Computational Machine Learning (COSC2793)

Assignment-2: Machine Learning Project

Student Name: Salina Bharthu Student No: s3736867

Contents

Table of Figures

Objective

The Objective of this document is to outline implementation, evaluation and comparison of machine learning algorithms that are used to predict total energy consumption.

Introduction

As a part of this project, we are given a dataset containing features such as temperature and humidity of different rooms, pressure, windspeed, visibility, etc. measured over the period of 6 months, at every 10 minutes along with total energy use. Initially, the data is explored to remove any inconsistencies followed by understanding the relation of features among themselves and their importance with dependent variable. This supervised machine learning problem to predict the total energy usage is implemented using 3 different models that are Support Vector Machine, Gradient Boosting Machine and Long Short Term Memory. The results are evaluated and compared to select the best suitable model to predict the total energy usage.

Methodology

Data Extraction and Exploration

Initially data is extracted and checked against inconsistent and missing values. The timeseries data have approx. 20k observations, which is measured every 10 minutes that can be used to predict energy usage. Here, the granularity of the data needs to be reduced via up sampling as there are 6 observations every hour which does not provide any significant observation for energy usage pattern. Therefore, the data is resampled by every hour using mean value for all features. Figure 6- Time Series plot of energy before up sampling and Figure 7 - Time Series plot of energy after up sampling depicts that using mean value for resampling retains the actual behaviour pattern of the target variable and resampling can make the modelling more efficient by shrinking the observations without much information loss.

Feature Extraction and Feature Selection

After resampling the data hourly, I have extracted some features from date variable that can provide more information about target variable. The time series plot shows that it does not have any apparent trend. However, there are some hourly, daily, and monthly patterns (shows some seasonality) which can add value. Therefore, I have extracted hours, day, month, and weekend (0 for weekday and 1 for weekend) features from date followed by dropping the date column.
After adding these features, I have plotted boxplot for all variables that shows presence of outliers in many features such as T2, RH_5, etc. (However, few extreme outliers are managed by resampling data hourly). These outliers need to be managed as it can influence model performance by adding more variance. Also, summary statistics of the variables shows many independent variables that are having different scales (Press_mm_hg from 729 to 772 and T_out from -4 to 25) that needs to be fixed before fitting the data into machine learning model.

To deal with outliers and varying scale, I have used minmaxscaler which transforms the values in the range of 0-1 preserving the original distribution properties. Outliers are not removed completely in-order to retain the original behaviour of the features.

Now, to understand the correlation among variables, I have plotted covariance matrix. The matrix shows multicollinearity among many variables (0.9 correlation between RH_4 and RH_3). Also, there are few variables which are very poorly correlated with the target variable (such as -0.02 with rv1, 0.00 with T_9, etc.).

Now, to deal with multicollinearity, regularization is used. After separating dependent and independent variables into different dataframes, I have used LASSO regularization as it uses a penalty factor to retain important features which is advantageous as compared to other regularization models and suitable for non-linear models (we have non-linear relationship among features and target variable). LassoCV is used with list of alphas to get the model which generalizes well as low alpha can lead to underfit and high alpha can lead to overfit. Using the coefficient values calculated for each feature as per feature importance and correlation with other features, Lasso model picked 23 variables and eliminated 8(0 coefficient value) variables. Figure 8 - Feature Importance plot shows the coefficient values calculated by LASSO. To further remove the features contributing very less towards providing information about target variable, I have set a threshold of 10 (features with coefficient value more than 10) and -10 (features with coefficient value less than -10, high negative correlation) using coefficient values. For instance, Tdewpoint features is highly correlated to T9(0.79), from the plot it can be inferred that Tdewpoint is dropped and T9 is retained showing multicollinearity handling. Also, rv1 and rv2 both features are dropped as they are poorly correlated to Target energy (-0.02) showing feature selection based on importance to predict target variable. Here, we are left with 19 features which are further used to train the models.
After extracting important features, I have plotted them against the target variable using scatter plots. From that, non-linear relationship between independent and dependent variables can be inferred.

Data Modelling

Now, I have divided data into 3 different sets that are training, validation, and testing. The first split is for training-testing data (80:20) and second split is for training-validation data (60:40) applied on the training set left after first split. Training and validation data are used for model training and evaluation. Test set data is set aside, and models are tested for performance using that unseen data.

Model Selection and Evaluation

Implementation of SVM Regression

Now, to deal with the non-linear relationship, I have decided to implement Support Vector Machine Regressor (SVR). SVR uses kernel function that can transform the large number of non-linear features we have, into high dimensional space where they can be linearly separable. I have used gridsearchCV to find best parameter set as it generates the candidate models using all the parameters passed in the param_grid parameter. All candidate models are evaluated using score function of the estimator and 5-fold cross-validation. The model which gives the best mean score is picked and parameters used are returned as the best parameter set.

Table 1 - Hyper Parameter tuned for SVM

Parameter and its significance	Values passed	Best value
Kernel- Used to select the kernel function to transform features into suitable feature space	'rbf', 'poly'	‘poly’
C- Regularization parameter to balance between overfitting and underfitting (higher C can lead to overfitting)	1, 10, 100	100
Gamma - coefficient for kernel function	‘scale’, ‘auto’	‘scale’
Epsilon - used to define the error allowed per training data instance	0.01, 0.1, 0.5	0.1

The resultant parameter set is picked by considering mean value of the scores (accuracy) across all 5 splits. Now, to evaluate the model with parameters picked by gridsearchCV, I have plotted learning curve (train data size vs accuracy) and validation curve (model_complexity vs Accuracy) using 10-fold cross-validation.

Figure 9 - Learning Curve of SVM at split1 shows that the train and validation scores does not converge at the end, which means more training data is required. Therefore, I have changed the train-test split ratio (85:15) and train- validation split ratio (80:20). Figure 1 - Learning curve of SVM Regressor at split2 depicts comparatively lower accuracy difference and slightly better accuracy with high amount of training data.

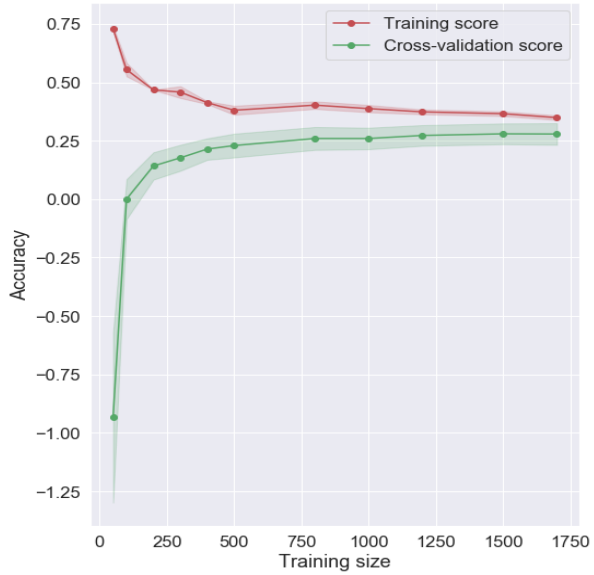

Figure 1 - Learning curve of SVM Regressor at split2

Figure 2 - Validation Curve of SVM Regressor at split2

For validation curve, I have used C value within the range of 0 to 150 to depict model_complexity as it is very important SVR regularization parameter that deals with overfitting and underfitting. Figure 10- Validation Curve of SVM at split1 with less training data ratio shows that train and validation scores converge at higher C value and validation scores are higher than train scores that can be occurred if the validation set does not properly reflect the properties of actual dataset. Figure 2 - Validation Curve of SVM Regressor at split2 with updated train-validation and test split ratio shows that, the validation and training scores converge somewhere around C value 60. The higher training accuracy with increasing model complexity shows that with higher C value the model tends to overfit. However, at C=100, which is the best value of C as per gridsearchCV, there is very minor difference between train and validation accuracy.

To further evaluate model, I have calculated MAE (robust to outliers and calculate regression loss) and train, validation, and test accuracy score (R2_score is also a good evaluation metric for regression models, but it is not recommended to use with nonlinear models and RMSE is not robust to outliers and in this dataset the outliers are not completely removed).

Table 2- the evaluation scores for SVM on both train-validation-test splits

	MAE (validation data)	Train Accuracy (Validation data)	Test Accuracy (validation data)	MAE (unseen data)	Test Accuracy (unseen data)
Split-1	37.22	0.35	0.31	36.91	0.34
Split-2	34.17	0.34	0.35	40.08	0.33

The MAE of validation data is reduced after increasing the training data size and increased for unseen test data. Also, there is very less difference between train and test scores, that shows that model does not suffer from overfitting.

Implementation of Gradient Boosting Regressor

Gradient boosting is an ensemble learning model which boost the performance of model by combining multiple weak learners to build a strong predictor. Decision trees are used as a week learner here. The model adds new learners gradually and sequentially while considering the existing ones to learn from

their mistakes. In gradient boosting, residuals of the previous weak learner are used as an input in next learner that makes it suitable for regression problems like prediction.

As GBM is very flexible by providing several hyper parameter tuning options, I have first decided to tune tree-based hyper parameter (max_depth, min_max_split) using gridsearchCV followed by boosting hyper parameters.

Table 3 - Hyper Parameter tuned for GBM

Parameter and its significance	Values passed	Best value
Max_depth - maximum number of levels(splits) a tree can have (higher value leads to overfitting)	Range (1,4,1)	3
Min_samples_split - minimum number of observations to be considered for split	Range (100,500, 20)	100
learning_rate – To control the rate of gradient descent convergence. Small value can reduce overfitting but take longer to reach optimal fit	0.01, 0.05, 0.1, 0.3, 0.5	0.05
Loss – Loss function to be minimized	'ls', 'lad'	ls
n_estimators – number of trees to be modelled. Very high value can lead to overfitting	25, 50, 80	50
Subsample – fraction of observations to be selected for each tree	0.7, 0.8, 1	0.8

The model is trained using the best parameter set picked by gridsearchCV followed by plotting learning and validation curves to evaluate the performance.

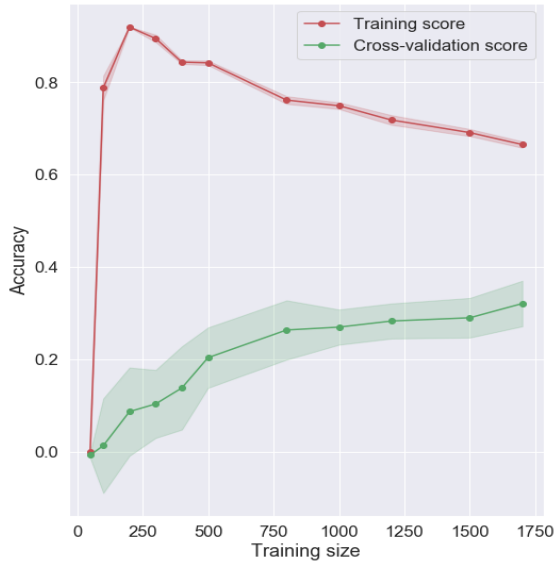

Figure 3 - Learning Curve of GBM Regressor at split2

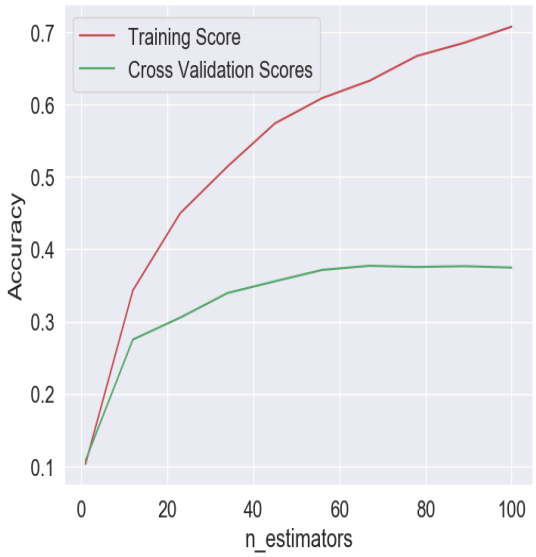

Figure 4 - Validation curve of GBM Regressor at split2

The learning curves of both splits show that, train and validation scores does not converge, even after increasing train data size. The sudden increase in test accuracy after reaching 100 observation is because of min_sample_split parameter which is set at 100. Also, the validation curve depicts that, with increasing model_complexity (high value of n_estimators that shows the number of trees), the train score is increasing, and validation score is stable after some increment. The learning and validation curve behaviour depicts the overfitting.

Table 4 -The evaluation scores for GBM Regression model for both splits

	MAE (validation data)	Train Accuracy (Validation data)	Test Accuracy (validation data)	MAE (unseen data)	Test Accuracy (unseen data)
Split-1	42.87	0.56	0.30	42.69	0.29
Split-2	38.57	0.65	0.39	41.30	0.44

The MAE for validation and test data increased after adding more training data in split2. The high difference between train and test score shows that model overfits the data.

Implementation of LSTM Model

Here, we have timeseries data of energy usage along with other independent features. The energy consumption at particular time can have some long-term dependencies (for instance, energy usage/temperature or humidity before 24 hours) or some short-term dependencies (for instance, feature values before 1 hour). In such cases using deep learning LSTM (Long Short Term Memory) model can give more accurate predictions considering some useful data from past and dropping some unnecessary data from current input. LSTM is an advance architecture of RNN (Recurrent neural networks) that deals with the problem of exploding and vanishing gradient problem (when using backpropagation in very deep RNN, the gradient used to calculate the weights can become very large or very small). LSTM have memory blocks that are connected through layers with three gates attached to it, that are forget gate (which information to discard), input gate (useful value from input to update memory unit) and output gate (output value using input and memory unit).

For implementation of deep neural network with LSTM, feature selection is not performed as LSTM can find the right features from large chunk of data. I have used hourly resampled and normalized timeseries data (normalized data with mean close to 0 can speed up learning and converge quickly) and divided it into train, validation and test splits followed by reshaping them into 3D array of [samples, timesteps, features]. Timesteps value is set to 1, for 1 lag at each observation. I have used sequential model containing Input LSTM layer, two LSTM layers as hidden layer (Considering high dimension of features and complexity of the problem) and dense layer with one node (suitable for regression problem) as output layer. The parameters of LSTM layers (activation function and number of neurons) are tuned using gridsearchCV. Also, I have used adam optimizer to compile the model as it is a good choice to balance between fast training and avoiding stuck in local minima by using past gradients to speed up learning. The learning rate of adam is also tuned using gridsearchCV. Moreover, the model parameters such as batch size and epochs are also tuned.

Table 5 - Hyper parameters tuned for LSTM

Parameter and its significance	Values passed	Best value
Batch size - the number of samples to be considered before a weight update of the network	128, 264, 512	128
Epochs - Number of times the data will be passed to the neural network	500, 1000, 1500	1000
Learning rate – Adaptive learning rate for different parameters	0.001, 0.01, 0.1, 0.3, 0.5	0.1
Neurons – Number of neurons to be used in layers	30, 50, 80, 100	100

In LSTM the default output activation function ‘Relu’ is used, that is computationally efficient than tanh and sigmoid functions as it does not activate all the neurons at the same time considering the output of the linear transformation. While training the LSTM model, the Callback function early stopping is also used in-order to prevent model to run for many epochs with no improvement in accuracy. For early stopping, if for next 80 epochs, the validation loss value does not change more

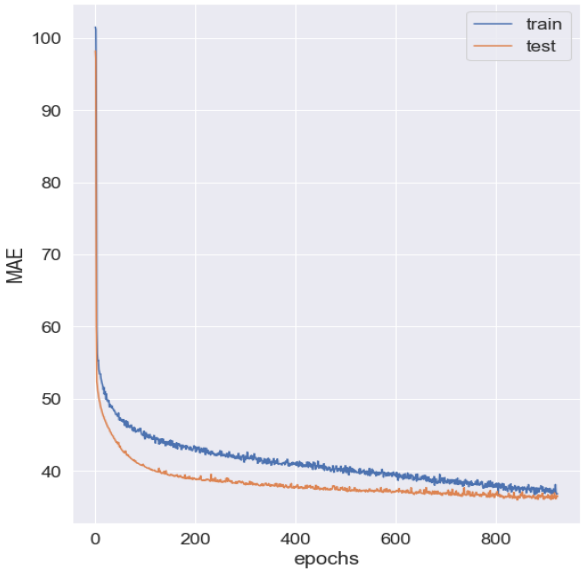

Figure 5 - Validation Curve of LSTM

than 0.001, the model execution is interrupted. I have used loss function and metric as MAE as it is suitable for regression problems. The model is trained using best parameter set found and validation curve is plotted. There is slow decay in training and validation loss and training loss with increasing epochs and it converge around 1000 epochs.

The MAE value for train data is 34.88, validation data is 36.52 and test data is 39.61.

Conclusion

The evaluation metric Mean Absolute Error obtained for all 3 implemented models are very close to each other. However, from learning curve, validation plot and train (0.65) – test (0.44) accuracy score of GBM Regressor, it can be inferred that the model overfits the data. This limitation is handled well by SVM and LSTM, both models provide comparatively similar and well prediction power and can be implemented in real world practices. However, By looking at Figure 11 - Actual Vs Predicted Scatter plot of LSTM and Figure 12 - Actual Vs Predicted Scatter plot of SVM, It can be seen that LSTM generalizes the data well as compared to SVM. The line of fit is comparatively more diagonal for LSTM. Also, the MAE on unseen data for LSTM (39.61) is slightly less than MAE of SVM (40.08). Therefore, for this experiment LSTM performs comparatively well.

Limitation and Future Work

All three model does not provide high prediction power as this task uses limited number of hyper parameters with limited range because of computational limitations. For instance, degree is very important parameter of SVM with of 'poly' kernel and it requires high computational power and longer run time. Also, the LSTM model parameters are tuned one at a time here, if they can be tuned together by gridsearchCV, it can yield better resultant parameter set. The predictive power can be enhanced by tuning more hyper parameters and considering large volume of training data.

References

- *sklearn.svm.SVR — scikit-learn 0.23.1 documentation*. Scikit-learn.org. (2020). Retrieved 29 May 2020, from https://scikit-learn.org/stable/modules/generated/sklearn.svm.SVR.html.
- *1.11. Ensemble methods — scikit-learn 0.23.1 documentation*. Scikit-learn.org. (2020). Retrieved 29 May 2020, from https://scikit-learn.org/stable/modules/ensemble.html?highlight=gbm#gradient-tree-boosting.
- Team, K. (2020). *Keras documentation: LSTM layer*. Keras.io. Retrieved 29 May 2020, from https://keras.io/api/layers/recurrent_layers/lstm/.
- *MyApps Portal*. Rmit.instructure.com. (2020). Retrieved 29 May 2020, from https://rmit.instructure.com/courses/67399/pages/week-10-learning-materials-slash-activities?module_item_id=2160990.
- Brownlee, J. (2020). *How to Grid Search Hyperparameters for Deep Learning Models in Python With Keras*. Machine Learning Mastery. Retrieved 29 May 2020, from https://machinelearningmastery.com/grid-search-hyperparameters-deep-learning-models-python-keras/.

Appendix

Plots for detailed understanding

1. Time Series plot of Target Variable Before and After Resampling

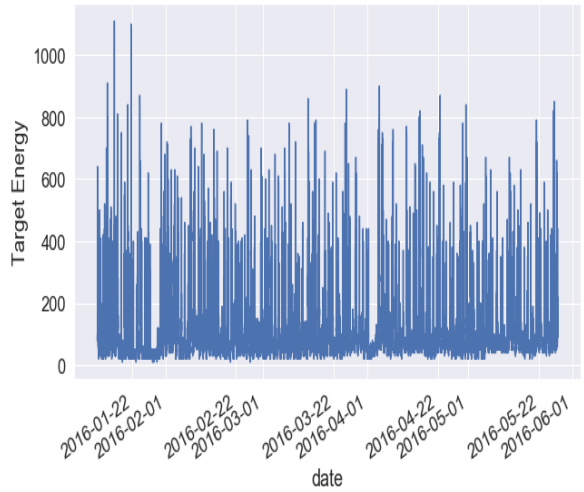

Figure 6- Time Series plot of energy before up sampling

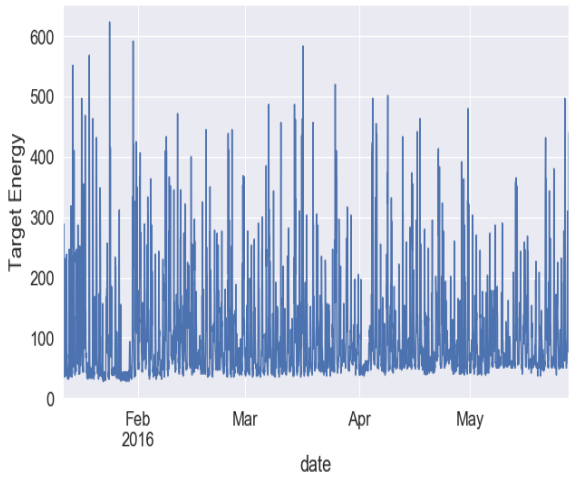

Figure 7 - Time Series plot of energy after up sampling

2. LASSO coefficients showing feature importance

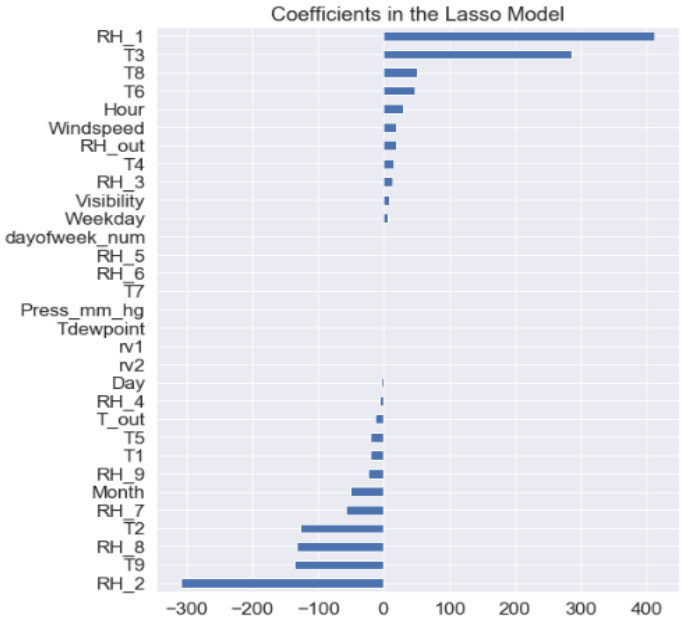

Figure 8 - Feature Importance plot

3. Learning and validation Curve for SVM Regressor at split1(train-test – 80:20 and train-validation – 60:40)

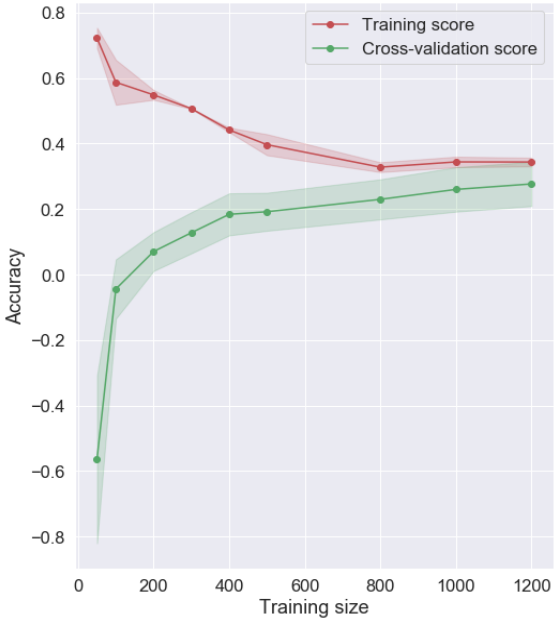

Figure 9 - Learning Curve of SVM at split1

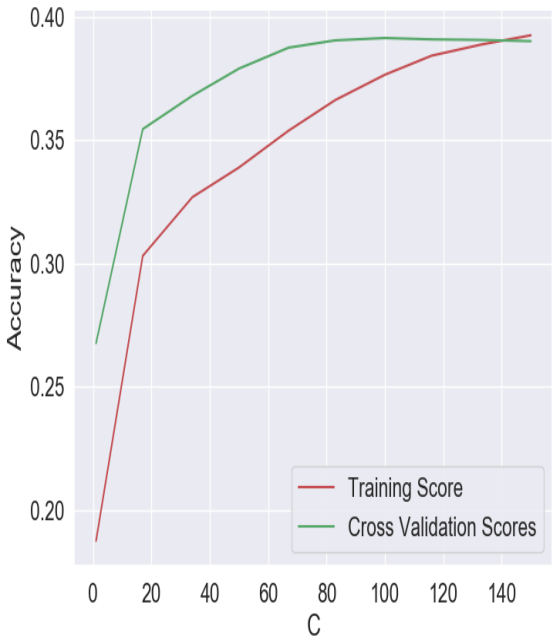

Figure 10- Validation Curve of SVM at split1

4. Actual Vs Predicted graph of SVM and LSTM

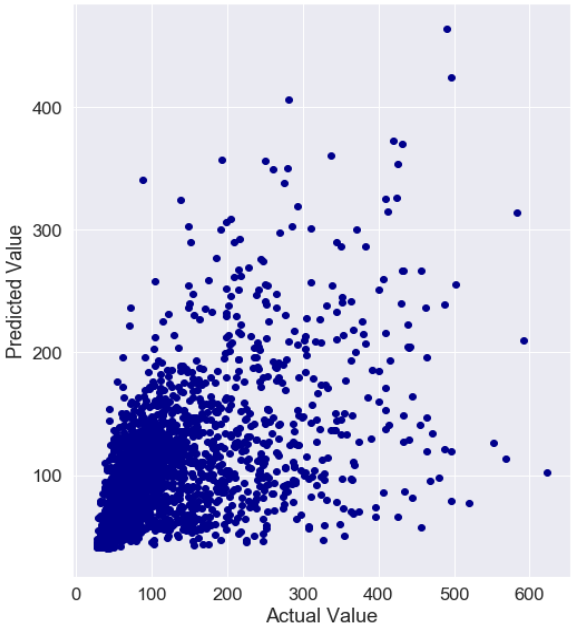

Figure 11 - Actual Vs Predicted Scatter plot of LSTM

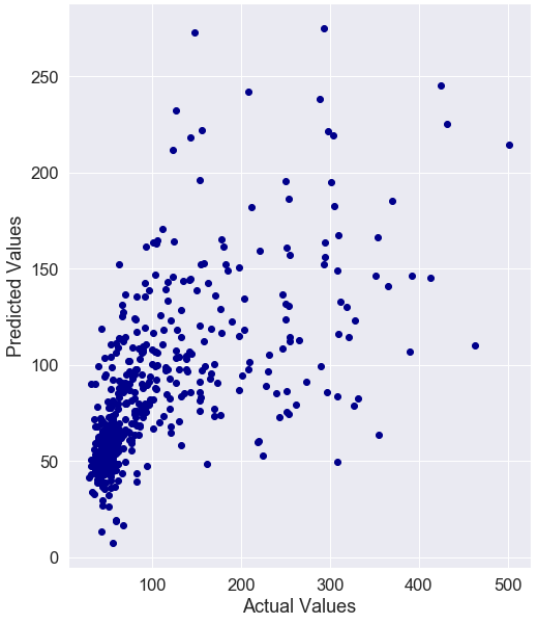

Figure 12 - Actual Vs Predicted Scatter plot of SVM