**Insurance Purchase Prediction Write-up**

**Author: Tossou, Dagbegnon Armand**

Date: 14 December 2020

## 1. Comparing modeling approaches:

The top two models that I selected are:

1. Support Vector Machine (SVM), with rescaled features to the 0-1 range, a '*C*' value of 0.3 and the *RBF* kernel; and
2. Extra Tree (ET), without feature transformation.

I think SVM will perform the best. It has reliably outperformed all the other models that I considered in this exercise, once I rescaled or standardized the data. ET produced the second-best performance on the validation dataset, and it didn't require any feature transformation. Both methods are nonlinear, compared to a model like linear regression. But SVM is considered a low complexity model, while ET qualifies as a high complexity model.

SVM is a classification algorithm that maximizes the margin (i.e., support vector) among class variables. The SVM algorithm plots each data item as a point in n-dimensional space (where n is number of features in the dataset) with the value of each feature being the value of a particular coordinate. Then, it performs classification by finding the hyper-plane that differentiates the two classes very well (look at the below snapshot). SVM is known to be high-performing algorithm with little tuning. It's effective in high dimensional spaces. However, SMV is sensitive to noise, and requires more training time on larger datasets. I've noticed that SVM took much longer to run when working on this problem.

ET is an ensemble machine learning algorithm that combines the predictions from many decision trees. This algorithm works by creating a large number of unpruned decision trees from the training dataset. Predictions are made by using *majority voting* in the case of classification problems. ET can be computationally faster and can produce a higher performance in presence of noisy features.

## 2. Some things to consider in a business context:

- Trade-offs between model performance and resource allocation: for instance, I used 80/20 split of the dataset for modeling and validation. This is standard practice, but I could consider other splits, such as 70/30, two-third/one-third or whatever best practice the company has in place. These choices could have serious implications for resource allocation (time, equipment, money, etc.) especially when dealing with very large datasets with millions/billions of observations and/or variables.
- Handling of missing values: in a business context, I'd have in-depth domain expertise that informs my understanding of the dataset: what each feature represents, the best way to measure each feature, other alternative formats to consider for the most relevant features, what missing values represent for each feature (genuine blanks, data entry errors, "Don't know" or "Prefer not to answer" cases, features that are too difficult to measure, ...), etc. Missing values may be very important to models, and as such there exists numerous ways and techniques to handle them, so domain expertise can help me decide the best ways to go: imputing values (min, max, median, mode, etc.), dropping observations, etc.
- Feature engineering: again, with domain expertise, I could come up with new features that could greatly help machine learning models. For instance, I could consider generating unique features such as by transforming or combining existing ones. I could try polynomial (cross) features (square terms,

interactions among features, etc.) to capture relationships among independent variables and help decrease bias of machine learning models, if doing so doesn't contributing to overfitting.

- Model complexity: more complex machine learning models (e.g., deep learning) could also prove more promising than simpler models. But they also require more resources, such as computing power, tuning, training time, especially when dealing with big data. In a business setting, it might be economically feasible to improve processing time through parallelized/distributed computing solutions.
- Parameter tuning: this is quite an art in machine learning as the optimal parameters of a model can depend on many things. Tuning helps with the bias-variance trade-off. I look forward to familiarizing myself with established best practices at the company in this area.
- Multicollinearity issues: I considered eliminating explanatory features with VIF > 10. This would result in a potential loss of data for the benefit of better performing machine learning algorithms. Again, I'm sure that the company has some best practices around this scenario.
- Oversampling or undersampling: the target (i.e., y variable) shows some imbalance with a class distribution of roughly 1:4 in the minority (label '1') class to the majority class (label '0'). Luckily, the imbalance is not a case of severe skew in the class distribution, such as a 1:100 or 1:1000 ratio. Therefore, I did not bother to consider methods such as random oversampling and undersampling, or Synthetic Minority Oversampling Technique (SMOTE). Given more resources as a the company Data Science Team member, I could come up with further improvements to this predictive model.

**3. Explaining to a business partner the concept that a model is better than the other:**

This exercise requires some empathy, i.e., putting myself in the shoes of the other person. It's important to keep our conversation at the conceptual level and avoid falling into the trap of super technical machine learning / data science jargon. Some of these basic principles are straightforward. For example, everyone will understand that every model relies on some fundamental assumptions to work. Also, the idea of 'optimization' or gradual improvement in a process is very intuitive. In our conversation, I would try employing as much of well understood business jargon as possible and drawing upon simple real-life examples to illustrate technical scenarios/trade-offs. If I were talking with a finance/accounting person for instance, I could use some examples related to investment strategies like selecting an optimal portfolio, or cashflow management. Now from a purely technical standpoint, the trick here is to talk about how I improved model accuracy gradually whilst preventing high variance (or overfitting) problems. I can focus on clarifying some basic logics behind feature engineering/ transformation, spot-checking several techniques to see which perform better under certain circumstances, etc. I can emphasize how I used grid search to tune hyperparameters. I can explain the basic concepts behind using k-fold cross validation or pipelines in the machine learning workflows as ways to avoid data leakage and subsequent overfitting. Communicating highly technical concepts to non-technical stakeholders takes some skill and practice. It's always a learning experience for both parties.