UIC
UNIVERSITY
OF ILLINOIS
AT CHICAGO

PROJECT FINAL REPORT

Forecast sales using data provided by the Rossman store-chain

Authors:
Marco Cavallo
Filippo Pellolio
Andrea Rottigni
Ronnypetson Souza da Silva
Benedetto Vitale
Professor:
Brian Ziebart
Course:
Introduction to Machine Learning

December 11, 2015

Contents

1 Introduction

In this report we are going to offer a detailed description of the assumptions made on the data and of the techniques adopted to approach the problem, together with an evaluation measure chosen to estimate how that specific techniques have performed w.r.t the task considered.
For more details on the task, please consult the project proposal and the project intermediate report, given that here we will take as granted the information already provided in those documents.

2 Preprocessing Reminder

We preprocessed according to what has already been presented in the first project report. Here we are going to consider the data obtained after having applied the preprocessing procedures already depicted , so we will focus more on the details, assumptions and results relative to the various approaches adopted.

3 Evaluation Reminder

In this section we present again, as a reminder, the formula relative to the Kaggle evaluation measure, which we have used both by submitting the data on the website and by implementing the formula ourselves and then applying it to our validation set chosen according to the specific approach considered.
Here is the formula:

$$\text{RMSPE} = \sqrt{\frac{1}{n}\sum_{i=1}^{n}\left(\frac{y_i - \hat{y}_i}{y_i}\right)^2},$$

Where y_i denotes the sales of a single store on a single day and $\hat{y}_i$ denotes the corresponding prediction. Any day relative to a store with 0 sales is ignored in scoring.
We have chosen to adopt this formula for the error computation, given that we are dealing with a regression task, thus finding a more suitable evaluation measure is quite difficult, considering that standard evaluation techniques cannot be used.
In addition, this seems a reasonable way of computing the error w.r.t. the golden value, since it is normalized by both the golden value and the number of instances considered in the test set.
We would have liked also to try a cross-validation in order to better estimate the error

on the train data, but given the huge dataset, it would have been too computationally heavy.

We know present the various approaches with detailed description on the experiments and the relative evaluation procedure:

4 Code

All the code and scripts we used for this project can be found at our Repository, all the code is fairly well commented and straight-forward, so it should not be hard to read and understand it.

5 Approach I: Regression Tree

One of the first approaches that we tried was based on decision trees: they are obtained by recursively partitioning the data space and fitting a simple prediction model within each partition - as a result, the partitioning can be represented graphically as a tree and analyzed based on the splits obtained. Due to this possibility of visually and explicitly representing decisions, we tought it could also turn out useful for some future decision making. Since we had to deal with infinite ordered values (e.g. sales), we relied on the so called "regression trees", with prediction error measured by the average difference between the observed and predicted values.
Starting from the already preprocessed data and leveraging the scikit-learn tool available for Python, we tried many combinations of features on which to apply the algorithm on, starting from all the applicable ones and then filtering out little by little the ones that seemed less significant from the point of view of the prediction. We surprisingly noticed that number of considered features didn't influence so much the prediction error, which anyway only reached 11% with the best configuration and many assumptions. As stated before, we performed our evaluation both on Kaggle's leaderboard and by splitting the dataset into a train (70%) and a test (30%) set.
We also tried some modifications to the preprocessed data in order to consider some eventualities - for example, we tried to filter out all the months that were not present in the test set, but we realized that it is not a matter of time of the year but of sales over time. So, instead of relying on the basic features "month" and "week", we derived a new incremental feature numbering the subsequent weeks. However, it didn't bring the huge change that we expected and, since in the meanwhile we had already obtained better results with other techniques, we preferred to concentrate on them.

6 Approach II: Linear Regression

Another of the first approaches with which we came out to forecast the amount of products sold for each day was linear regression. We believed that linear regression could have lead to good results for such a task, but we have probably underestimated the fact that most of the attributes were binary and this is probably the reason why all of ours attempts using linear regression always produced very poor results. We used the python scikit-learn package to create our classifier. We tried several different preprocessing hoping to improve the accuracy of our prediction. The first time we tried to fit the classifier with all the dataset. Since the results were very poor we tried several other different approaches:

- Make a different linear regression for each store: it seems reasonable to create different linear regression for each store because store in big cities are radically different from store in small villages.
- Make a linear regression using only the months that we have to predict: since the amount of products sold might be strongly dependent on the season, we decided to consider only the months corresponding to the ones we had to predict.
- Combine the two different approaches explained above.

We decided to make no further try with linear regression, because none of these attempts produced acceptable results: performances were below predicting the median of each day, so we decided to concentrate our energies over the classifiers that seemed to be the most promising: k-nearest neighbors and random forests.

7 Approach III: Naive Bayes with Quantiles Division

We have then decided to try a Naive Bayes approach, by noticing that stores can be assigned to a certain group according to the range of variations in their sales during the period.
First of all, a subset of the data in the train set is chosen; 80% of this subset is then used to training the learner and the rest for testing.
The whole subset is then preprocessed in the following way:
the DayOfWeek feature is transformed in 7 binary features, one for each day of the week. In some test cases, the Store feature is deleted. All the cells in the train set are filled, so we can use this part of the data without the need to fill empty cells. The current method using Naive Bayes Classifier takes the number of sales of all samples and then defines k classes based on k-1 k-quantiles. The values used for k were from 4 to 7 (because there are many samples with 0 sales, thus we were not able to have more than 7 classes without having 0.0 as a quantile).
After this, the Store feature is ignored and the classifier considers the first 4 or 3 features (depending on the presence of the Store feature) as continuous features and the last 10 as binary. Then the parameters of all features in all classes are learned empirically and also the class priors, generating the model for the classifier, that chooses the class with the higher posterior probability. The data chosen is a set with 66396 examples, that corresponds to one month of sales and 53117 examples were used for training and the rest for testing. The best result found was obtained using k = 4. After the classifications of the test and also the training examples, we got an accuracy of 63.82% in the training data test and an accuracy of 61.16% for the test data. The error on the test data was 15.63% and 6.81% for the test using the training data. The following graphics shows the distribution of the classified test data:

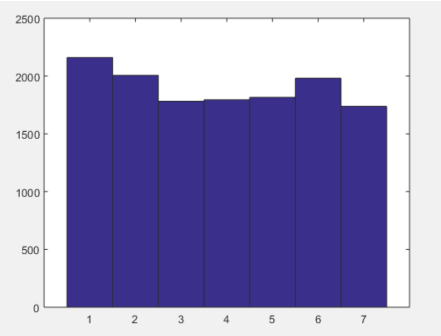


Figure 1: Distribution of the classified test data

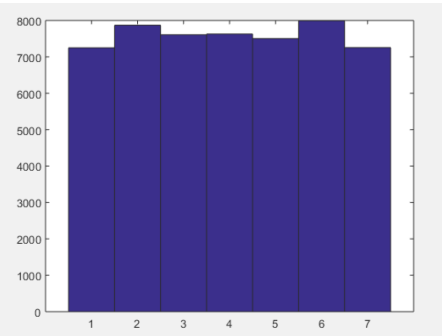


Figure 2: Distribution of the classified train data

8 Approach IV: K-Nearest Neighbours(KNN)

Since we noticed that the dataset provided is huge, but not huge enough to prevent us from trying the k-nearest neighbor, we decided to give it a try.
The first thing we noticed is that once we found the k most similar rows in the train set we could not simply compute the mean and use it as a prediction because the data may refer to very different stores. Computing a mean between the number of sales of a huge store in the center of Berlin and of one in a 1000 people town makes no sense. So we decided to preprocess our data and calculate a new computed feature, the deviation from the mean of that row: if a store has a mean selling of 1000 and on that day it sells 1100 we know that that day he sold the 10% over it's mean.
So the KNN here it's really predicting only a percentage of sells with respect to the mean of the store of that particular test data.

8.1 Simple K-Nearest Neighbor

As a first try we tried a trivial run of the k-nearest neighbor with k=5 and 10, the attributes in this case were not weighted and the store attributes were completely discarded.
The results were pretty bad and the computation took a long time, so we decided that there was a need for caching in the following trials.

8.2 Caching in KNN

Since the computational time for the algorithm to be run was pretty big, we decided to cache the results for every test data analyzed, since we were considering few attributes this approach sped up the computation greatly. Later on, when considering more attributes the boost decreased, but it was still useful.

8.3 Considering the Store

The first approach did not lead to great results, but it was pretty trivial, so we decided to continue on this way refining the similarity function.
The first refinement was to consider the stores attributes related to the test data. Two tuples are more similar if the store they are associated to is of the same type, if it's roughly of the same size or if it is exactly the same store.
This led to a little improvement, but it still was way less then expected. So we decided to drastically change our similarity function.

8.4 New Similarity

Since we were considering all the attributes we had, we couldn't add any other to the game, so we decided to tune the weight we gave to each attribute.
The first thing we noticed is that the day of the week was a big influencer in the number of sales: the fact that two data were both referring to a Saturday made them way more related than the same store on a Tuesday, so we gave the day of the week a big weight.
Among all the data referring to the same day of the week we now had to find the second most influencing attribute. It turned out to be the store type: apparently all the pharmacies sells more on some day of the week, while the toy stores sell less in that particular date. (We don't really know what types of stores our dataset is referring to, we only know them by means of an a,b,c,d classification. The previous was just an example).
Using these new weights led to way better results, so we decided to go on and explore more correlations in the data.

8.5 Ignoring Noise

After a while we realized that with our few attributes the similarity function was not the problem anymore, it gave us the most similar results, but it was painfully low. Even with caching, since we were considering all of the attributes, the computation time was really high, because every test row was different from the others.
Then we realized that, since our test data is all coming from a very limited date span (September-October) data from other seasons was never considered in the most similar data, but still it was processed by our algorithm.
So we decided to preprocess the data in a very trivial way: ignore all the data not coming from the date span of interest.
This not only led to way faster computation, but even to better results! At first we couldn't get our head around why the performance could have gotten better since we eliminated data that was never selected in the KNN computation, then we realized that this data was still considered in the store mean sales computation. We were still considering Christmas in the sales mean, while we were only interested in a couple of summer-end months.
As incredible as it seems, ignoring most of the train set turned out to be the best thing we could do.
Sadly the result we obtained in this way were still not good enough and, since we couldn't think of anything else to do to improve the KNN, we decided to move on to a new approach.

9 Final Approach: Random Forest

Since we have noticed that such a regression task could be suitable to ensemble methods, given that by trying various features configuration and by merging the predictions we could take more information into account, we have decided to use *Random Forest Regressor* of the *Python* library *scikit-learn* to solve the relative task.
We have tried many approaches with random forests, trying to improve the results according to what was clearly biasing our classifier towards mistakes in that specific approach.
Now we are going to present each one of them, together with the observations that has made us change our mind towards different approaches:

9.1 Standard Random Forest

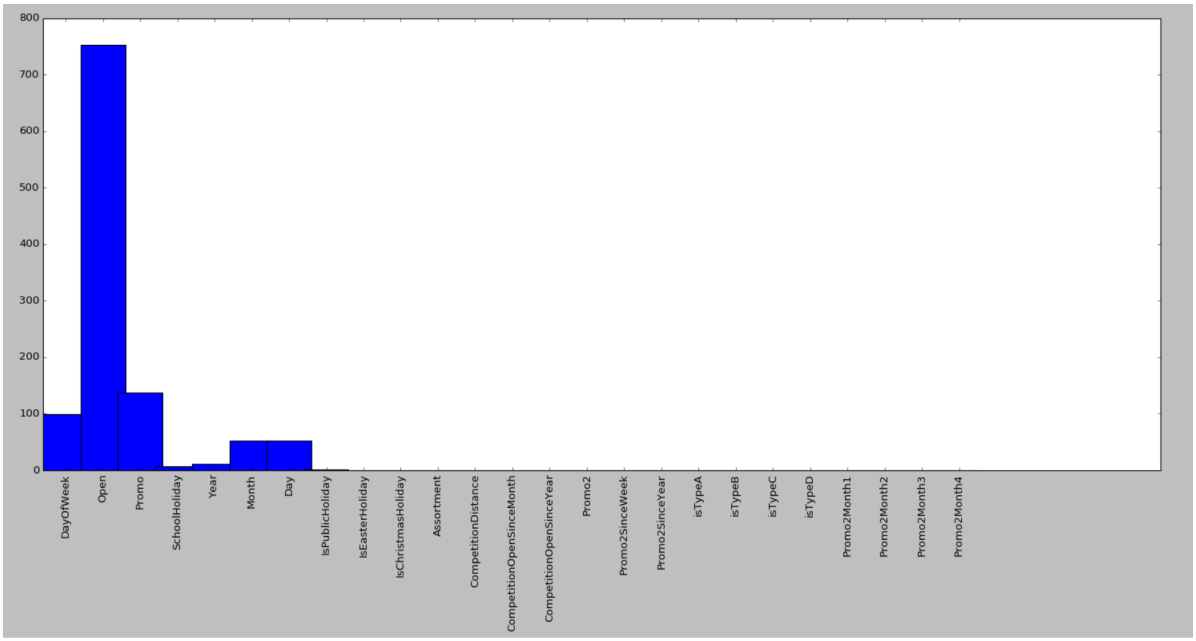
To begin with, we have tried a standard Random Forest, trained on the whole training set and then tested on the provided test set. However, given the huge amount of data, we were not able to perform this test, that's why we have thought of finding a way to consider a smaller amount of data per time.

9.2 Sampling Random Forest

In order to consider a smaller amount of data, we have sampled for a certain number of times, without replacement, our training data for a reasonable amount of data(300000 per fold), then each fold obtained by sampling is used to train a different random forest and generate a different prediction on the testing set. In the end, we choose as final prediction for the i-th row of the test the one which is less distant from all the others predictors, always relative to the same row of the test. So, by doing this for every row we obtained our final predictions.
For the error computation, we have first split the training set in two, using 70% of the data to train and then 30% to test(in order to have a period to predict with the relative golden labels and also because a Kaggle competition has two phases and during the first one, which is our case, we are tested just on the 30% of the data). Then, once obtained the final prediction from the method depicted above, we have computed the error according to the formula shown above, yielding 0.4 as final error value.
As we can see the error is quite high.
We have hypothesized that this is due to the fact that the various rows sampled from the training set have too variable 'Sales' values in order to be properly used as a whole training set; we clearly need to group the stores according to some separation criteria in order to give to the model more reasonable input data.

9.3 Random Forest By Store

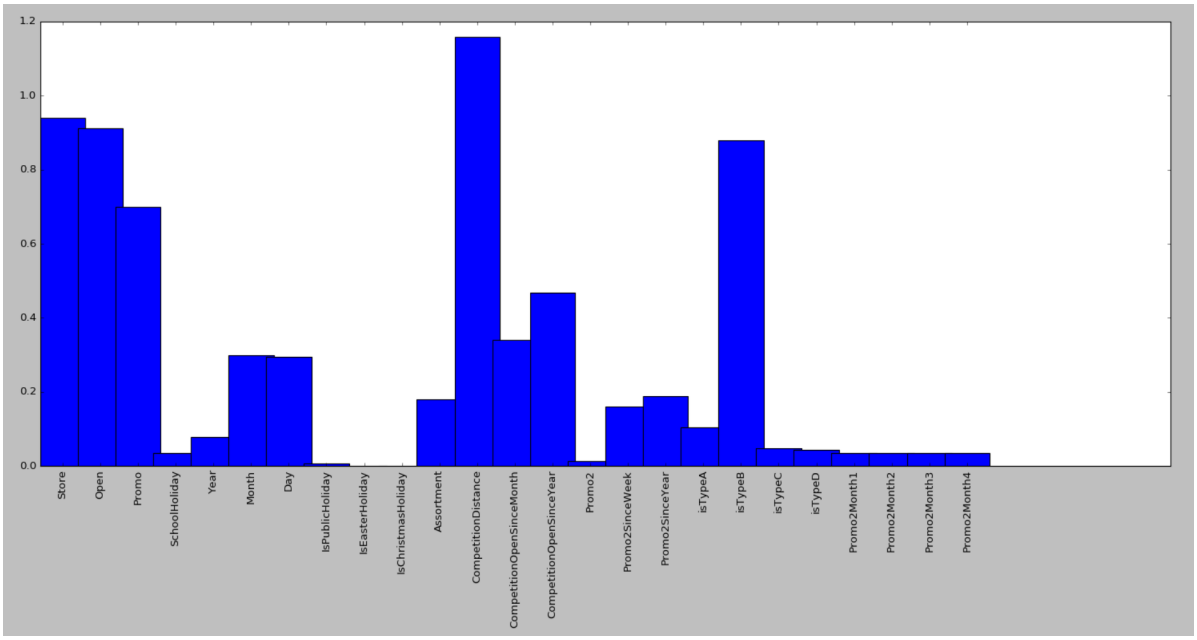
The first really immediate separation that came into mind was the one of grouping the train and test set by store, then training a random forest per store and using it to predict on the rows of the test set relative only to that specific stores. This approach brought to huge improvement in the error estimate, which was 0.134.
This time the test set used were different for each store:
Once obtained the training rows relative to a specific store, we used again 70% of the data to train and then 30% to test and then computed the final error as in the case above; this was done for every store in the train data.
For this approach we estimated also the importance of every single attribute in the splits performed by the Random Forest, obtaining the following barchart graph:



As we can see, when dividing by store many attributes were useless, since they are the same for every row relative to that store, that's why we needed to find some separation criteria in which we could exploit in a more significative way those attributes.

9.4 Random Forest By Day of the Week

Another almost immediate separation criteria that came into mind was the one of grouping stores according to the day of the week and then proceeding exactly as the case above but for each day of the week instead, yielding in the end just slightly worse results than the previous case(the Kaggle error computed as above was 0.141).
We tried to apply a further separation in this case, by dividing the rows relative to each day of the week according to if 'promo1' was active or not for the stores relative to that day of the week. This separation was applied to both the test and the train data, obtained as in the previous approaches. Then, a different random forest was trained for the 'promo1' data and not 'promo1' ones and then the relative classifier is used to predict, obtaining an error estimate of 0.062, which is a huge improvement w.r.t. the case in which 'promo1' was not considered as explicit separation factor.
Also for this approach we estimated also the importance of every single attribute in the splits performed by the Random Forest, obtaining the following barchart graph:



As the barchart shows, various attributes still gave almost zero contribution with respect to the splits performed by the algorithm, but still we have a much better overall use of them w.r.t. the previous case.
We tried as well not to consider the attributes bringing low information when fitting the classifier, but this has given almost the same results, given that the Random Forest algorithm is good enough in not considering useless features.

9.5 Clustering + Random Forest

Thinking of other possible ways to properly separate the stores, we thought also to the following approach which uses clustering:

- Compute for each store the mean and the variance of both sales and customers
- Apply a clustering algorithm to the stores in order to group them properly
- Given the stores in each cluster, take from the train only the rows relative only to them
- train a Random Forest on the train data retrieved before
- test the data relative to those stores using the previously trained Random Forest
- repeat the procedure until all clusters have been covered

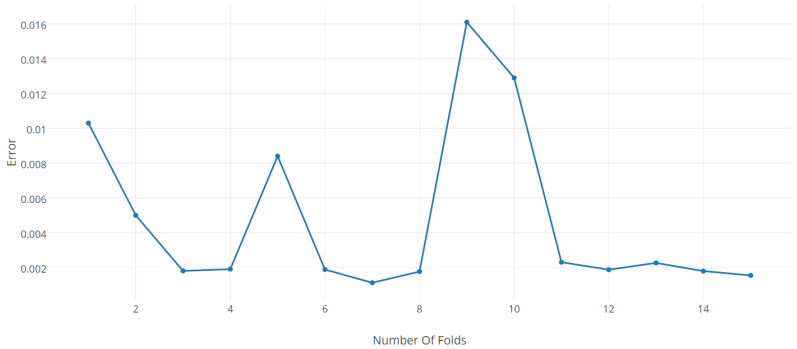
As clustering algorithm we tried both *k-means* and *Affinity Propagation.* Given the general difficulties in finding the best K after many trials, we preferred the second one, given its flexible nature. The Affinity Propagation Algorithm identifies a set of exemplars data points among the stores and then assign the other stores to the most similar exemplar according to the statistics computed before.
In our case, the estimated number of clusters is 533 with a total number of stores of 1115. The was then computed as the above cases, where once obtained the rows in the training set relative to the cluster, we have used the 70% to train the model and the 30% to test it, computing in the end the final error over all the test sets considered for the various stores in the clusters.
However this approach gave results slightly worse than the previous case:
we obtained an error of 0.0904 without further dividing train and test according to if 'promo1' was active that day or not, while, surprisingly, an again slightly worse results with an error of 0.0914 when dividing also according to the 'promo1' feature. This did not promise a future improvement w.r.t. our previous approach, so we combined two different approaches already used in order to improve the results.

9.6 Forest Per Day with sampling

As a final approach, we tried to combine the sampling approach with the stores per day separation(which contains more rows per file w.r.t. the division per store).
Basically, this is the proposed approach:

- Divide stores per day of the week
- For each day of the week data do the following:
  - use 70% of the data to train and 30% to test
  - apply the 'promo1' separation to the test
  - Sample the train portion a certain number of times for a fixed sample size and for each sample do the following:
    * apply the 'promo1' separation
    * train a different Random Forest on the 'promo' and 'noPromo' data
    * predict with the relative forest the 'promo' and 'noPromo' test data
- Compute the final error

This approach resulted in a considerable improvement w.r.t. the error obtained with the usual procedure without sampling. We think that this is due to the fact that in order to decrease the randomness factor of such classifier, training it many times and then trying to get the most stable predictions(less distant from all the others for each row in the test considered) is going to give an overall better performance, once the initial stores have been separated in order to decrease the initially excessive data variability.
Here we present a plot of the error trend w.r.t. the number of folds used for each day of the week, where the size of the each fold is kept fixed at 100000 rows:



As we can see, we got the best error when we use 7 folds, while for all the other numbers of folds chosen we get always a considerably greater error, exception made for 15 folds, where we got a slightly worse error than 5 folds.

10 Lesson Learned

Prediction isn't easy at all.
From this experience we have learned some important lessons:

- One of the first lessons we have learned from this project is that preprocessing plays an important role because data is very noisy and a good preprocessing might make the difference.
- Another important lessons is that computations on large amount of data are extremely slow and sometimes we had to stop the running process because after several our of computation was still computing. So optimization of the algorithms used, especially in the k-nn case, is another key aspect.
- Never give up, even if sometimes can be frustrating spending several hours in modifying slightly an algorithm and spending even more hours waiting for the end of a computation that might lead to an accuracy that do not improve your previously results.
- Not all classification methods perform equally well and the appropriate classifier must be chosen according to the data available. That?s probably the reason why an extremely power classifier such as linear regression had so poor accuracy.
- Sometimes things work and you don't really know why. Maybe you spend days in elaborating complex theories and they work really bad and instead the simplest approach sometimes works better. So you can experience the Occam Razor in real life.

11 Conclusion

During the various test performed, we noticed that in order to obtain suitable prediction for the target value, it does not suffices to choose the right algorithm alone. The right target value is obtained through proper combination of various aspects, which we have identified to be the following:

- Must be performed the right preprocessing in order to maximize the information hidden in the data. In fact, also in our case there are many features that could have potentially brought us information, but only after that they have been properly processed and expressed in a more easily understandable form for the various algorithms used
- Choosing the right algorithm is crucial as well(even if not alone), considering that, according to the specific problem, the way in which data are used could provide better result w.r.t. other ways. In our case the Random Forest have provided the best results, given the huge amount of data available from which to learn, but, for example, Random Forest are not that good when you have just few data.
- Much domain knowledge is required as well in order to extract the highest amount of information from the data available and, eventually, to engineer new features relative to the domain, but not yet incorporated into the available data, as it could be the case of the geographic location for the stores(unfortunately not available here, since no hint on the location of the stores has been given), that could have potentially brought much more information.

In conclusion, we cannot generalize an approach to a various set of problems; each problem requires an 'ad hoc' approach, where one needs to identify each aspect according to the context considered and then making decision according to that.
We can summarize the message by remembering that:

Essentially, all models are wrong, but some are useful *George P. Box*