Analysis of Rossmann Store Sales Prediction

Parag Bhingarkar
Northeastern University
Bhingarkar.p@husky.neu.edu

Pratik Kadi
Northeastern University
Kadi.p.@husky.neu.edu

***Abstract* - In this project, we are applied machine learning techniques to a real-world problem of forecasting the sales of Rossmann store and compared Time series model with regression models. Rossmann store sales prediction was a Kaggle competition. The aim of this project is to find out if time factor analysis on traditional machine learning algorithms outperforms time series analysis and forecasting with ARIMA. We trained four conventional regression models and compared them with ARIMA. We used feature selection, model selection to improve forecasting results. To compare our models, we used percentage error and accuracy.**

Index Terms -ARIMA, Random Forest, Boosting, Factor Analysis.

I. Introduction

Rossmann is a chain of drug store that operates in 7 European countries. We received 'Rossmann Germany store' sales data from Kaggle.com. The aim of this project is to find whether accuracy of prediction will improve after time factor analysis. Reason for opting this topic is that this is intuitive to understand. Product sales forecasting is vital topic of purchasing management. Forecast of sales is important in maintaining and determining stock levels and perfectly calculating future demands of products.
A machine learning algorithm has been widely used for prediction. By analyzing various vital parameters and other characteristics, we could predict more accurately.

II. Dataset

The publicly available dataset is used for this project and has been obtained from Kaggle. We are provided with historical sales data for 1,115 Rossmann stores. There total size of training data is ~ 1 million. Store information is given separately.

Date	Store	DayOfWeek	Sales	Customers	Open	Promo	StateHoliday	SchoolHoliday
2015-07-31	1	5	5263	555	1	1	0	1
2015-07-31	2	5	6064	625	1	1	0	1
2015-07-31	3	5	8314	821	1	1	0	1
2015-07-31	4	5	13995	1498	1	1	0	1
2015-07-31	5	5	4822	559	1	1	0	1

TABLE I

For training and testing the splitting percentage was 80-20%
To improve the accuracy of model we merged the training dataset with store information. Which further helped to improve accuracy and reduce the error rate.

1. Data Cleaning:
   Data cleaning was performed to remove all the null and missing values from the dataset. Since they could make model less efficient while prediction

III. Methodology

Supervised learning is defined as the task of learning the function that maps an input to an output based on an input-output pairs. In this project, several supervised learning models have been implemented.

A. Linear Regression: Linear regression is a linear approach to modelling the relationship between a scalar response (or dependent variable) and one or more explanatory variables (or independent variables). The case of one explanatory variable is called simple linear regression. For more than one explanatory variable, the process is called multiple linear regressions. [1] This term is distinct from multivariate linear regression, where multiple correlated dependent variables are predicted, rather than a single scalar variable.
Equation of multiple linear regression:

$$Y = \beta_0 + \beta_1 X_1 + \beta_2 X_2 + \ldots + \beta_n X_n$$

Where Y- Dependent Variable
B0- Intercept, B1….Bn – independent variables

```
reg = LinearRegression()
reg.fit(X_train , y_train)
pred = reg.predict(X_test)
```

```
rmse = np.sqrt(mean_squared_error(y_test,pred))
print(rmse)
```

```
0.3731435568682413
```

```
model = Ridge(random_state = 1)

param_grid = {
"alpha" : [0.01 , 0.2 , 0.250 , 0.3,0.5]
}
grid = GridSearchCV( model , param_grid , cv = 5 , scoring = "neg_mean_squared_error")
grid.fit(X,y)
```

```
GridSearchCV(cv=5, error_score='raise-deprecating',
       estimator=Ridge(alpha=1.0, copy_X=True, fit_intercept=True, max_iter=None,
   normalize=False, random_state=1, solver='auto', tol=0.001),
       fit_params=None, iid='warn', n_jobs=None,
       param_grid={'alpha': [0.01, 0.2, 0.25, 0.3, 0.5]},
       pre_dispatch='2*n_jobs', refit=True, return_train_score='warn',
       scoring='neg_mean_squared_error', verbose=0)
```

```
grid.best_estimator_
```

```
Ridge(alpha=0.5, copy_X=True, fit_intercept=True, max_iter=None,
   normalize=False, random_state=1, solver='auto', tol=0.001)
```

B. Random Forest: Random forests or random decision forests are an ensemble learning method for classification, regression and other tasks that operates by constructing a multitude of decision trees at training time and outputting the class that is the mode of the classes (classification) or mean prediction (regression) of the individual trees. Random decision forests correct for decision trees' habit of over fitting to their training set.

```
rf = RandomForestRegressor(n_estimators=15,max_depth=8)
rf.fit(X_train,y_train)
```

```
RandomForestRegressor(bootstrap=True, criterion='mse', max_depth=8,
           max_features='auto', max_leaf_nodes=None,
           min_impurity_decrease=0.0, min_impurity_split=None,
           min_samples_leaf=1, min_samples_split=2,
           min_weight_fraction_leaf=0.0, n_estimators=15, n_jobs=None,
           oob_score=False, random_state=None, verbose=0, warm_start=False)
```

```
rf_pred= rf.predict(X_test)
```

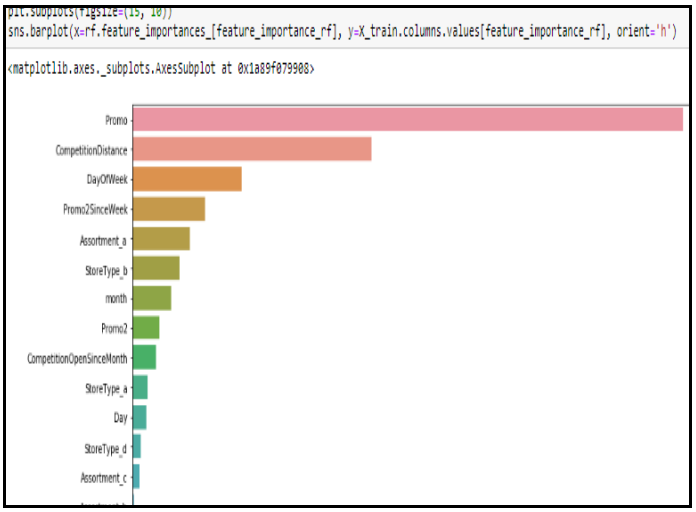
```
rmse_rf = np.sqrt(mean_squared_error(y_test,rf_pred))
rmse_rf
```

```
0.33776607192253
```

Visual representation of Importance of Features Which shows that promo and day of week makes are more significant when predicting the sales

```
plt.subplots(figsize=(15, 10))
sns.barplot(x=rf.feature_importances_[feature_importance_rf], y=X_train.columns.values[feature_importance_rf], orient='h')
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x1a89f079b8>
```



C. Gradient Boosting: Gradient boosting is a machine learning technique for regression and classification problems, which produces a prediction model in the form of an ensemble of weak prediction models, typically decision trees. It builds the model in a stage-wise fashion like other boosting methods do, and it generalizes them by allowing optimization of an arbitrary differentiable loss function. For papers published in translated journals, first give the English citation, then the original foreign-language one [6].

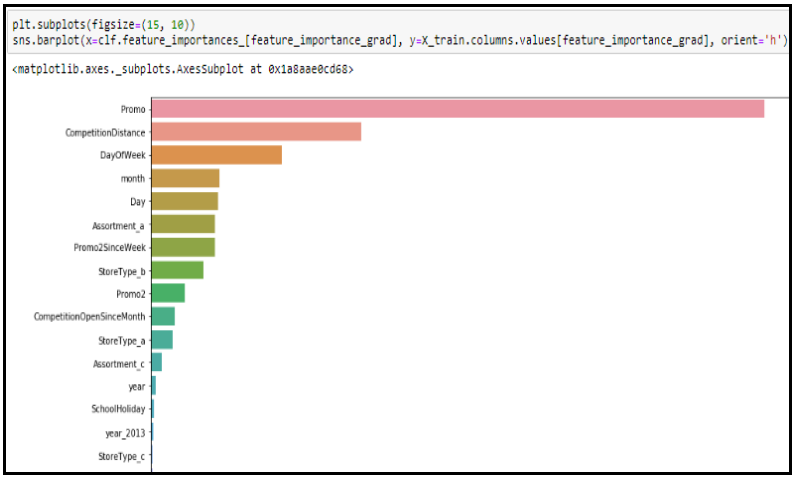
```
params = {'n_estimators': 400, 'max_depth': 4, 'min_samples_split': 2,
          'learning_rate': 0.01, 'loss': 'ls','criterion':'mse'}
clf = GradientBoostingRegressor(**params)
clf.fit(X_train, y_train)
```

```
GradientBoostingRegressor(alpha=0.9, criterion='mse', init=None,
             learning_rate=0.01, loss='ls', max_depth=4, max_features=None,
             max_leaf_nodes=None, min_impurity_decrease=0.0,
             min_impurity_split=None, min_samples_leaf=1,
             min_samples_split=2, min_weight_fraction_leaf=0.0,
             n_estimators=400, n_iter_no_change=None, presort='auto',
             random_state=None, subsample=1.0, tol=0.0001,
             validation_fraction=0.1, verbose=0, warm_start=False)
```

Visual representation of Importance of Features

```
plt.subplots(figsize=(15, 10))
sns.barplot(x=clf.feature_importances_[feature_importance_grad], y=X_train.columns.values[feature_importance_grad], orient='h')
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x1a8aae0cd68>
```



D. XGBoost: XGBoost is an optimized distributed gradient boosting library designed to be highly efficient, flexible and portable. It implements machine learning algorithms under the Gradient Boosting framework. XGBoost provides a parallel tree boosting (also known as GBDT, GBM) that solve many data science problems in a fast and accurate way. The same code runs on major distributed environment (Hadoop, SGE, MPI) and can solve problems beyond billions of examples.

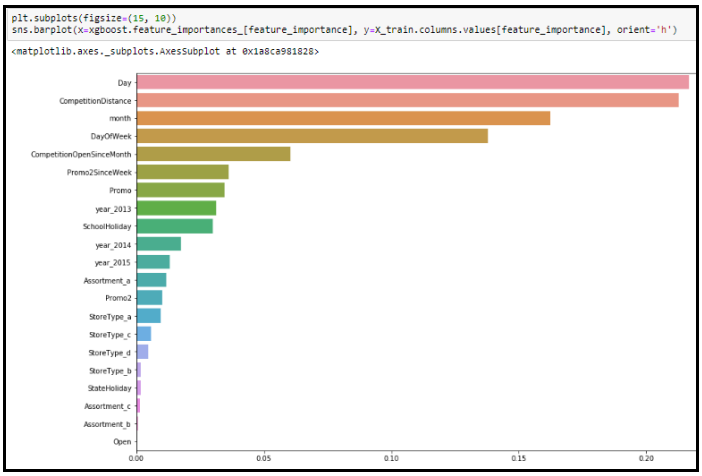
```
xgboost = XGBRegressor(max_depth=15,n_jobs=4,n_estimators=120,subsample=0.7)
xgboost.fit(X_train,y_train)
```

```
XGBRegressor(base_score=0.5, booster='gbtree', colsample_bylevel=1,
       colsample_bytree=1, gamma=0, learning_rate=0.1, max_delta_step=0,
       max_depth=15, min_child_weight=1, missing=None, n_estimators=120,
       n_jobs=4, nthread=None, objective='reg:linear', random_state=0,
       reg_alpha=0, reg_lambda=1, scale_pos_weight=1, seed=None,
       silent=True, subsample=0.7)
```

The important features are represented in the image below which represent that the day and month are very significant. Which makes sense since sale of any store will depend on day and month.

```
plt.subplots(figsize=(15, 10))
sns.barplot(x=xgboost.feature_importances_[feature_importance], y=X_train.columns.values[feature_importance], orient='h')
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x1a8ca981828>
```



E. Time series with ARIMA: Autoregressive Integrated Moving Average Model. An ARIMA model is a class of statistical models for analyzing and forecasting time series data. It explicitly caters to a suite of standard structures in time series data, and as such provides a simple yet powerful method for making skillful time series forecasts.

```
mod = sm.tsa.statespace.SARIMAX(y,
                                order=(3, 0, 0),
                                seasonal_order=(1, 1, 0, 12),
                                enforce_stationarity=False,
                                enforce_invertibility=False)
results = mod.fit()
print(results.summary().tables[1])
```

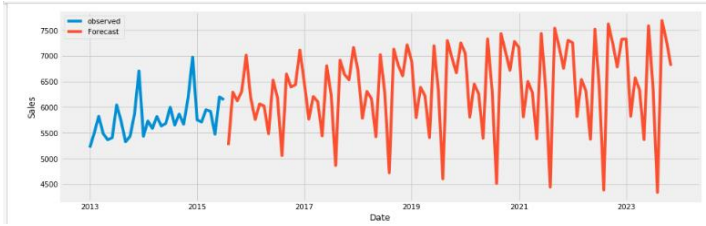
```
==============================================================================
                 coef    std err          z      P>|z|      [0.025      0.975]
------------------------------------------------------------------------------
ar.L1         -0.5968      0.000  -1395.666      0.000      -0.598      -0.596
ar.L2         -0.8976      0.000  -1816.841      0.000      -0.899      -0.897
ar.L3         -0.1258      0.000   -253.546      0.000      -0.127      -0.125
ar.S.L12       0.7890      0.000   2641.317      0.000       0.788       0.790
sigma2         0.0298   5.86e-06   5086.832      0.000       0.030       0.030
==============================================================================
```

```
y_forecasted = pred.predicted_mean
y_truth = y['2014-07-01':]
mse = ((y_forecasted - y_truth) ** 2).mean()
print('The Mean Squared Error of our forecasts is {}'.format(round(mse, 2)))
```

```
The Mean Squared Error of our forecasts is 58236.69
```

```
print('The Root Mean Squared Error of our forecasts is {}'.format(round(np.sqrt(mse), 2)))
```

```
The Root Mean Squared Error of our forecasts is 241.32
```



PREDICTION USING ARIMA MODEL

IV. RESULT

All the models are compared on the basis of RMSE and accuracy of prediction the sales. Since the data was time-series based we compared all other regression models with base models i.e ARIMA model. The RMSE for time series model is ~240. The results are shown in the table below-

Model Name	Accuracy	RMSPE
Linear Regression	63.50%	37.50%
Random Forest	67%	33.70%
Gradient Boosting	34%	34%
XG Boosting	89%	11%

V. CONCLUSION

From our experiments, we can conclude that the traditional machine learning algorithms such as XGBoosting and Random forest aid in more efficient prediction of sale compared to time series analysis with ARIMA. Even though the data was time dependent, we can say that by data wrangling we can apply supervised regression algorithm. After changing the shape of data and separating some columns it is possible to achieve better accuracy at predicting the dependent variable Sales. When we compared all the regression models with time series model, some models performed well and could yield accurate result. In the future, we also hope to explore the usage of neural networks in time series prediction, as they are also a widely used method for time-series prediction. It'll be challenging but at the same point interesting to compare their performance with current models.

ACKNOWLEDGMENT

We would like to thank our guide Prof. Nick Brown for his guidance and valuable inputs throughout the project.

REFERENCES

[1] http://www.stat.yale.edu/Courses/1997-98/101/linreg.htm
[2] http://blog.citizennet.com/blog/2012/11/10/random-forests-ensembles-and-performance-metrics
[3] https://en.wikipedia.org/wiki/Gradient_boosting
[4] https://github.com/nikbearbrown/INFO_6105/blob/master/Week_6/NBB_Decision_Trees.ipynb
[5] https://github.com/nikbearbrown/INFO_6105/blob/master/Week_6/NBB_Decision_Trees_Random_Forest.ipynb
[6] https://github.com/nikbearbrown/INFO_6105/blob/master/Week_5/NBB_Linear_Regression.ipynb
[7] https://www.analyticsvidhya.com
[8] https://www.kaggle.com/c/rossmann-store-sales
[9] https://xgboost.readthedocs.io/en/latest/