on the output value, the range then changed to 13.645078045562732.

Since log scaling is independent from training and test; performing log scaling for training and testing at the same time won't cause the leakage of the future unknown data.

3.2 Standardization

Our feature space contains nominal features and they've already been processed by one-hot encoding, that is to say, they only have either 1 or 0 feature distance, therefore, performing standardization on these features won't help model convergence.

We then only performed standardization on the features that are not nominal.

The standard score of a sample x is calculated as:

$$z = (x - u) / s$$

where u is the mean of the training samples, and s is the standard deviation of the training samples. [1]

3.3 Nominal Features Processing

Then we tried to have an understanding on the nominal features and to see if we can find valuable information to make adjustments to the feature dimension.

Consider the nominal features 'Day of the week' and bool feature 'Published on a weekend?':

There are total seven days of a week; By plotting the figure days vs. shares and isWeekend vs shares:

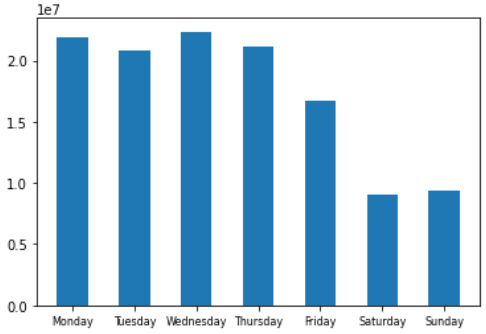


Figure: Days vs. Shares

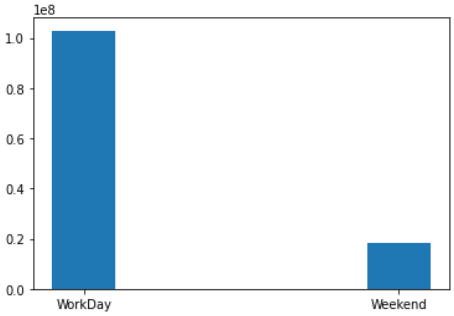


Figure: IsWeekend vs Shares

We can see that most article shares happen during the work days. However, we can't conclude that workday will have more influence on the shares by simply looking at the amount of the total shares based on the day. By calculating the average shares by day and by weekend, one can tell that the influence on the final shares prediction by day may not differ very much. The figures below provide the average shares based on day and based on whether it's workday or weekend.

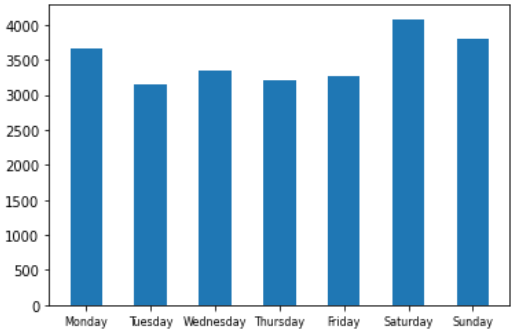


Figure : Day vs Average Shares

As we can see from the plot, each average share by day is around 3250. Therefore, using the original nominal feature processing (one-hot) can suffice to represent these nominal features.

Next, consider the nominal feature 'Article Category (Mashable Data Channel)':

If we add by columns corresponding to each channel type, we will find the sum of the total channels does not equal to the number of the total articles (number of our training data points). That is to say, some articles belong to the "other" channels. Plot the figure channels vs. total shares and figure channels vs. average shares below:

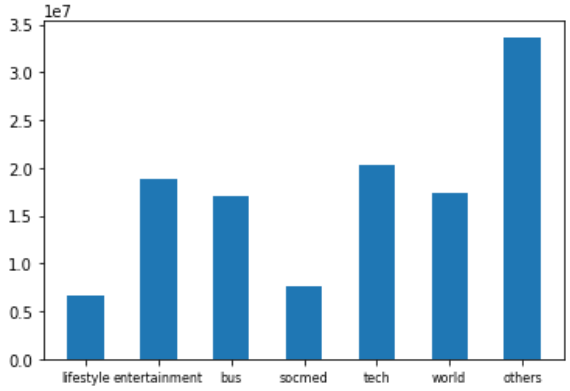


Figure : Channels vs. Total shares

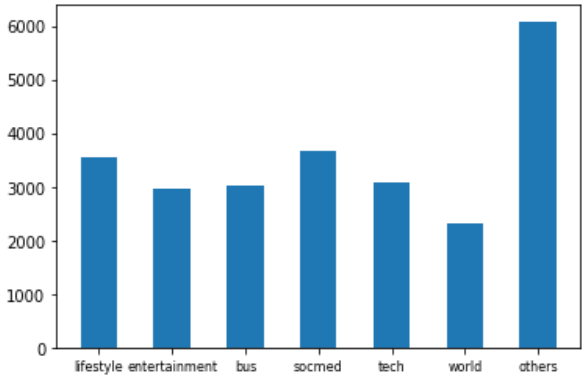


Figure : Channels vs. Average Shares

Although the total shares by channel differs a lot, if we took the average shares by the channel and neglect the shares by the 'other' channel, influence to the total shares prediction by channel again won't differ a lot. Since we don't know the specific types of channels within the 'other' channels, here we decided not

to update the feature space of the nominal feature 'article category' by replacing them with the average shares by channel category.

In conclusion, we decided to keep using these processed nominal features without expanding or reducing the original ones.

3.4 Data Processing Procedures

1. Read in features and targets from our training data sets and testing data sets and make a deep copy of these datas (since we don't want to make any changes to the original data sets).
2. Perform log scale on the selected columns both on features and targets from training and testing.
3. We split the original feature space and target into a new train and validation data set for the purpose of the validation.
4. Perform step 2 on the split data sets
5. Do standardization on the split training set and use the scalar from the training data set to transform the validation data set.
6. Perform step 5 on the entire training set and transform the testing data set.

4. Different Approaches and Implementation

There are a total of 8 approaches we have tried to handle this news share prediction regression problem. We compare these 8 models' performance with the performance of the baseline model in the 6th section. In this section, we will mainly discuss the model details and how we use packages from sklearn to train our dataset.

To start off, the baseline model is implemented by deploying LinearRegression() from sklearn.linear_model.

1. LASSO

   Least Absolute Shrinkage and Selection Operator (LASSO) is used over regression methods for a more accurate prediction. It's a regularization technique that prevents the training model from overfitting. The cost function of LASSO is as followed:

$$\sum_{i=1}^{n}(y_i - \sum_j x_{ij}\beta_j)^2 + \lambda \sum_{j=1}^{p} |\beta_j|$$

Where, λ denotes the amount of shrinkage. The bias increases with increase in λ, variance increases with decrease in λ.

During the training process, data values are shrunk towards a central point as the mean. Lasso procedure tends to provide simple, sparse models. LASSO is a choice to try since we have 58 features and LASSO automatically performs feature selection.

To implement Lasso, we used Lasso from sklearn.linear_model. The default max_iter is 1000, it may cause the model not converging. Then we set the max_iter to 10000 since here we have data points over 30000. In the hyper parameter tuning section, we focused on tuning the parameter alpha, which is λ in the cost function.

2. Ridge

Ridge regression is a model tuning method that is used to analyse any data that suffers from multicollinearity. This model performs L-2 regularization (2 norm). When the issue of multicollinearity occurs, least-squares are unbiased, and variances are large, this results in predicted values to be far away from the actual values. Ridge prevents multicollinearity by shrinking and changing the parameters value of alpha to control the penalty term λ.

The cost function for Ridge regression:

$$\text{Cost} = \sum_{i=1}^{N} \left\{ y_i - \sum_{j=0}^{M} \beta_j x_{ij} \right\}^2 + \lambda \sum_{j=0}^{M} w_j^2$$

λ is the penalty term. λ given here is denoted by an alpha parameter in the ridge function.

When the value of alpha gets higher, the penalty also gets bigger and therefore the magnitude of coefficients is reduced. Unlike Lasso, ridge doesn't provide sparse weight space. Therefore, we can not use it as a feature selection method. We want to use Ridge for this regression problem since it will prevent some weights becoming too large and cause the model to be unstable.

To implement Ridge regression, we used Ridge() from sklearn.linear_model. For the purpose of comparing it with Lasso Regression, we set max_iter equals to 10000 as well. In the process of hyper parameter tuning, we focused on tuning parameter alpha, which is the penalty term in the cost function of Ridge Regression.

3. SVR

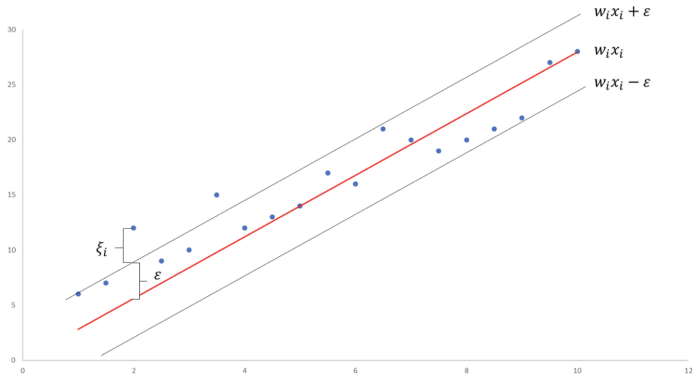
SVR gives us the flexibility to define how much error is acceptable in our model and will find an appropriate line to fit the data. The objective function of SVR is to minimize the L2-norm of the coefficient vector. The object function and the constraints are as followed:

$$MIN\ \frac{1}{2}||\boldsymbol{w}||^2 + C\sum_{i=1}^{n}|\xi_i|$$

,

$$|y_i - w_i x_i| \leq \varepsilon + |\xi_i|$$

The training samples influence the SVR model's fitting performance since the SVR algorithm is sensitive to the interference in the training data. Besides, SVR is useful in resolving high dimensional features regression problems. Thus, we also chose SVR as one of our attempts.



Illustrative Example of SVR with Slack Variables

We have the hyper parameter *C* and ϵ to tune. As *C* increases, our tolerance for points outside of ϵ also increases. From the previous implementation of SVM in homework 6, we found increased C and ϵ will increase the training performance. We deployed the package SVR from sklearn.svm and set the parameter C to 10 (since as C increases, the model training time will become extremely long).

In the process of hyper parameter tuning, we focused on tuning parameter gamma, which is ϵ from the SVR constraints.

4. KNN Regressor

K nearest neighbors is an algorithm that stores all available cases and predicts the numerical target based on a similarity measure like distance. The KNN algorithm uses 'feature similarity' to predict the values of any new data points. The prediction is the average or mean of relevant

neighbors. The figure below shows how KNN regressor uses our training data set to make predictions on the test dataset.

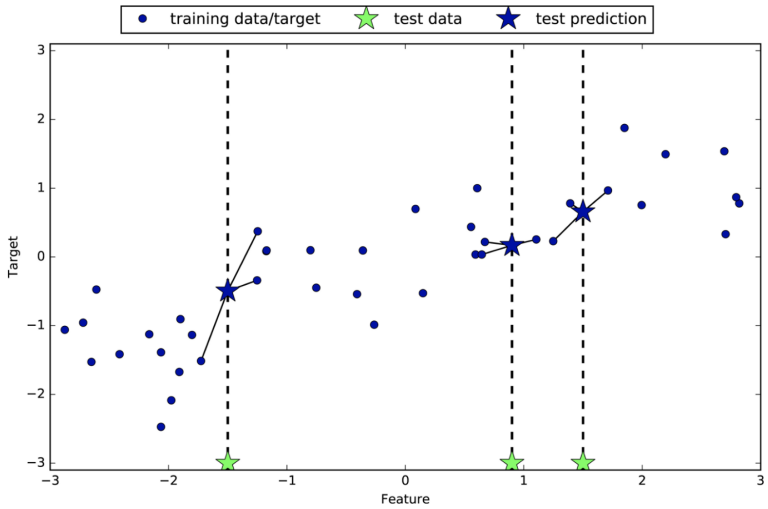


figure 2 . predictions make by three- nearest-neighbors regression on the wave dataset

There are many ways to calculate the distance between points. From sklearn.neighbors we import KNeighborsRegressor, and found it provides "uniform" and "distance" for the weights calculation. For example, 'distance' : weight points by the inverse of their distance. In this case, closer neighbors of a query point will have a greater influence than neighbors which are further away. In our project, we chose "distance" as the weights parameter since our training data set have a wide range and some "neighbors" are way bigger than the normal range.

During the hyper parameter tuning, we focused on tuning parameter n_neighbors so that this model can perform not bat when we make new predictions.

5. Gradient Boosting Machine

Gradient Boosting is an ensemble learner. Gradient Boosting will create a final model based on a collection of individual models. The predictive power of these individual models is weak but combining many such weak models in an ensemble will lead to an overall much improved result. In the Gradient Boosting machine, the most common type of weak model used is decision trees. During learning, XGboost learns from past mistakes.

The algorithm for Gradient Boosting machine is as followed:

Algorithm 1 Friedman's Gradient Boost algorithm

Inputs:

- input data $(x, y)_{i=1}^{N}$
- number of iterations M
- choice of the loss-function $\Psi(y, f)$
- choice of the base-learner model $h(x, \theta)$

Algorithm:

1: initialize $\widehat{f}_0$ with a constant
2: **for** $t = 1$ to M **do**
3: compute the negative gradient $g_t(x)$
4: fit a new base-learner function $h(x, \theta_t)$
5: find the best gradient descent step-size ρ_t:

$$\rho_t = \arg\min_{\rho} \sum_{i=1}^{N} \Psi\left[y_i, \widehat{f}_{t-1}(x_i) + \rho h(x_i, \theta_t)\right]$$

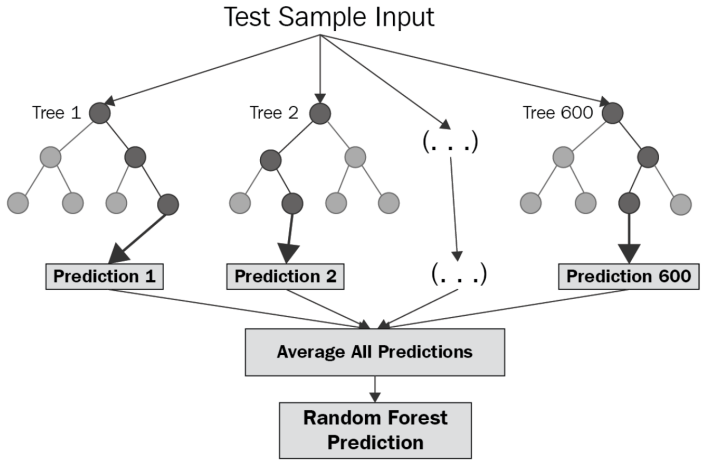
6: update the function estimate:
$\widehat{f}_t \leftarrow \widehat{f}_{t-1} + \rho_t h(x, \theta_t)$
7: **end for**

The disadvantage of the gradient boosting machine is prone to be overfitting. We implemented the gradient boosting machine from sklearn.ensemble.GradientBoostingRegressor, and in order to prevent the model from overfitting, we chose initial parameters as n_estimators= 150, loss = 'huber', random_state=5. Huber loss is a combination of the least absolute deviation and least squares regression since here we have a wide range of the target value.

In the process of hyper parameter tuning, we focused on tuning parameter max_depth.

6. Random Forest Regressor

Random forest is a Supervised Learning algorithm which uses ensemble learning methods for classification and regression. It is a bagging technique rather than a boosting technique. The trees in random forests are run in parallel. Below is the structure of Random Forest:



Random Forest Structure

Like Gradient Boosting Machine, Random Forest will also be prone to overfit. So, during the model training, we don't want the number of the trees to be large nor the max_depth of the single tree. We deployed sklearn.ensemble.RandomForestRegressor and set n_estimators = 150, bootstrap = False and random_state = 5, n_jobs = -1. The reason why we set bootstrap equal to false is that our target value has a wide range, if we randomly sample the original datasets, we might fall into the problem that we seldom have the chance to learn from the data with large target value.

During the process of hyper parameter tuning, we tuned the parameter max_depth since as the max_depth grows, the size of the tree will be bigger and it will have the problem of overfitting and the computational costs are quite expensive.

7. Multi-layer Perceptron (MLP)

A multilayer perceptron is a neural network connecting multiple layers in a directed graph, which means that the signal path through the nodes only goes one way.

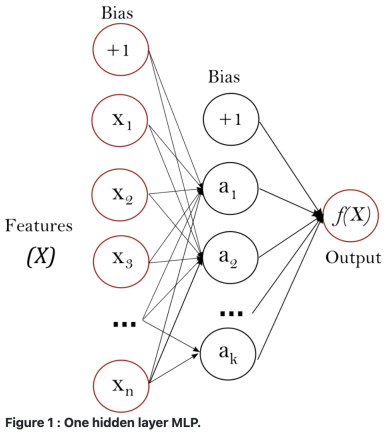


Figure 1 : One hidden layer MLP.

The disadvantages of Multi-layer Perceptron (MLP) include: MLP with hidden layers may provide several local minimums due to non-convexity of the loss function, and thus different initialization will lead to different performance. MLP requires tuning a number of hyperparameters such as the number of hidden neurons, layers, and iterations.

Our MLP model was implemented using sklearn.neural_network.MLPRegressor. We chose default parameters as hidden_layer_sizes= 75, max_iter =1000, random_state =5, activation = "tanh", learning_rate = "adaptive". The parameter we tuned was alpha, which is the coefficient of L2 regularization.

8. Stochastic Gradient Descent (SGD)

   This algorithm is useful in cases where the optimal points cannot be found by equating the slope of the function to 0. In SGD, it uses only a small sample of the entire data sets to perform in each iteration. And as the name stochastic indicates, it randomly chooses some sample data to train. It could provide faster convergence than the normal gradient descent techniques. In our SGD model, we set the loss as "huber" as we did for all other models to provide a more sound loss minimization.

   We implemented this model by import sklearn.linear_model.SGDRegressor. And in the process of hyper parameter tuning, we tuned alpha (as the term λ in the L2 regularization).

5. Hyper Parameter Tuning

Since our data set is relatively larger than the normal data set. Here, we considered using validation rather than cross validation to save the time of computation.

Before we performed validation, we first split our entire training set into 85% training and 15% for validation, and the sizes of the training and validation are:

training data has 30297 observation with 58 features

validation data has 5347 observation with 58 features

After train-test-split, we need to perform log on training features and target and validation training and target. And we can then use our new training to fit a standardization scaler than transform our new training data set and validation data set.

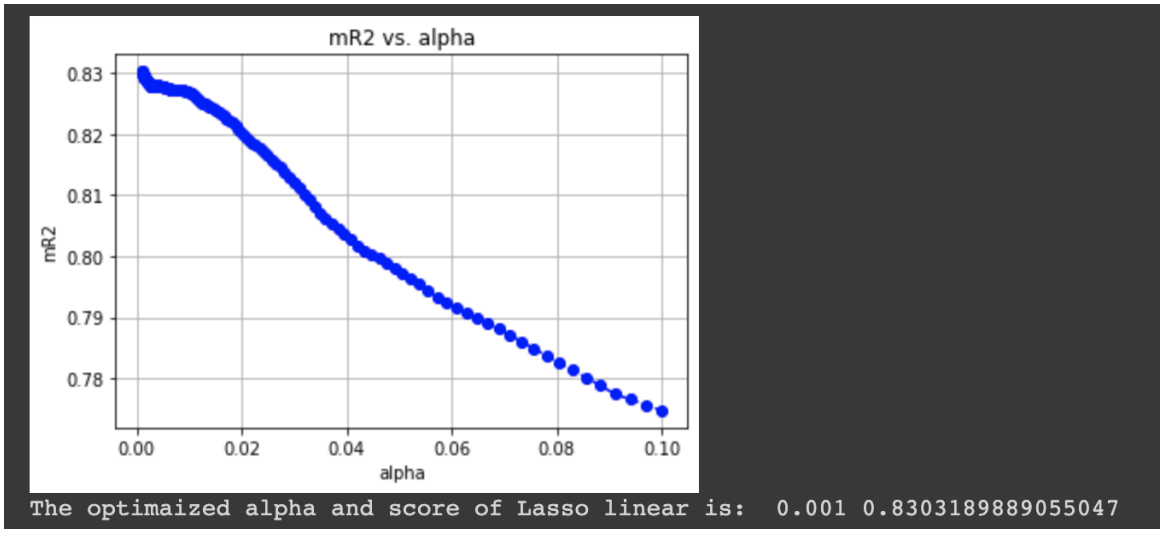
Then we performed the validation process by selecting potential parameters and training on the new training data set, then we used the model trained to make predictions on our validation set. Each time, we recorded the mR2 scores of the validation data set. Although the objects are the pMAE and pMSE scores, here we wanted the model with the highest mR2 score for choosing the model that can best interpret the future test set. And the truth is, when the mR2 score is higher, the pMAE and pMSE scores on the test tend to be lower than the baseline system. The following are the parameters we tuned for the specific model and how their mR2 scores on the validation set changes vs various parameters.

1. LASSO

   Hyper parameters to tune: alpha

   Range: alphas = np.logspace (-3, -1,num=150)

   Results:

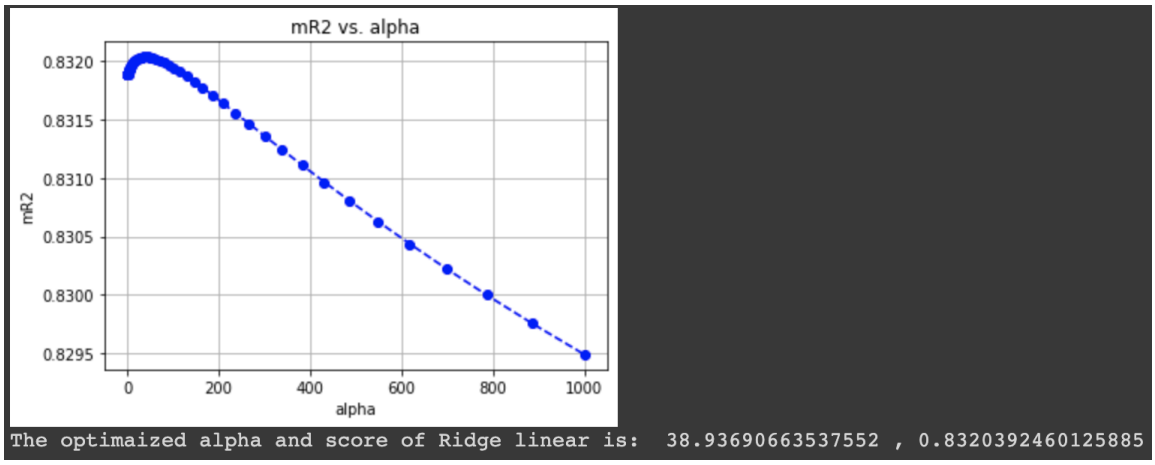


2. Ridge

   Hyper parameters to tune: alpha

   Range: alphas = np.logspace (-10, 3, num=250)

   Results:

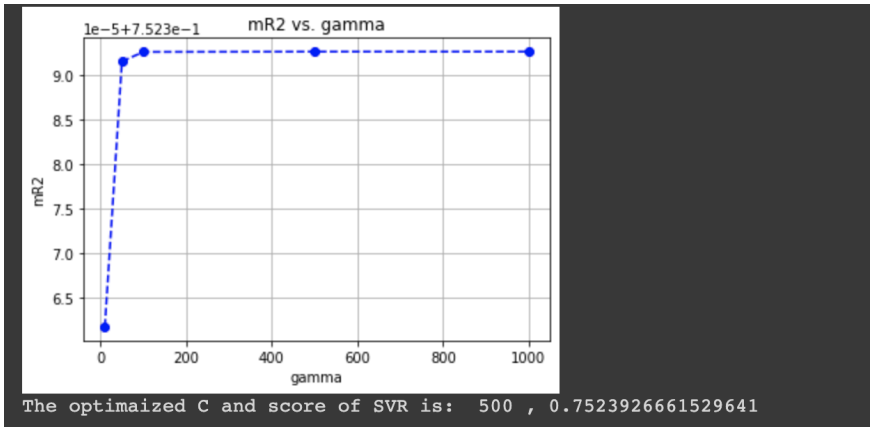


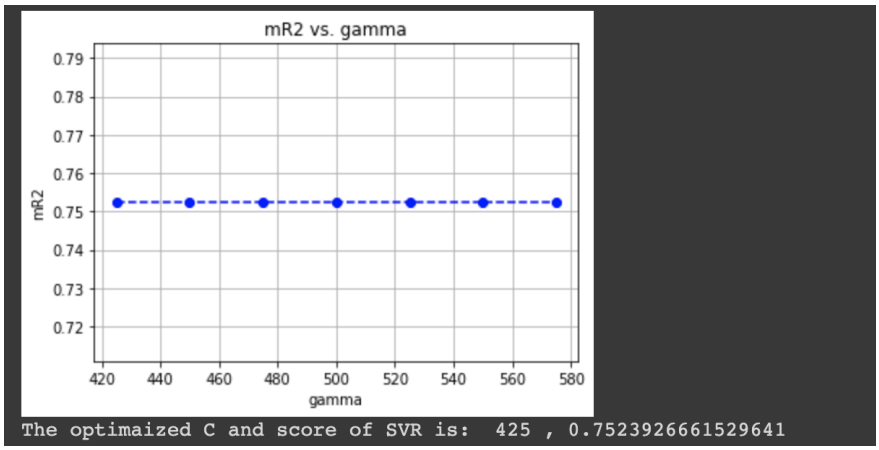
3. SVR

   Hyper parameters to tune: gamma

   Range: first use C = np.array([10, 50, 100, 500, 1000]) to roughly choose the region of the C falls in, then use C = np.array([425,450,475, 500, 525,550,575]) to choose C more precisely.

Results:



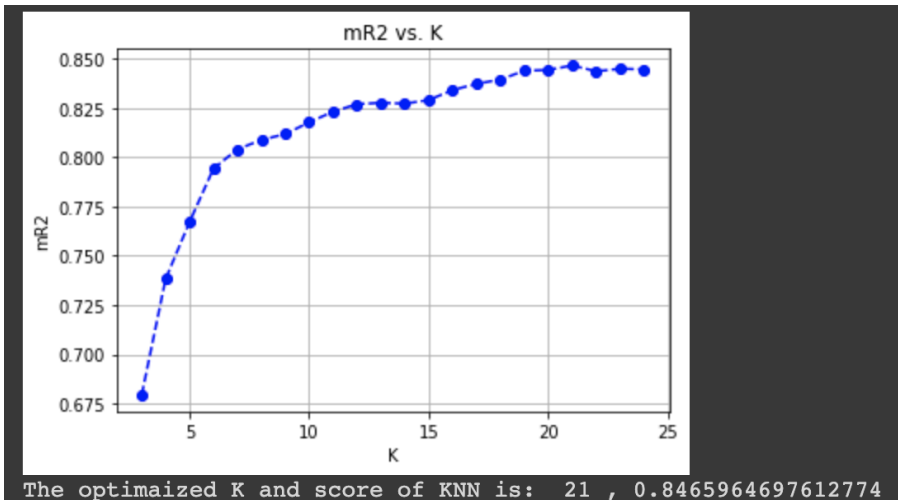


4. KNN Regressor

   Hyper parameters to tune: n_neighbors

   Range: K = (3, 25)

   Results:

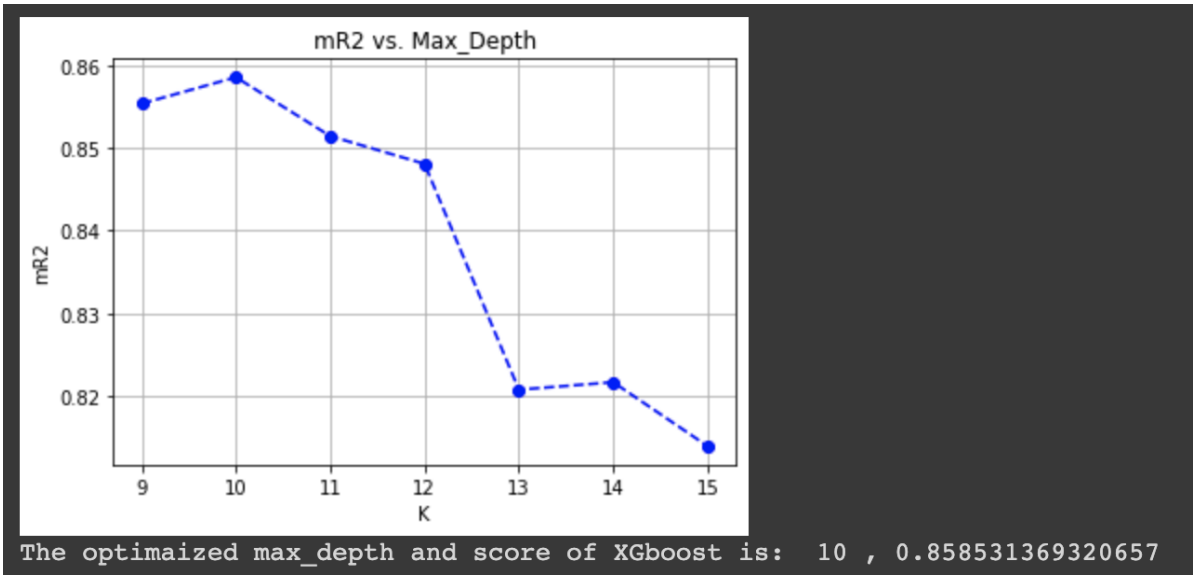


5. Gradient Boosting Machine

   Hyper parameters to tune: max_depth

   Range: np.array(list(range(9,16)))

   Results:

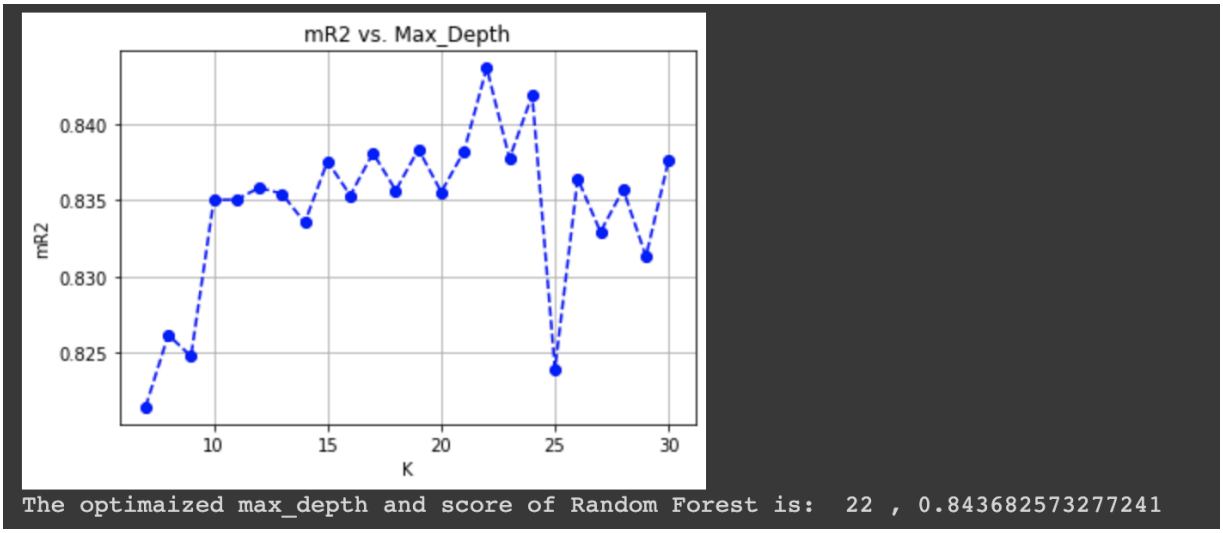


6. Random Forest Regressor

   Hyper parameters to tune: max_depth

   Range: K = np.array(list(range(7,31)))

   Results:

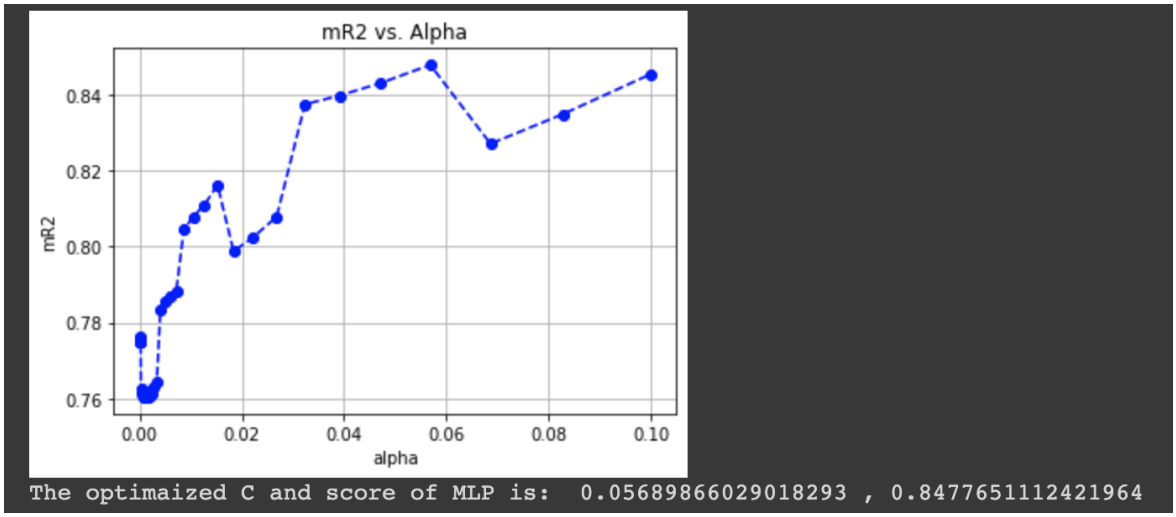


7. Multi-layer Perceptron (MLP)

   Hyper parameters to tune: alpha

   Range: alphas = np.logspace (-5, -1, num=50)

   Results:

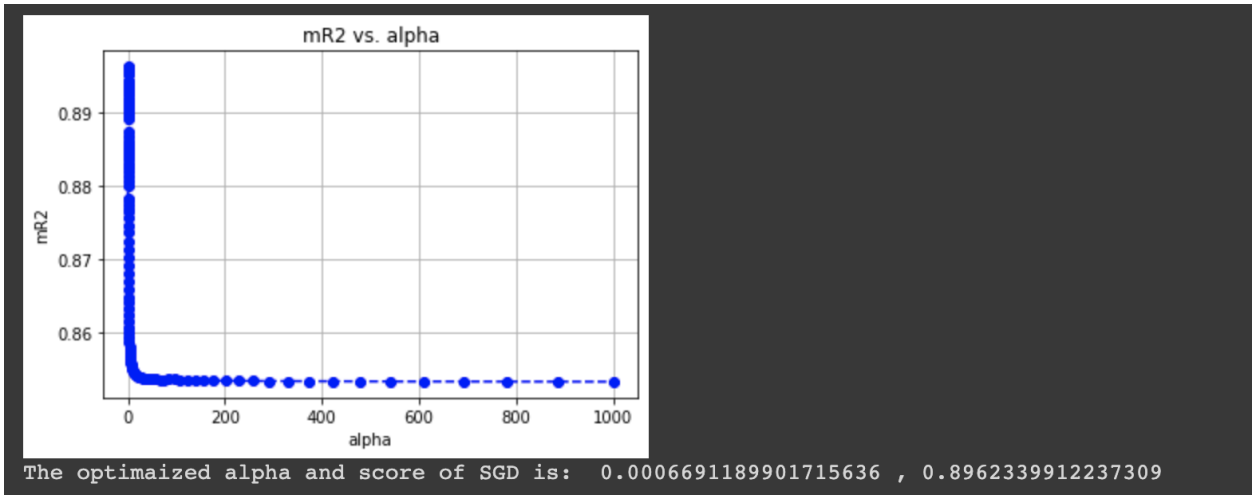


8. Stochastic Gradient Descent (SGD)

   Hyper parameters to tune: alpha

   Range: alphas = np.logspace (-5, 3, num=150)

   Results:



6. Model Performance Evaluation

After the validation process, we then obtained the tuned hyperparameters for the better performance of our training model. In this section, we used tuned parameters to train our entire training data set and calculated the pMAE, pMSE and mR2 scores both on the training data set and the testing data set.

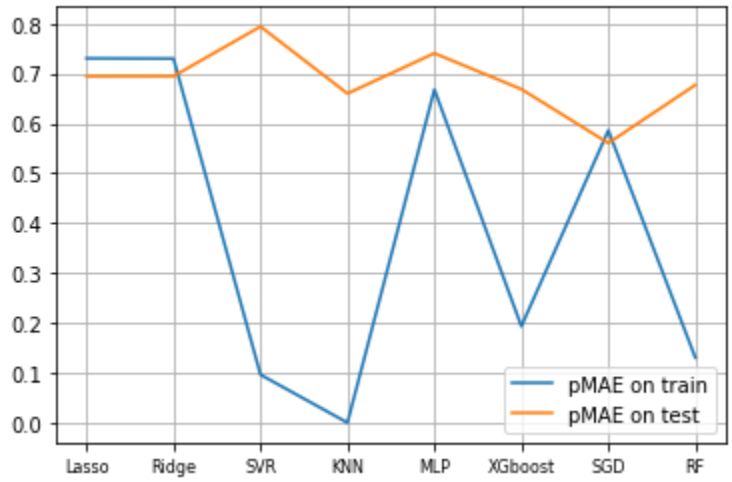
For the baseline, we have LinearRegression model and its performance on train and test data set are:

Train: pMSE: 26.141994095580547
pMAE: 1.655254669345867
mR2: -0.40353362598950593

Test: pMSE: 6.814368200854747
pMAE: 1.5730983197320147
mR2: 0.217480058603865

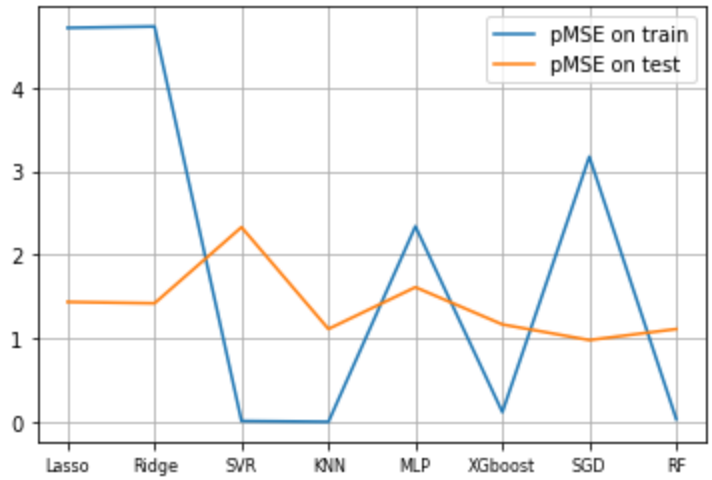
We plot the figures for easy comparison of each model's performance.

pMAE:



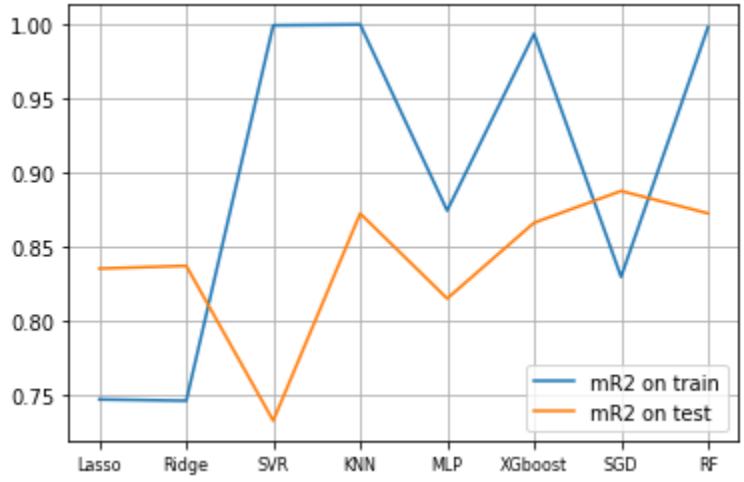
As we can see from the figure above, almost every model has a lower pMAE on the training dataset than the test set does. And all metrics were improved when comparing to the baseline model. It means that our model did explain our training data set to some extent. Although models like Lasso and Ridge didn't perform well, I think it is because the training data set has a quite wider range than the test data set does, and these two models still cannot handle the extreme values quite well. And thus, I will consider these two models under-trained and won't be used for our final model.

pMSE:



From this figure above, we can see that pMSE on the test data set for each model is roughly in the range from 1 to 2. But pMSE on the training data set for each model varies quite a lot. It is because some linear models cannot handle well on the wide range of the original training data set and when our less-widely-ranged testing data set passed to this model, it still performs well and gives small pMSE.
Combined with the pMAE figure, we now only have SVR, KNN, GradientBoosting and Random Forest as our candidate models.

mR2:



mR2 describes whether the predictions match the observed values well. Therefore, the higher the mR2 is , the better the model interpretation of the data is. From this mR2 figure above, we have KNN and Random Forest that have the highest mR2 scores. But due to the limitation of the uploaded file size of the vocareum. We decided to choose KNN as our final best model.

We uploaded our trained model into the vocareum and then we have the output as followed, which we used for comparison of the baseline predictions:

Vocareum Results:

Baseline Model:

public_pMAE_score,11.924

public_PMSE_score,214.109

Our Chosen Best Model:

public_pMAE_score,0.829

public_PMSE_score,2.763

Again, the metrics indicate our selected best model has a good interpretation of the data both for the training data set and the testing data set by decreasing the pMAE and pMSE scores significantly.

8. Summary and Conclusions

During this project for news shares prediction, we learnt a lot of useful supervised machine learning techniques like Random Forest and Gradient Boosting Machine. The greatest achievement we gained was that if we were to try to make a prediction from the wide range of data sets from the real world, we should tighten ourselves simply by implementing the obvious models. As this data set provided for this project, it was previously processed by the natural language processings (NLP) and some categorical features have already been processed by one-hot encoding. These are all useful techniques when dealing with the non-numerical features. So as conclusion for this, we can alternatively try feature processing using NLP or other categorical feature engineering. Also, when choosing parameters, we can not only use cross-validation but also validation to save computational time.

And thanks to the sklearn library, it enables us to directly deploy hundreds of the models and multiple hyper parameters to tune for the best performance of the model for the specific problems.

What really surprised us, was that KNN when dealing with the imbalanced output data set, if hyper parameters are tuned properly, it can achieve as good performance as the model like Random Forest and Gradient Boosting Machine does.

If more time is permitted, we think that if we adjust the feature dimension in the proper way, the under-trained models like Lasso and Ridge should have a better performance than it did. And we could also try RBF neural networks to perform this regression task.

References

[1]. https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.StandardScaler.html

[2]. https://www.mygreatlearning.com/blog/understanding-of-lasso-regression/

[3]. https://www.mygreatlearning.com/blog/what-is-ridge-regression/

[4]. https://www.kaggle.com/residentmario/ridge-regression-cost-function

[5]. https://towardsdatascience.com/an-introduction-to-support-vector-regression-svr-a3ebc1672c2

[6]. https://www.analyticsvidhya.com/blog/2018/08/k-nearest-neighbor-introduction-regression-python/

[7]. https://medium.com/analytics-vidhya/k-neighbors-regression-analysis-in-python-61532d56d8e4

[8]. https://www.geeksforgeeks.org/ml-stochastic-gradient-descent-sgd/

[9]. https://en.wikipedia.org/wiki/Gradient_boosting#:~:text=Gradient%20boosting%20is%20a%20machine,prediction%20models%2C%20typically%20decision%20trees.

[10]. https://www.frontiersin.org/articles/10.3389/fnbot.2013.00021/full

[11]. https://medium.com/swlh/random-forest-and-its-implementation-71824ced454f

[12]. https://scikit-learn.org/stable/modules/neural_networks_supervised.html