Predicting Sales for Rossmann Drug Stores

BRIAN KNOTT, HANBIN LIU, ANDREW SIMPSON

Abstract

In this paper we examined four different methods for time series forecasting: Random Forests, Gradient Boosting, Hidden Markov Models, and Recurrent Neural Networks. We found that using Gradient Boosting yielded the best results with root-mean-square percent error (RMPSE) of 10.439% (785 of 3429).

I. Introduction

The goal of Rossmann Kaggle competition is to forecast the daily sales of Rossmann stores located across Germany using store, promotion, and competitor data. It is known that store sales are influenced by many factors, including promotions, competition, school and state holidays, seasonality, and locality. A reliable and robust prediction model will enable store managers to create effective staff schedules that increase productivity. The data are provided by Rossmann through Kaggle.

II. Related Work

The models we use in this paper have been widely studied. The framework for Gradient Boosting is laid out in [3] and [4], Random Forests are studied in [2], Recurrent Neural Networks in [5], and Hidden Markov Models in [6]. These papers describe how parameters effect model prediction.

Additionally, 3429 groups have submitted results to this Kaggle competition. The code for many of these submission is available on the competition's Scripts page. The majority of the submissions used Random Forest or Gradient Boosting approaches, likely because there are reliable packages for their implementation. Results of these approaches vary widely, however, due to different choices in parameters, preprocessing, and data splitting.

III. Dataset and Features

Rossmann provided data with 15 features for 1115 stores. Not all stores have data for each feature on each day, so we filled in missing data with median data for non-ensemble methods (ensemble methods did not require data for every field). We then extended the data to 18 features since some features, like the date, captured multiple variables (like month of the year and day of the month). The resulting data had the following feature set:

Table 1: Feature Set

Ecohumo ID	Description
Feature ID	Description
1	Store ID
2	Day Of Week
3	Open / Closed
4	Promotion Active
5	School Holiday
6	Store Type
7	Assortment
8	Distance to Competing Store
9	Month Competitor Opened
10	Year Competitor Opened
11	Promo2 Active
12	Week Promo2 Started
13	Year Promo2 Started
14	Interval of Promo2
15	Month
16	Year
17	Day
18	Competition Age (GB only)

We then normalized the data to have zero mean

before applying our learning algorithms.

IV. Methods

I. Random Forests

Random Forest Regression is an ensemble learning algorithm that operates by aggregating many random decision trees to make predictions while avoiding overfitting. We started by using the Random Forest algorithm for black box prediction because its bagging techniques are robust to data anomalies (like missing data) and because random forest packages are widely available.

In particular, we used R's RandomForest package to carry out the training and prediction. We then used parameter optimization to improve on our prediction model. To see our parameter optimization method (and the parameters used) see the Results section.

II. Hidden Markov Models

Hidden Markov Modeling is a sequence / state estimation algorithm that assumes that the dataset derives from a Markov Process with hidden state information. Hidden Markov Models are appropriate for this dataset since its data evolves over time, a phenomenon unaccounted for in ensemble learning.

$$P(z_t|z_{t-1}, z_{t-2}, ... z_1) = P(z_t|z_{t-1})$$
 (1)

Hidden Markov models rely on two sets of Markov assumptions. The first is the limited horizon assumption. This assumes that the probability of being in a state depends only on the most recent states. For a first order Markov model, the current state z_t depends only on the state z_{t-1} (Eq. 1).

The second assumption is that the state transition process is stationary. This means that the conditional distribution of state transitions does not change over time (Eq. 2).

$$P(z_t|z_{t-1}) = P(z_2|z_1); t \in 2...T$$
 (2)

The transition between states can be estimated by averaging the number of times a state transitioned from another stated. We used Laplace smoothing to replace zero probabilities with a small probabilities (Eq. 3 where k = number of states).

$$\hat{p}(s_i|s_j) = \frac{\sum_{t=1}^{T} 1\{z_{t-1} = s_i \land z_t = s_j\} + 1}{\sum_{t=1}^{T} \{z_{t-1} = s_i\} + k}$$
(3)

A hidden Markov model is made of two sets of states one hidden and the other observed. We want to find the most likely series of hidden states given the observed states and known transitions between the observed states and the hidden states. This can be done naively by keeping track of all the provabilities; however, this is computationally expensive. A dynamic programing approach, known as the Viterbi algorithm, is used instead. This algorithm instead keeps track only of the maximum probabilities through a recursive definition.

III. Recurrent Neural Networks

Recurrent Neural Networks work similarly to standard artificial neural networks with the addition of feedback loops. The internal recursive states allow this kind of neural network to exhibit dynamic temporal behavior, making it appropriate for processing time series data.

A neuron is made up of two parts a weighted sum (weights, inputs, and a bias) and an activation function. In order to learn on non-linear data, the hidden layer neurons usually use an activation function that is in the sigmoid family (logistic, Gaussian, etc.). The output of each neuron is the weighted sum passed into the activation function. The output of each neuron is passed to each neuron in the next layer. When predicting a continuous value, a single linear output neuron is usually used. Learning is usually done on neural networks through the back propagation algorithm. This algorithm splits the error (squared error) between each weight it the network. In order to do this, we use the chain rule. On the hidden layers the sum of the derivative is part of the chain that is multiplied to find the error. An example of how error flows back to the hidden

$$\frac{\partial Error_{total}}{\partial w_{hidden}} = \left(\sum_{HigherLayer} \frac{\partial Error_{total}}{\partial output_{higher}} * \frac{\partial output_{higher}}{\partial net_{higher}} * \frac{\partial net_{higher}}{\partial output_{hidden}} \right) * \frac{\partial output_{hidden}}{\partial net_{hidden}} * \frac{\partial net_{hidden}}{\partial w_{hidden}}$$

$$(4)$$

layer is given in Eq. 4 (net is net sum of weights times inputs and bias).

In recurrent neural networks the output of the hidden layer neurons feeds back as another input. A basic layout of a recurrent neural network is given in figure 1. Since the input from the previous time is passed as input to the current time, recurrent neural networks have a form of memory. With a recursive definition, this memory covers every input set.

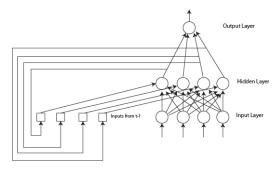


Figure 1: An example recurrent neural network. Outputs from the hidden layer neurons from the previous time step are fed back in as input to the next time step.

IV. Gradient Boosting Regression

Gradient Boosting is an ensemble learning algorithm that uses a weighted average of simple models to learn a more complex model. The algorithm first uses a simple model to fit the data, then a simple model to fit the residuals between the data and the first model. This process continues on each model's residuals until an appropriate fit is found.

As the model increases in complexity, overfitting tends to start occurring. To avoid overfitting, the Gradient Boosting algorithm keeps a portion of the data (different in each iteration) out of the training set and uses it for cross validation. The algorithm keeps track of the cross validation error and determines that an appropriate fit has been found once the cross validation error increases for a given number of iterations.

We used the R package XGBoost to train our Gradient Boosting models, then used parameter optimization to find the best solution.

V. Results and Discussion

Random Forest

For our Random Forest approach, we used the R package RandomForest for training models and making predictions. This package allows for the variation of three major parameters. Each parameter is mainly used to balance the tradeoff between runtime and fit quality. Because random forest regression is a very resource hungry algorithm, parameters must be set with feasibility in mind. The parameters given to the model are as follows:

- *mtry* the number of features allowed in each decision tree. For a given value *k*, trees may not use more than *k* regressor variables from the 18 features given to the model. Higher values for *mtry* allows each tree to model more information from the features, but requires more trees to generate an informative average.
- *ntrees* the number of trees averaged in the model. A higher value means runtime will be longer, but with a better fit.
- sample size because the training algorithm is so resource hungry and Rossmann provided so much data, it is not feasible to run Random Forest Regression

on the whole dataset. Therefore this parameter sets the number of random data samples that the model is trained on.

Table 2 shows the parameters for our best two Random Forest models as well as the prediction RMSPE as scored by Kaggle (Kaggle scores on data for which we only have access to the feature data).

Table 2: Parameter Selection for Random Forest Regression

mtry	ntrees	samplesize	RMSPE
7	100	100000	12.09%
9	300	100000	11.98%

II. Hidden Markov Model

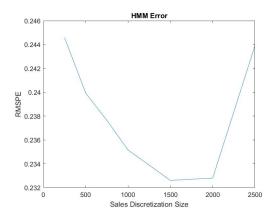


Figure 2: HMM error using Laplace Smoothing

For testing with HMMs, the stores were treated separately. The decision to use the stores separately came from An experiment normalizing the data to zero mean and combining the stores to estimate transitions. Combining all this produced a much higher error (over 70% RMSPE) and was computationally expensive.

The observed states, what we know in advance, is everything except sales and customer data. There are 224 state for different observations (7 days in a week, Store Open/Closed, Promotion Active, 4 state holidays, and a school holiday). The sales data needed to be

discretized to work with the hidden Markov model. A smaller discretization leads to the potential ability to be close to the true value; however, it also decreases the probability estimation between states as there are more states. Figure 2 show the results of different levels of sales discretization. This experiment showed that dividing sales by 1500 and rounding produced the lowest training error.

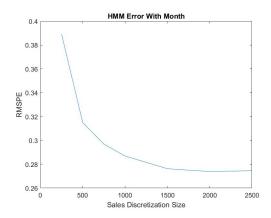


Figure 3: HMM error using Laplace Smoothing and month data

In order to account for seasonality effects, the month was also included as an experiment. However, when tested the training error was higher than not including this data (see figure 3). Increasing the number of observed states by a factor of 12 (2688 states) likely decreased the estimation of state transitioning.

III. Recurrent Neural Network

Due to the temporal nature of the data hold out cross validation was used. 20% of the data was left out for validation. In order to work with the Matlab's functionality, the data was normalized. Each store was treated separately for reasons similar to the Hidden Markov method. During testing it was found that the predicted values for closed days did not always go to zero, so we set values for closed days to zero. An experiment was to test the effect the different numbers of neurons had in the hidden layer. The experiment show that 7 neurons produced the lowest training error (see figure 4).

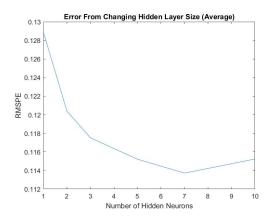


Figure 4: RNN Error averages of different number of hidden neurons (closed days forces to 0).

The actual error from Kaggle had much higher error of around 24%. This may be due to over fitting. The neural network may have learned the pattern leading up to the test date, but the Kaggle test data started at a different time.

IV. Gradient Boosting

Similar to Random Forests, Gradient Boosting is an ensemble based regression, efficient for black-box prediction. However, Gradient Boosting is much faster to train than Random Forests. It has been used in several Kaggle competition-winning solutions and has been developed into a the R package XGBoost.

Table 3: Model Performance

Parallel Trees	# Features	RMSPE
default(1)	17	10.68%
default(1)	17	10.70%
10	17	10.52%
10	17	10.50%
default(1)	18	10.58%
10	18	10.56%
	default(1) default(1) 10 10 default(1)	default(1) 17 default(1) 17 10 17 10 17 default(1) 18

As shown in Table 3, our first run of Gradient Boosting generated a RMSPE of 10.681%, a significant improvement on our best Random Forest model. To further improve our model, we increased the number of trees from 3000

to 5000, however the model wight 5000 trees performed worse than the previous model. We believe this to be due to overfitting since the cross-validation error does not improve from 3000 to 5000 trees while the training error goes down (as seen in Figure 5). Therefore, we chose to keep the number of trees set at 3000. We then added feature 18 to our feature set (Competition Age).

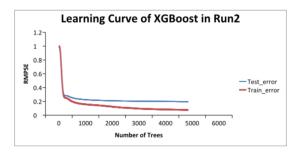


Figure 5: Learning Curve for 5000 tree model

Adding Competition Age to our feature set did not improve our model on its own, but combining models (runs 3, 5, and 6) lead to a model with an improved RMSPE of 10.439%, our best result.

Table 4: Model Performance

Model	RMSPE
Gradient Boosting	10.44%
Random Forest	11.98%
Hidden Markov Model	16.52%
Predict Median Sales	19.25%
Recurrent Neural Nets	24.88%

VI. Conclusion

We found that though Hidden Markov Models and Recurrent Neural Networks are better in theory for time-series data, ensemble methods performed much better in practice. We believe this to be due to their robustness in black box optimization considering the data provided is unstructured and some features may not correlate well with sales.

REFERENCES

- [1] Figueredo, A. J. and Wolf, P. S. A. (2009). Assortative pairing and life history strategy a cross-cultural study. *Human Nature*, 20:317–330.
- [2] A. Liaw and M. Wiener (2002). Classification and Regression by randomForest. *R News*, 2(3):18–22.
- [3] Friedman, Jerome H (2001). Greedy Function Approximation: A Gradient Boosting Machine. *The Annals of Statistics*, 29(5):1189-1232.

- [4] Friedman, Jerome H (2002). Stochastic Gradient Boosting. *Computational Statistics & Data Analysis*, 38(4):367-378.
- [5] Samarasinghe, Sandhya (2006). Neural Networks for Applied Sciences and Engineering: From Fundamentals to Complex Pattern Recognition. *Auerbach Publications*
- [6] Zucchini, Walter and MacDonald, Iain L. (2009) Hidden Markov Models for Time Series: An Introduction Using R (Chapman & Hall/CRC Monographs on Statistics & Applied Probability) Chapman and Hall