Machine Learning Engineer Nanodegree

Capstone Project

Jose Rodriguez
June 21st, 2017

I. Definition

Project Overview

Before my enrollment on the Machine Learning Nanodegree Program I had an Idea of what Supervised Learning was, but now after seen all the scope of what this course offers me, I have a very clear understanding of what to do to solve multiple problems and situations.

Supervised machine learning is the search for algorithms that reason from externally supplied instances to produce general hypotheses, which then make predictions about future instances. In other words, the goal of supervised learning is to build a concise model of the distribution of class labels in terms of predictor features. The resulting classifier is then used to assign class labels to the testing instances where the values of the predictor features are known, but the value of the class label is unknown.

In this Capstone Project, I had the option to select a topic in Kaggle, so found a competition that I think suits my needs for this project, is the "Allstate Claims Severity Challenge": Allstate is currently developing automated methods of predicting the cost, and hence severity, of claims. In this challenge, Allstate enforce you to show off your creativity by creating an algorithm which accurately predicts claims severity.

Problem Statement

According to Allstate as stated in the Kaggle challenge: "When you've been devastated by a serious car accident, your focus is on the things that matter the most: family, friends, and other loved ones. Pushing paper with your insurance agent is the last place you want your time or mental energy spent. This is why Allstate, a personal insurer in the

United States, is continually seeking fresh ideas to improve their claims service for the over 16 million households they protect."

Using the data supplied by Allstate in the Kaggle challenge, that is a recompilation of real data from claims reported to Allstate (this data was transformed to protect the private and personal information), we can predict the claim cost by using Supervised Learning algorithms, specifically Regression algorithms to predict the continue value of a Claim Loss.

To predict the Claim Loss, we are going to use the data provided in (https://www.kaggle.com/c/allstate-claims-severity/data). The Dataset has a total of 587633 claims with a total of 116 categorical features, 14 continues features and one target loss value. Based on this data we are going to predict the Loss of a Claim using 4 Regression algorithms.

(https://www.kaggle.com/c/allstate-claims-severity)

Metrics

I will be using the Mean Squared Error. In statistics, the mean squared error (MSE) or mean squared deviation (MSD) of an estimator (of a procedure for estimating an unobserved quantity) measures the average of the squares of the errors or deviations—that is, the difference between the estimator and what is estimated. We are trying to minimize the MSE. I preferred MSE because it is relatively easy to understand and it punish hardly the worst results, it makes more clear when we are making a good or bad job (in my opinion).

$$MSE = SSE / (n - m)$$

where SSE is sum of squared error, n is sample size and m is the number of parameters.

(https://en.wikipedia.org/wiki/Mean_squared_error)

II. Analysis

Data Exploration

The data is provided in (https://www.kaggle.com/c/allstate-claims-severity/data).

Each row in this dataset represents an insurance claim. You must predict the value for the 'loss' column. Variables prefaced with 'cat' are categorical, while those prefaced with 'cont' are continuous.

The training Dataset has a total of 587633 claims with a total of 116 categorical features and 14 continues features, one “loss” column (target)that refers to the Loss of the Claim.

To download the data, you need to have an account in Kaggle and accept the rules of the challenge, the dataset size is 22mb aprox.

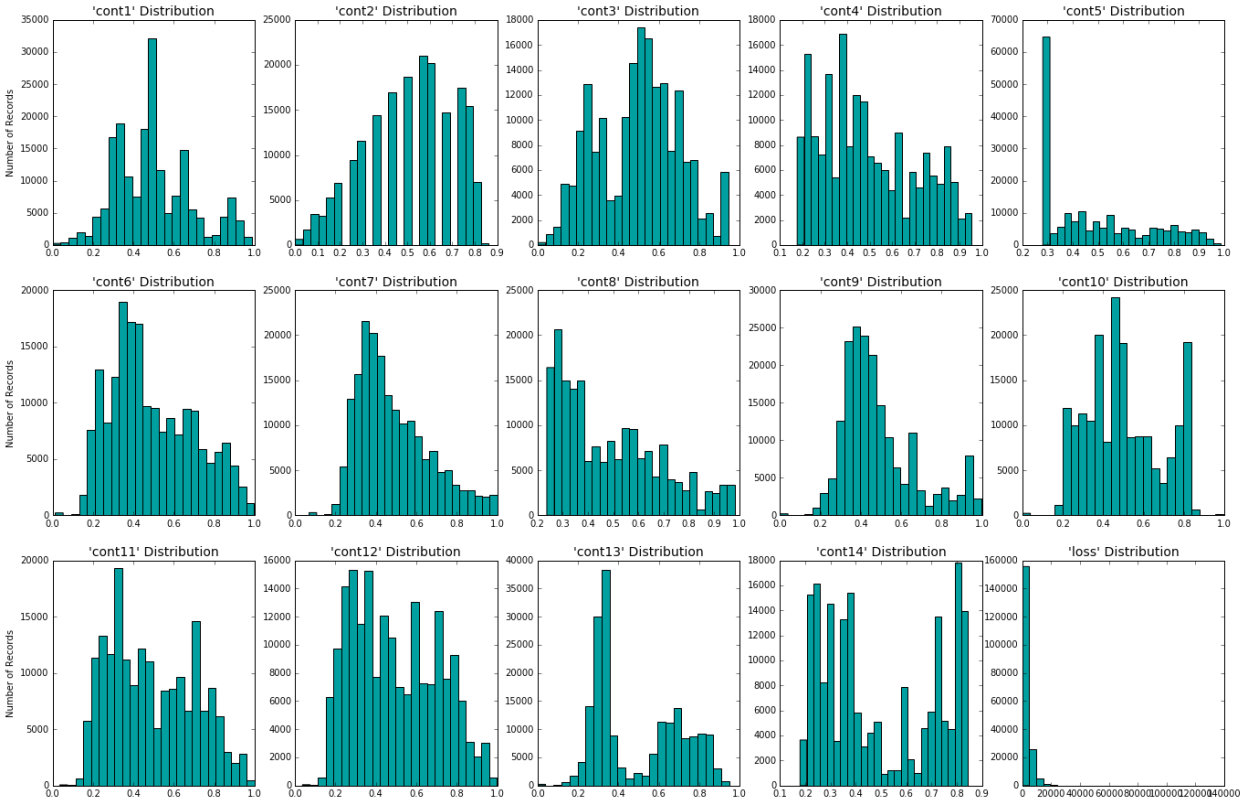
Data Statistics:
The features statistics are the following:

/	cont1	cont2	cont3	cont4	cont5	cont6	cont7	cont8
count	188318	188318	188318	188318	188318	188318	188318	188318
mean	0.493861	0.507188	0.498918	0.491812	0.487428	0.490945	0.48497	0.486437
std	0.18764	0.207202	0.202105	0.211292	0.209027	0.205273	0.17845	0.19937
min	0.000016	0.001149	0.002634	0.176921	0.281143	0.012683	0.069503	0.23688
25%	0.34609	0.358319	0.336963	0.327354	0.281143	0.336105	0.350175	0.3128
50%	0.475784	0.555782	0.527991	0.452887	0.422268	0.440945	0.438285	0.44106
75%	0.623912	0.681761	0.634224	0.652072	0.643315	0.655021	0.591045	0.62358
max	0.984975	0.862654	0.944251	0.954297	0.983674	0.997162	1	0.9802

/	cont9	cont10	cont11	cont12	cont13	cont14	loss
count	188318	188318	188318	188318	188318	188318	188318
mean	0.485506	0.498066	0.493511	0.49315	0.493138	0.495717	3037.338
std	0.18166	0.185877	0.209737	0.209427	0.212777	0.222488	2904.086
min	0.00008	0	0.035321	0.036232	0.000228	0.179722	0.67
25%	0.35897	0.36458	0.310961	0.311661	0.315758	0.29461	1204.46
50%	0.44145	0.46119	0.457203	0.462286	0.363547	0.407403	2115.57
75%	0.56682	0.61459	0.678924	0.675759	0.689974	0.724623	3864.045
max	0.9954	0.99498	0.998742	0.998484	0.988494	0.844848	121012.3

From the above tables we can see that all continuous features values are normalized, except for the ‘loss’ target feature. Also, we can notice there is no missing (or null) values as all the columns count are 188318. All the continuous values are non-negative.

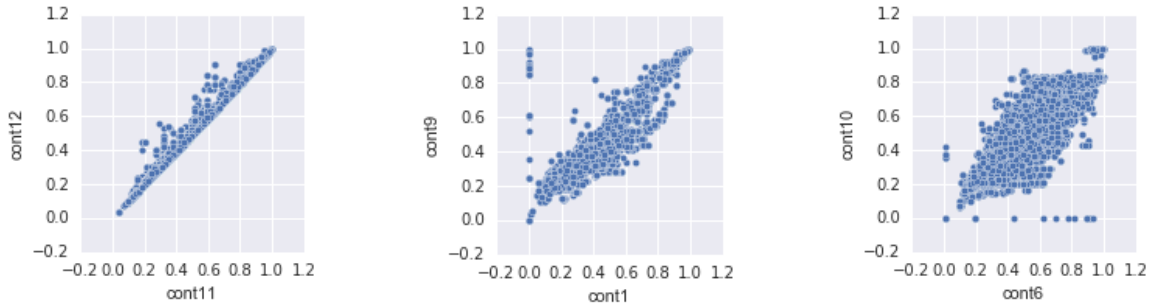
Skewed Features:
To analyze the distribution of the feature data and the skew of them I made the following plot:



In the plot we can observe that there are some features that have some kind of outliers or pikes in their values, but I thought there were no reason to modify or transform the data from the continuous features. The target value by the other hand was highly skewed and had pretty obvious outliers.

Data Correlation:

We could figure out that the dimensionality of the data is pretty high as we have 116 categorical features and 14 continues features. We need to look to the data to see if there are any pair of feature whose behavior was correlated to the other, so using a Panda integrated function I could know which pair of feature where correlated and I plot the three more correlated:



As we can see, these three pairs of features have a proportional behavior, so we could say they are correlated. Being two features correlated means that one of them doesn't give extra information from the other's, so one of them could be eliminated.

Algorithms and Techniques

The selected algorithms would be:

1) Support Vector Machine (linear)

In machine learning, support vector machines are supervised learning models with associated learning algorithms that analyze data used for classification and regression analysis. We will be using Sklearn's LinearSVR implementation.

https://en.wikipedia.org/wiki/Support_vector_machine

2) K-Nearest Neighbors

In machine learning, the k-nearest neighbors' algorithm (k-NN) is a non-parametric method used for classification and regression. In both cases, the input consists of the k closest training examples in the feature space. We will be using Sklearn's KNeighborsRegressor implementation.

https://en.wikipedia.org/wiki/K-nearest_neighbors_algorithm

3) Boosting

Boosting is a machine learning ensemble meta-algorithm for primarily reducing bias, and also variance in supervised learning, and a family of machine learning algorithms which convert weak learners to strong ones. We will be using Sklearn's AdaBoostRegressor implementation of a Boosting meta-algorithm powered by its default base estimator: DesicionTreeRegressor.

https://en.wikipedia.org/wiki/Boosting_(machine_learning)

https://en.wikipedia.org/wiki/Decision_tree

4) Stochastic Gradient Descent

• Stochastic gradient descent (often shortened in SGD), also known as incremental gradient descent, is a stochastic approximation of the gradient descent optimization method for minimizing an objective function that is written as a sum of differentiable

functions. In other words, SGD tries to find minima or maxima by iteration. We will be using Sklearn's SGDRegressor implementation.

https://en.wikipedia.org/wiki/Stochastic_gradient_descent

Benchmark

As benchmark, I will be using a Support Vector Machine model to compare the MSE to the MSE obtained for the Capstone Project. We will be training/testing this model with the same process of the other classifiers following the Project Design described below.

III. Methodology

Data Preprocessing

There were some issues with the provided data, so I had to preprocess it to be able to use it and get the best results as possible. The main changes I had to deal with were:

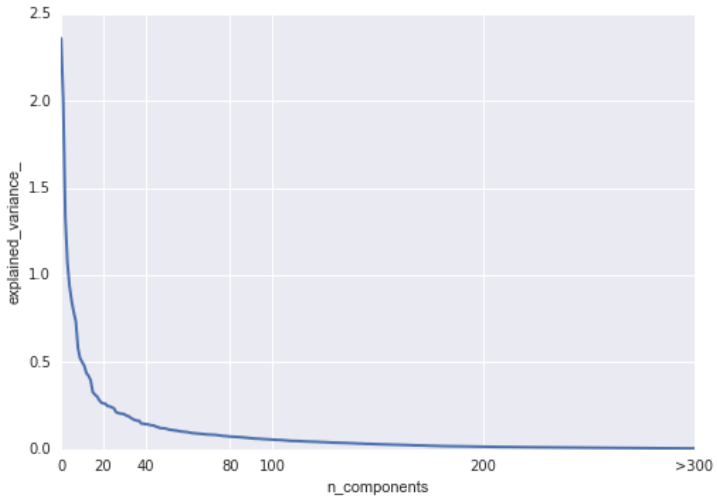
Normalize skewed target:
The target value was highly skewed and had pretty obvious outliers, so that made me use the Logarithmic function to normalize the distribution of the data of the target. We could get the original value by applying the exponential function. After we made the data transformation we get the following plot, where we can see how the data distribution is normalized:

Encode Categorical features:
The categorical features in the Sklearn library need to be encoded and flattened to be able to use them. So I used the LabelEncoder and the OneHotEncoder libraries to transform from labels to numbers the values and to flatten each categorical feature to

multiple binary features, respectively. So the outcome of this process was an array of features with the following shape: (188318L, 1153L). That means we have 188318 rows of claim information with 1153 features each one. That was a huge increment based on the 130 features that we had at first. Here the implementation was a little bit hard as the LabelEncoder has to encode column by column. I expected that this Class would accept two-dimensional arrays to process, but it was not the case.

Dimensionality Reduction:

At this moment we had 1153 features to learn from, so to be able to train the models we need to reduce the dimensionality of the dataset. For this I used an algorithm named PCA who can express the main variance of the features in some vectors named Principal Component where it projects the different values of some data. In the following plot we can see the explained variance based on the number of n_components.



Looking at the above image we could see that most of the variance is in the first 100 n_components. I used PCA to reduce the dimensionality to 100 features based on the first 100 Principal Components. The outcome of this stage is an array of data with shape (188318L, 100L). A lot better to process compared to the 1153 features we get from the previous stage.

Training and Testing Data Split:

To train and validate the model we need a dataset to train on and a dataset to test and validate against to. I used train_test_split module to achieve this by splitting the 30% of the dataset to be used as validation dataset. The outcome were 2 pairs of features and targets with the shape: Train: (131822L, 100L) and (131822L,), and Test: (56496L, 100L) and (56496L,).

Some bumps I had to overcome in the development of these implementation was the training time. Most of the algorithms I choose to try were slow when fitting or predicting the outcome. To overcome this problem, I use for the first training and best algorithm selection just the 20% of the training data. That way in the first training phase I would have to use less time to naïvely train the 4 algorithms. One of the worst time performance algorithm was SVR with a fitting and prediction time of 393 seconds and 109 seconds, respectively, just for the 20% of the training data.

Implementation

The algorithms selected to be tested against the data were: Support Vector Machine (SVR, kernel=linear), K-Nearest Neighbors (KNeighborsRegressor), Boosting (AdaBoostRegressor) and Stochastic Gradient Descent (SGDRegressor). To make the first approach to them I decided to use their default values and execute them against the data (training and testing) to compare them using their results.

To avoid spending too much time with the training and testing of these in this first phase I just used the 20% of the training and test data. This was a decision I made since the SVR

To compare them I used the mean_squared_error function. As the execution time is also an attribute to take care of I used it to compare them also. The results were:

/	Boosting	SGD	KNN	SVR
MSE	6805910.78	5660036.36	5974360.5	6018719.39
Fitting time	23.8939998	0.08599997	0.25099993	392.929
Prediction time	0.42400002	0.01499987	418.589	108.627

We can see from the above table that the best result for MSE was for the SGD algorithm with MSE: 5660036.36 and has the best fitting and prediction times (0.08599997 and 0.01499987 respectively). Compared to SVR who had a MSE: 6018719.39, SGD had a better result, besides the speed of SGD was a lot better than the SVR velocity.

Refinement

I used GridSearchCV to tune the SGD model optimizing the following parameters:

- n_iter: Using the following values [5, 10, 20, 40, 60, 100, 120] to try out different values for the number of iterations to take in the training.
- Average: this parameter is set to True to average the weights of every sample processed.

With this options for the parameters I've got the following best estimator:

```
SGDRegressor(alpha=0.0001, average=False, epsilon=0.1, eta0=0.01,
       fit_intercept=True, l1_ratio=0.15, learning_rate='invscaling',
       loss='squared_loss', n_iter=5, penalty='l2', power_t=0.25,
       random_state=None, shuffle=True, verbose=0, warm_start=True)
fitting time: 115.70600009
prediction time: 0.0119998455048
```

From the previous execution of the model we can see that we improved the result we get on the naive execution. We pass from a MSE of 5777346.0931 to 5591999.00906 using the normalized values of loss.

At this point I decided that the model was fast enough to make a test, so now we are going to see what happen if we don't transform the target values and don't reduce the dimensionality. Using the same GridSearchCV the best estimator was:

```
SGDRegressor(alpha=0.0001, average=True, epsilon=0.1, eta0=0.01,
       fit_intercept=True, l1_ratio=0.15, learning_rate='invscaling',
       loss='squared_loss', n_iter=60, penalty='l2', power_t=0.25,
       random_state=None, shuffle=True, verbose=0, warm_start=True)
fitting time: 716.026000023
prediction time: 0.480000019073
```

And the result was 4082608.64216, so it seems that my hypothesis was wrong about transforming the target data. It had a better result when we use the raw target and without reduced dimensionality.

We achieved an MSE of 4082608.64216 using the SGDRegressor with the original values for target and without dimensionality reduction. This result is better than the one produced by the SVR model without optimization (5971154.43957) by a few hundreds. So my hypothesis of data transformation of the target value and the reduced dimensionality from PCA wasn't good for this case. The PCA in this case wasn't good enough because the algorithm SGD has a very good time performance and it is able to manage that amount of features, but It was too much faster with the data with reduced dimensionality.

Model Evaluation and Validation

The model we finally get as outcome from the training and selection process was: SGDRegressor using the original target value, encoded categorical features and the original continuous features as input, using n_iter = 60 and average = true as best parameters for the model to fit the data.

This final model meets the expectation of the project as our target goal was to get better than the SVR algorithm, that it did. By the other hand, although I improved the result of the algorithm as best as I could, I thought that this model could be much better and the results obtained are not good enough to be a real solution to the problem.

The model is somehow robust, when I made small perturbations to the data using cross_val_score the model didn't get greatly affected by it. So if we get a better result from this model we could be confident of the model behavior won't be greatly affected by noise nor outliers. The results I get from running cross_val_score over the training data (131822L, 1153L) and target (131822L,) using 3-fold cross validation:

```
[ 4716570.46130768  4182057.34187917  4033495.87649668]
```

Justification

The final results we had in the SGDRegressor using the original target and without reducing dimensionality were much better than the results obtained by our benchmark model (SVR). Next we can see a table to compare both results:

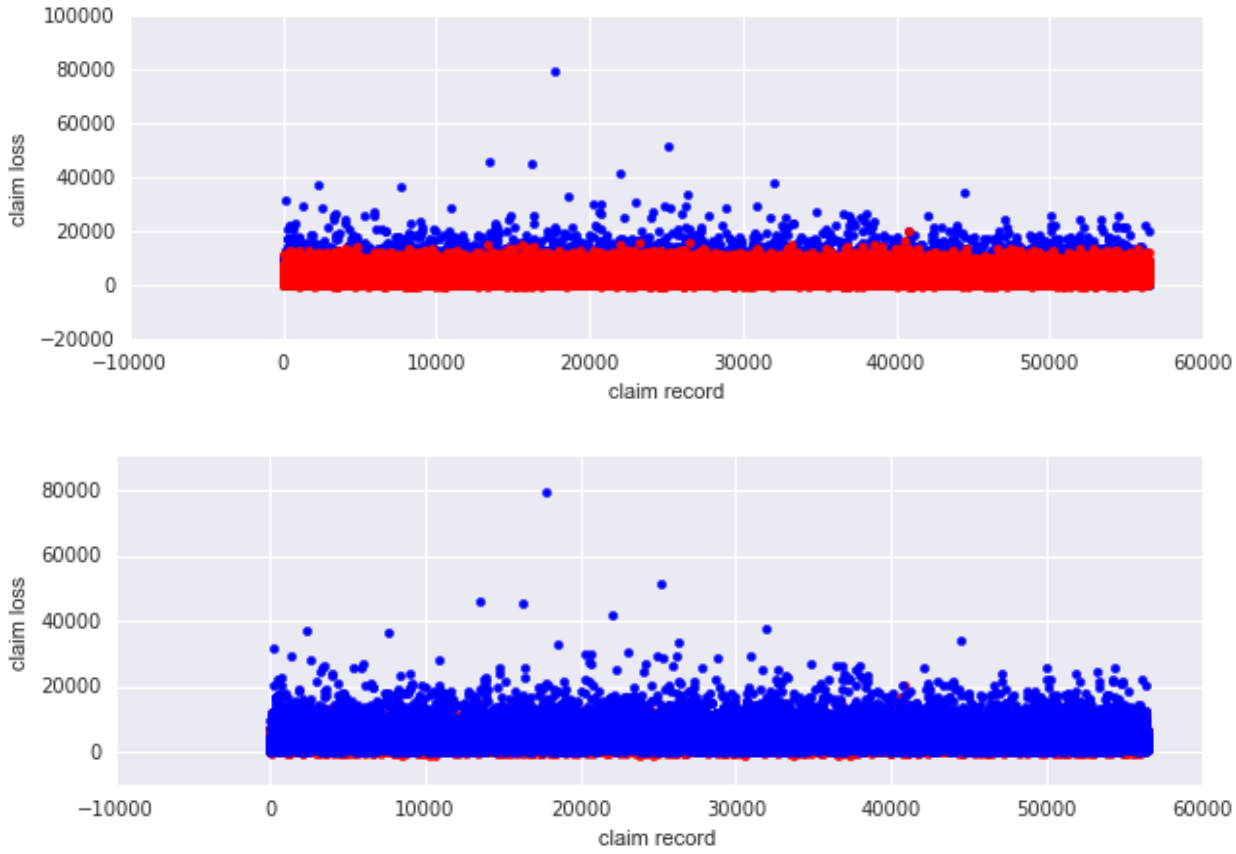
/	SVR	SGD
MSE	5971154.44	4082670.91
fitting time	465.459	729.197
prediction time	134.964	0.56000018

Based on this we could said that we met the expectations of this project, but I don't get a result good enough for a company as Allstate to use as a solution to their problem.

V. Conclusion

Free-Form Visualization

I plot the y predicted by our model and the y original to see both behaviors one against the other. The result was the following:



From the previous images we can see that the behavior of the predicted values (red) was pretty close to the original one (blue) at least it had a similar shape. From some point of view, we could see that the red dots could be a little low related to the blue ones. So the model seems to get the shape but it has some error against the other.

Reflection

Now we can summarize all the process we made and highlight some of the special cases and characteristics of this project. The steps we take during this were:

1) Process the data: identified the features and target columns, we needed to assure that we have numeric values in all columns, in our case we had categorical features containing non-numeric values. This was a problem; as most machine learning algorithms expect numeric data to perform computations with. We also needed to encode the categorical features to handle the labels as Boolean features (0,1). We saw that the target feature distribution was skewed, we normalized it by applying the logarithmic function.

2) Reduce the dimensionality: As we identified that there were correlated features and had a high number of features and it would grow a lot when we encode the categorical features, we needed to reduce the dimensionality of the features. We used Sklearn's PCA to achieve this.

3) Training and Testing Data Split: We divided the data into training and testing data, this would help us to measure the precision of our model after the training.

4) Model Evaluation: Here we used the selected models to perform the training and after this we compared the results of each of them based on their performance metrics (MSE and timing), after the training/testing and based on their error we selected the best model.

5) Model Tuning: In this step, I tuned the selected model and obtained improved results, using Sklearn's GridSearch to tune the parameters. After that I tried running the model without transforming the target value and without reducing dimensionality. This execution was pretty good as we had better results without those transformations.

6) I compared the results of the tuning model and its scores to the Support Vector Machine results from the pre-tuning face. I had better results using the tuning model than the results compared to the SVR.

The final solution didn't get my expectations as it is not a good solution for a production environment. The predictions are not good enough to believe in the response it returns.

Improvement

The things I think could be improved are:

- Use GridSearchCV with Pipeline to review all the possible algorithms with tuning of the parameters and trying different options to reduce the dimensionality.

- I think that maybe using an ensemble model to average the responses from the different models could get better results.
- I think there has to be a better solution, so it could use my current solution as benchmark.