Machine Learning Done Wrong

Statistical modeling is a lot like engineering.

In engineering, there are various ways to build a key-value storage, and each design makes a different set of assumptions about the usage pattern. In statistical modeling, there are various algorithms to build a classifier, and each algorithm makes a different set of assumptions about the data.

When dealing with small amounts of data, it's reasonable to try as many algorithms as possible and to pick the best one since the cost of experimentation is low. But as we hit "big data", it pays off to analyze the data upfront and then design the modeling pipeline (pre-processing, modeling, optimization algorithm, evaluation, productionization) accordingly.

As pointed out in my previous post, there are dozens of ways to solve a given modeling problem. Each model assumes something different, and it's not obvious how to navigate and identify which assumptions are reasonable. In industry, most practitioners pick the modeling algorithm they are most familiar with rather than pick the one which best suits the data. In this post, I would like to share some common mistakes (the don't-s). I'll save some of the best practices (the do-s) in a future post.

1. Take default loss function for granted

Many practitioners train and pick the best model using the default loss function (e.g., squared error). In practice, off-the-shelf loss function rarely aligns with the business objective. Take fraud detection as an example. When trying to detect fraudulent transactions, the business objective is to minimize the fraud loss. The off-the-shelf loss function of binary classifiers weighs false positives and false negatives equally. To align with the business objective, the loss function should not only penalize false negatives more than false positives, but also penalize each false negative in proportion to the dollar amount. Also, data sets in fraud detection usually contain highly imbalanced labels. In these cases, bias the loss function in favor of the rare case (e.g., through up/down sampling).

2. Use plain linear models for non-linear interaction

When building a binary classifier, many practitioners immediately jump to logistic regression because it's simple. But, many also forget that logistic regression is a linear model and the non-linear interaction among predictors need to be encoded manually. Returning to fraud detection, high order interaction features like "billing address = shipping address and transaction amount < $50" are required for good model performance. So one should prefer non-linear models like SVM with kernel or tree based classifiers that bake in higher-order interaction features.

3. Forget about outliers

Outliers are interesting. Depending on the context, they either deserve special attention or should be completely ignored. Take the example of revenue forecasting. If unusual spikes of revenue are observed, it's probably a good idea to pay extra attention to them and figure out what caused the spike. But if the outliers are due to mechanical error, measurement error or anything else that's not generalizable, it's a good idea to filter out these outliers before feeding the data to the modeling algorithm.

Some models are more sensitive to outliers than others. For instance, AdaBoost might treat those outliers as "hard" cases and put tremendous weights on outliers while decision tree might simply count each outlier as one false classification. If the data set contains a fair amount of outliers, it's important to either use modeling algorithm robust against outliers or filter the outliers out.

4. Use high variance model when n<<p

SVM is one of the most popular off-the-shelf modeling algorithms and one of its most powerful features is the ability to fit the model with different kernels. SVM kernels can be thought of as a way to automatically combine existing features to form a richer feature space. Since this power feature comes almost for free, most practitioners by default use kernel when training a SVM model. However, when the data has n<<p (number of samples << number of features) -- common in industries like medical data -- the richer feature space implies a much higher risk to overfit the data. In fact, high variance models should be avoided entirely when n<<p.

5. L1/L2/... regularization without standardization

Applying L1 or L2 to penalize large coefficients is a common way to regularize linear or logistic regression. However, many practitioners are not aware of the importance of standardizing features before applying those regularization.

Returning to fraud detection, imagine a linear regression model with a transaction amount feature. Without regularization, if the unit of transaction amount is in dollars, the fitted coefficient is going to be around 100 times larger than the fitted coefficient if the unit were in cents. With regularization, as the L1 / L2 penalize larger coefficient more, the transaction amount will get penalized more if the unit is in dollars. Hence, the regularization is biased and tend to penalize features in smaller scales. To mitigate the problem, standardize all the features and put them on equal footing as a preprocessing step.

6. Use linear model without considering multi-collinear predictors

Imagine building a linear model with two variables X1 and X2 and suppose the ground truth model is Y=X1+X2. Ideally, if the data is observed with small amount of noise, the linear regression solution would recover the ground truth. However, if X1 and X2 are collinear, to most of the optimization algorithms' concerns, Y=2*X1, Y=3*X1-X2 or Y=100*X1-99*X2 are all as good. The problem might not be detrimental as it doesn't bias the estimation. However, it does make the problem ill-conditioned and make the coefficient weight uninterpretable.

7. Interpreting absolute value of coefficients from linear or logistic regression as feature importance

Because many off-the-shelf linear regressor returns p-value for each coefficient, many practitioners believe that for linear models, the bigger the absolute value of the coefficient, the more important the corresponding feature is. This is rarely true as (a) changing the scale of the variable changes the absolute value of the coefficient (b) if features are multi-collinear, coefficients can shift from one feature to others. Also, the more features the data set has, the more likely the features are multi-collinear and the less reliable to interpret the feature importance by coefficients.

So there you go: 7 common mistakes when doing ML in practice. This list is not meant to be exhaustive but merely to provoke the reader to consider modeling assumptions that may not be applicable to the data at hand. To achieve the best model performance, it is important to pick the modeling algorithm that makes the most fitting assumptions -- not just the one you're most familiar with.