Demonstrating the power of feature engineering – Part I

Feature engineering is often under-estimated in a machine-learning project. Many Data Science practitioners will go through this step too rapidly claiming they "know" their data or that the dataset is "already optimal" with all the relevant business features. Instead, they will go straight to the dessert and invest a big amount of time and computing resources doing model selection and hyper-parameters tuning process which in reality, is largely limited by the quality of the input dataset.

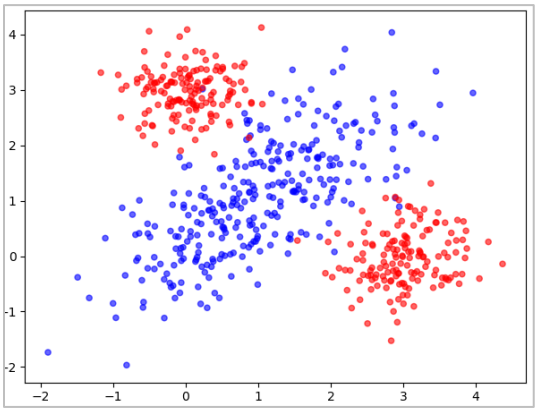
To find powerful feature engineering transformations you have to understand, of course, the business logics hiding in the data (Business transformations), but also how a desired machine-learning model mechanics works to find out how you can help that model mechanics to perform better with the feature-engineering step (Technical transformations).

Take this simple dataset example where you have only two features: X1 and X2. The target value (Y) represent the class you want to predict (Classification Problem):

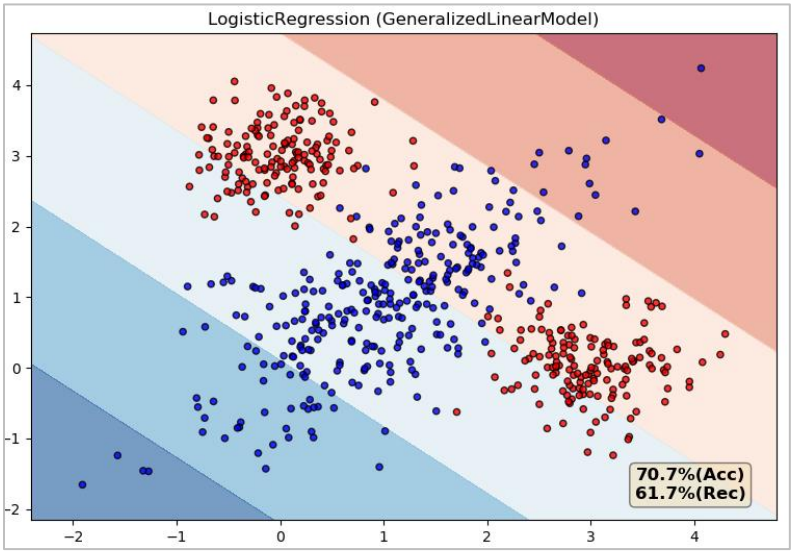
X1	X2	Y
1,1	2,1	Blue
0,5	-0,5	Red
3,4	0,5	Red
3,3	3,9	Blue
...	...	...

- **Red**: Business Example - Clients that churn before 3 years of business relationship
- **Blue**: Business Example - Clients that did not churn after 3 years of business relationship

If we scatterplot this dataset, we obtain:

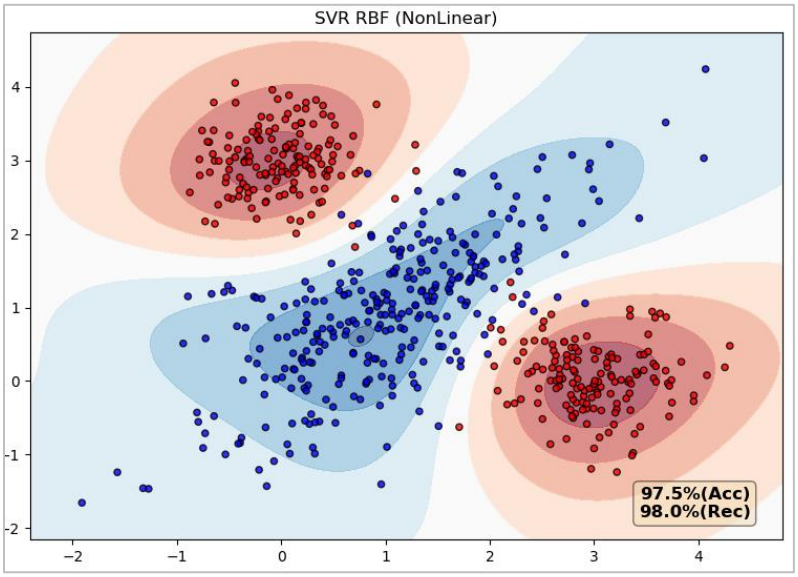


After a first look, you can say that you “know your data” and that this dataset appears non-linear. Soo if we try a GLM classification model like the popular linear Logistic Regression, it will not be optimal. Let’s try it anyway:



Well, this is bad. As we expected, the linear logistic regression model was unable to find a good generalization solution to split correctly the red class from the blue class. Instead, the model made a very simple linear split across the two distinct classes.

An instinctive basic solution to this problem would be: Let’s use a non-linear model instead! For example, let’s try a Support Vector Machine model with a Gaussian RBF kernel function:



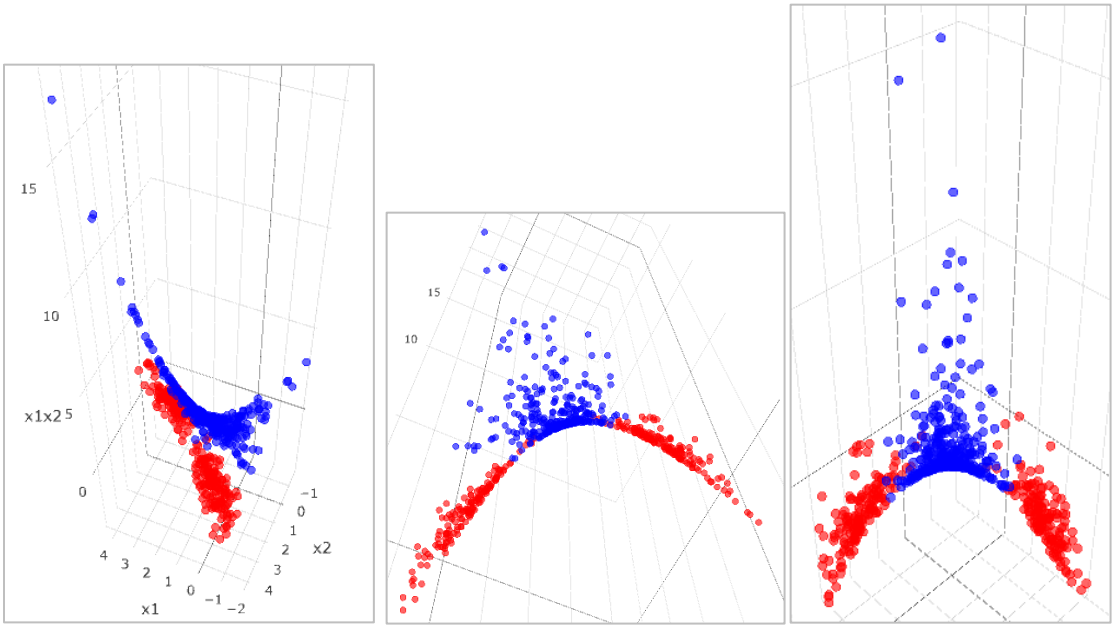
Wow, we got it! … Wait a minute… Rollback a second !

What if we made a simple feature engineering transformation instead of using a non-linear model to solve the problem? Can we ?

For example, let's create a very simple new feature X3 that is the product of X1 and X2, which is a polynomial transformation: X3 = X1 * X2:

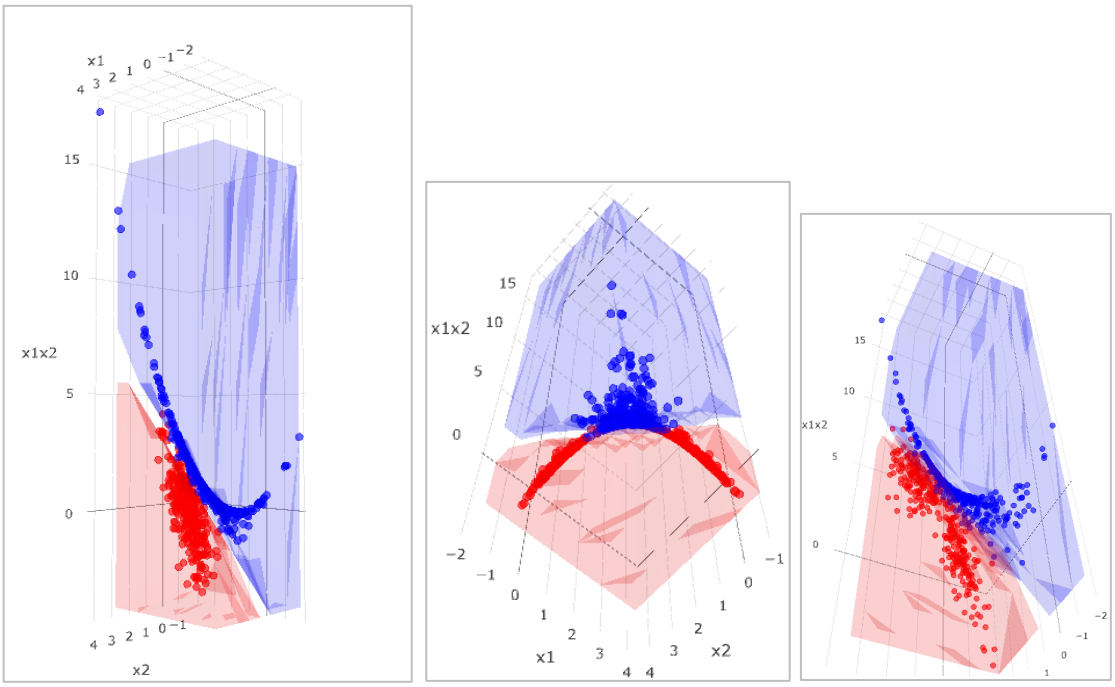
X1	X2	X1*X2 (X3)	Y
1,1	2,1	2,31	Blue
0,5	-0,5	-0,25	Red
3,4	0,5	1,7	Red
3,3	3,9	12,87	Blue
...	...	...	...

We scatterplot again:



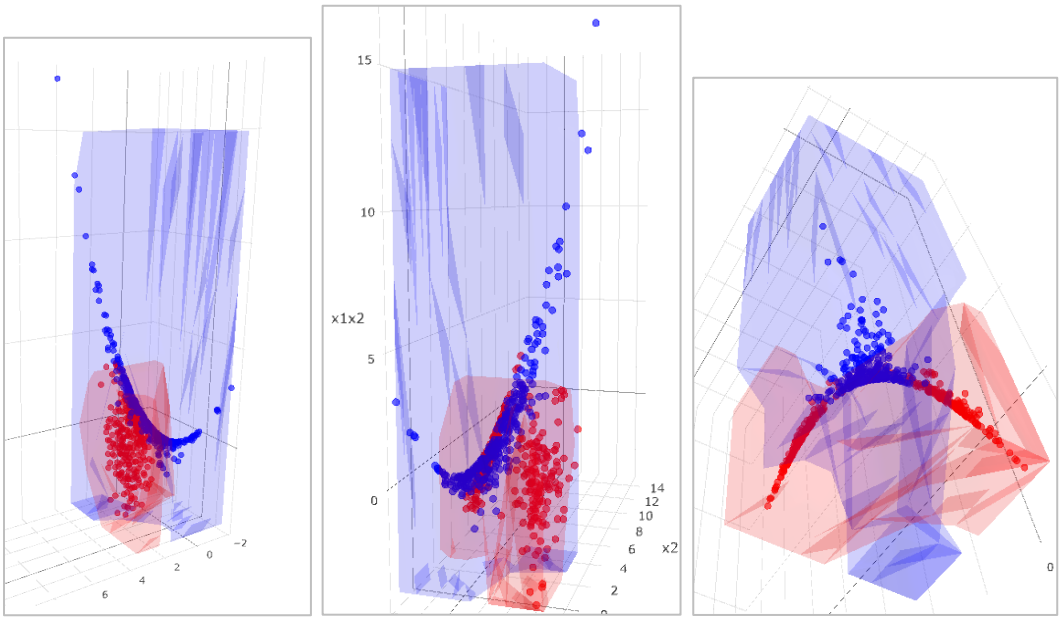
Is this new dataset still non-linear ?

Let’s try a linear logistic regression model on that new engineered dataset:



Well, it seems that after a very simple feature engineering step, the dataset is now linear and logistic regression can find a very good generalization solution to the classification problem!

It also helped the Support Vector Machine to find a better solution (Better precision and recall):



Conclusion: In this random sampled dataset, The Recall of the Logistic Regression model passed from 64,3% to 97,7%, while the SVM recall passed from 97% to 97,3% simply by using this

feature engineering tirick. But when we look at both models functions (Red and blue regions), logistic regression seems to have a better generalized solution to offer while SVM overfitted.

We just helped both models performance by doing a simple feature engineering step. This feature trick have impacted the data in a similar way to SVM Gaussian RBF kernel trick but in lot less complex manner. (*Remainder: "The kernel trick provides a bridge from linearity to non-linearity to any algorithm that can expressed solely on terms of dot products between two vectors by maping the input data into a higher-dimensional space (based on a selected kernel function)"*).

Sometimes, going forward with a more complex model or skipping the feature engineering step to hyper-parameters tuning is not always the best move to make. In this example, we can now use a simple linear logistic regression wich is, a lot less complex and computing intensive than Support Vector Machine and a lot more easier to explain to the business people.

In the ***Demonstrating the power of feature engineering – Part II***, I will demonstrate how to apply the same kind of feature engineering trick but using a more complex dataset with more features.

Hope this workshop have been usefull to you. The jupyter notebook of this experimentation is available on my github with interactive plotly plots: github

Dave Cote, Data Scientist, M.Sc. BI