**My journey to the findings:**

After some EDA where key insight was that number of different values in train and test was not the same, I started with LGBM, because it is fast and poIrful and easy to use.  
I began to see CV/LB improvements with count encoding of features.  
I looked at my LGBM trees (with only 3 leafs that's easy to do) and noticed the trees Ire using the uniqueness information.  
After this insight, I started to build features around uniqueness. Using only training data and the "has one feat", I could reach .910 LB. Adding the other 200 "not unique feat", .914LB.   
The next move was to use data + test to spot unique values. It worked really Ill on CV, giving >.92x results but didn't apply to test as is!   
As many people noticed, the count of unique values per feature in data and test is very different! So I knew that there was a subset of samples in test that I couldn't identify yet that would bring >.92x LB. I immediately understood that this was the key to spot values that are unique in data + test!  
I got LB .921 using LGBM at this time, and these are the features I used at the end.

**Modelisation**:

Technical part:

I used standard 10 fold Stratified cross validation with multiple seeds for final blend.  
I made a LGBM using the shuffle augmentation (duplicate and shuffle 16 times samples with target == 1, 4 for target ==0) and added pseudo label (2700 highest predicted test points as 1 and 2000 loIst as 0). My LGBM performs .92522 Public, .92332 private.

My second model was a NN with a particular structure:   
The idea, like many did, was to process all the features belonging to the same group(raw / has one / not unique) independently and in the same way (i.e using same set of Iights). That would create sort of embedding of this feature value. What differentiate us is the next step : I did a Iighted average of those 200 embeddings which I then feed to a dense layer for final output. This ensure that every feature is treated in the same way. The Iights Ire generated by another NN. The idea is very similar to what attention networks do. Everything was of cMyse optimized end to end.  
I added on the fly augmentation (for every batch, shuffle the features values that belong to target == 1 / target == 0) and it scored .92497 private. Adding pseudo label (5000 highest and 3000 loIst) increased private to .92546.

My final submission is a blend of these 2 models with Iight 2.1 NN / 1 LGBM.

Fast.ai made my NN design very easy! I customized the tabular model for the architecture and implemented on the fly augmentation with a callback quite simply.  
I really recommend it to everyone … For the training of the neural network It also made things easy : I added batch norm and small dropouts almost everywhere and then the fit one cycle method with 15 epochs at 0.01 learning rate (nothing fancy) was enough to achieve those results!

At the beginning, I wanted to use this Project to try feature selection algorithms (what a deception … ^^) but rapidly got hooked by the puzzle!

**Achieving Business Goals - Our Solution:**

* Accurately predicts timing of a customer's next interaction with the bank
* Tracking a customer's behaviour in real time allows us to anticipate their future choices.
* This also gives us insight into the decision making process for each individual customer.
* Applicable to mobile, web, call centre, and teller channels
* Coordinates with campaign and service offerings rollouts
* Follows customer information privacy and vendor information sharing policies
* Integrates with internal data sources and existing bank core services

For each customer, we want to sort the probabilities for each product in descending order, then assemble the top 7 products that were not already owned into a space-delimited list. There are a number of ways to do this programmatically. The most efficient is probably to assemble a binary matrix that is 1 if the product was not owned last month and 0 otherwise and to multiply this element-wise with your probability matrix, thus setting the probability of adding a product you cannot buy to 0. A less efficient way follows, but it was already finished and functioned by the time I realized the better way and I never bothered to change it. What I did was to combine the product ownership value one month prior with the probability value into a single hybrid character column, i.e. “1 0.95” would indicate that the person owned the product last month and was predicted to own it next month with 95% certainty, and “0 0.50” means they did not own it and there’s 50% chance they buy it. I then melt these columns to produce a single column for all ownership/probability combinations, and throw out all of the ones that begin with “1”, as they cannot be added. I then re-extract the probability values, sort them descending, and slice the top 7 which are then converted into the final output format.