The Data Science Life Cycle: From Client to Production

Pipeline and Challenges

by Shaked Markovich

The project

- The problem: Insured companies sufferers from lapsers (churn), losing those policies can be summed up to hundreds of thousands of dollars.
- The goal is to predict, in the next months, which policy will lapse so the company can try to retain the policy holder.

- With annual lapse rate of 4%, we achieved a model with precision of 45% and recall of 35%.

Pipeline

- Data from Multiple Sources:
  - Client-supplied data consists of several tables, each with approximately 1 million rows (both static and event-wise).
  - External data enrichment, including census and financial information, is incorporated after applying correlation analysis and clustering.
- Data is aggregated to enable 1-month forecasting, with preprocessing steps such as creating lags, cumulative sums, and time-series cross-validation for model compatibility.
- A grid search using Optuna is conducted to identify the optimal model.
- Model performance is validated using metrics PR-AUC and F1-score.
- Interpretability is enhanced through SHAP analysis.
- Calibrate prediction probabilities due to imbalance data.
- Monitor metrics, data distributions (data drift) and feature importance.

Challenges

- Low lapse rate (imbalanced data)
- Limited availability of personal information
- Classifying time series data (Dependency between rows)
- Data inconsistencies and fluctuations over time (Data Drift)
- Prediction to a specific span of time
- Variety of policies durations (from month to 80 years)

XGBoost

pros:

- Easy and intuitive to explain with feature importance and SHAP.
- Using boosting for better accuracy.
- Handle missing values.
- No need to scale (although it helps).
- Efficient with Large Datasets: It’s optimized for both speed and performance.
- Regularization: Built-in regularization terms help prevent overfitting.
- Handle imbalanced data with `scale_pos_weight` Parameter.

cons:

- Easy to overfit
- Requires Extensive Tuning

XGBoost

parameters:

- **learning_rate/eta:** (0 to 1) Controls the update step size during each iteration, shrinking the effect of each tree to prevent overfitting. It's a weight factor for the newly added tree in the ensemble.
- **max_depth**: Maximum allowed depth of a tree. As the depth increases, the model becomes more complex and potentially overfits the data.
- **min_child_weight**: (minimum sum of instance weight H) Minimum sum of hessian values for a split to occur in a tree. Increasing this value prevents overfitting by making the tree more conservative.
- **lambda**: L2 regularization term on weights, applied as a part of the Gain calculation. Helps control the model complexity.
- **gamma/min_split_loss**: Minimum loss reduction required to partition a leaf node further. Controls the tree's growth and complexity.