

How to win a Kaggle competition

Part 5: Case study

Chris Chen@Calgary Data Science Academy

AVITO DEMAND PREDICTION CHALLENGE

Predict demand for an online classified ad

Dataset

- ▶ Huge volume of data
 - ▶ 1.5M training samples, 0.5M testing samples(1.2GB)
 - ▶ 1.9M images (68GB)
 - ▶ Supplemental unlabeled data (17.7GB)
- ▶ Rich data types
 - ▶ Numeric: price, item sequence
 - ▶ Categorical: user id, user type, city, region, category, parent category, image class
 - ▶ Datetime: activation date etc.
 - ▶ Image
 - ▶ Text: title, description, parameters
- ▶ Image and text are CRITICAL to this challenge
- ▶ Infinite possibility for feature engineering

Well taken, Authentic Photos



Too Glossy



Authentic



Poor Quality

Believable and Informative Description

Description:
***AMAZING WATCH
FOR SALE!!!!***

DON'T MISS THIS
DEAL. IT'S THE DEAL
OF THE CENTURY!!

Unlikely

Description:
I have an adjustable
Chaleur D'Animale
Watch for sale.

It's never been worn
and still in the original
box. Battery included.

Informative

Description:
fancy watch for sale

no low ball offers, cash
and carry

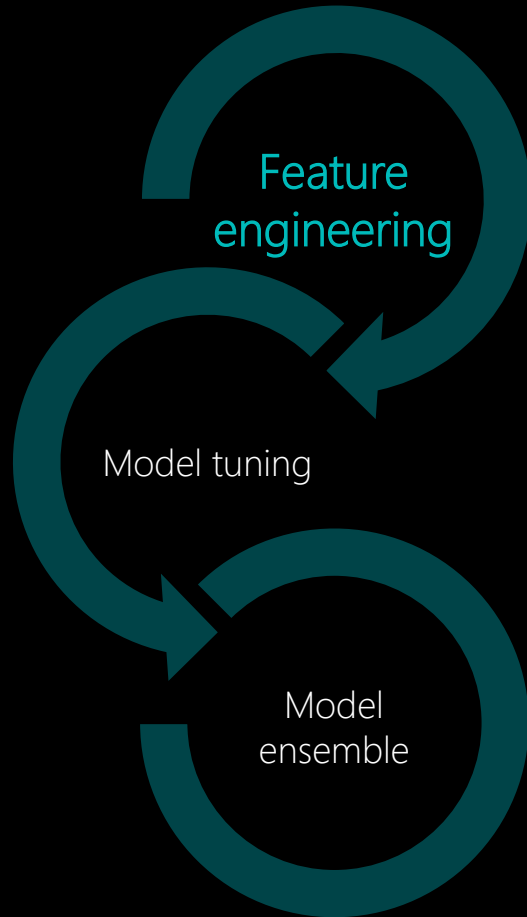
Poor Quality

GBDT vs. Deep Learning?

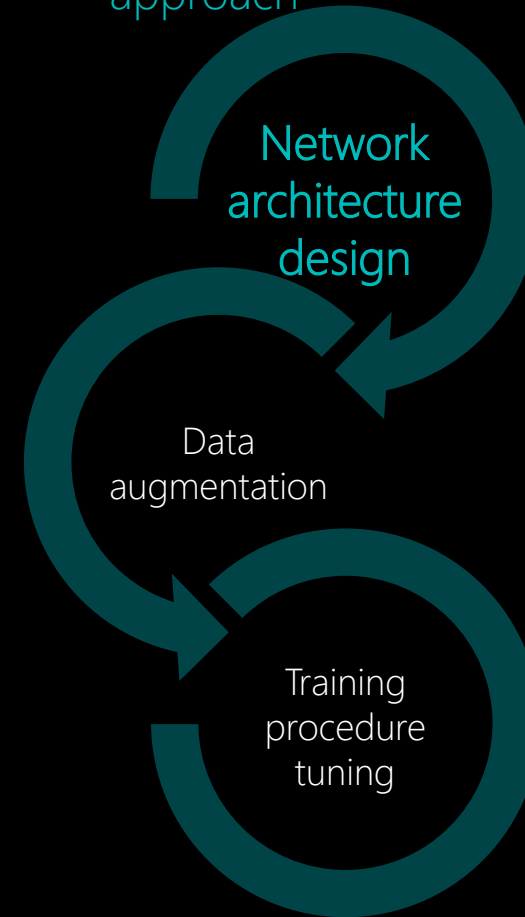
- GBDT (LightGBM, XGBoost and CatBoost) is still the go-to choice by most winning teams as they typically work better on datasets that consist heterogeneous features.
- However, the champion team "Dance with Ensemble" surprised the community by winning the Avito competition with a [multi-modal-deep-learning](#) solution which solely could have ranked top 10 in this competition. Their team lead, Kaggle Grandmaster [@Little Boat](#), shared the solution in a post-competition [discussion](#).
- I was able to replicate the similar approach which could have helped us get a Gold medal (#11) instead of a Silver one (#32)

Conventional approach vs. Deep Learning approach

Conventional approach



Deep learning approach



Categorical features

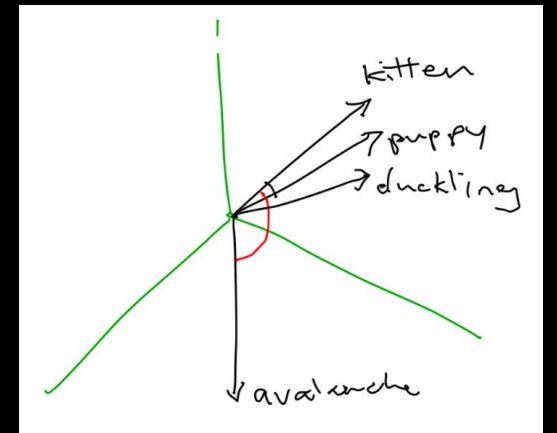
Conventional approach

- One-hot encoding
- Label encoding
- Count encoding
- Target encoding
-

Deep learning approach

- Embedding for categorical features
 - Inspired by word embeddings
 - The categorical embedding layers are trained along with other features with a unified optimization objective.

Categories	Embeddings
puppy	[0.9, 1.0, 0.0]
dog	[1.0, 0.2, 0.0]
kitten	[0.0, 1.0, 0.9]
cat	[0.0, 0.2, 1.0]



An example of three dimensional embedding, retrieved from [An Introduction to Deep Learning for Tabular Data](#)

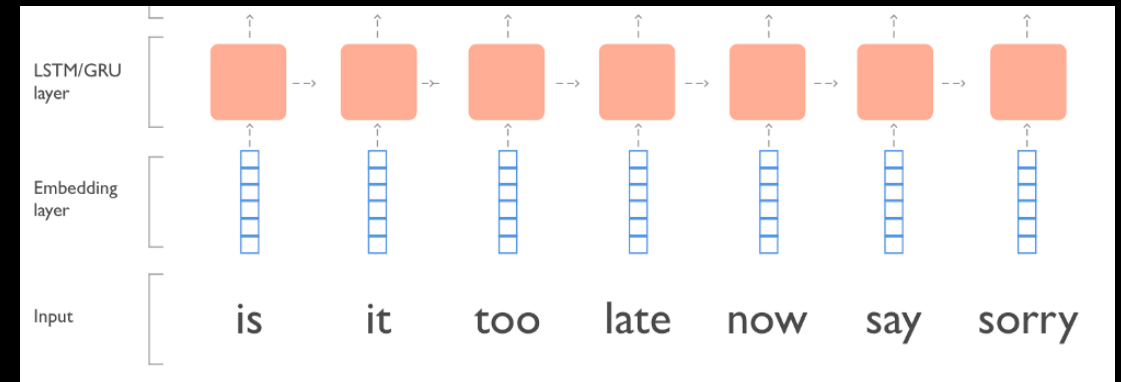
Text

Conventional approach

- Word frequency
 - Vector count
 - TF-IDF
- Word embedding
 - Have to aggregate embeddings of words to represent a sentence. Information loss is possible after aggregation.

Deep learning approach

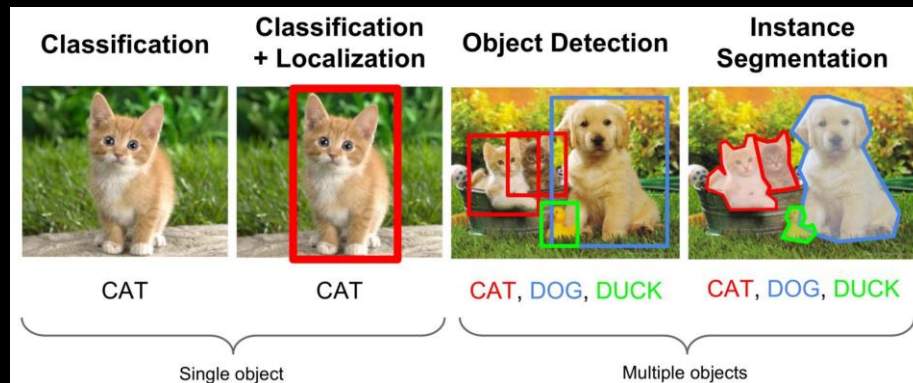
- Word embedding + sequence model (RNN/LSTM/GRU etc.)
 - Embeddings of each word in a sentence are stacked and sent to sequence models hence there's no information loss.



Image

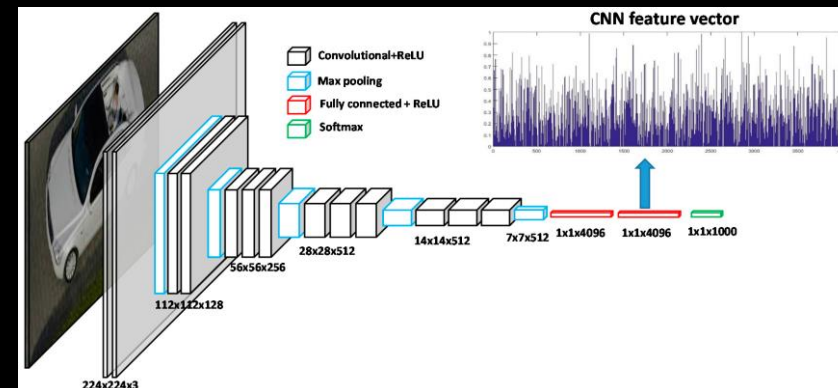
Conventional approach

- Classical CV features
 - size, dullness, whiteness, colors, blurriness, uniformity
- Features extracted from pre-trained models
 - Classification result from vgg, resnet, inception, xception, densenet etc.
 - Objects detected by yolo, ssd etc.

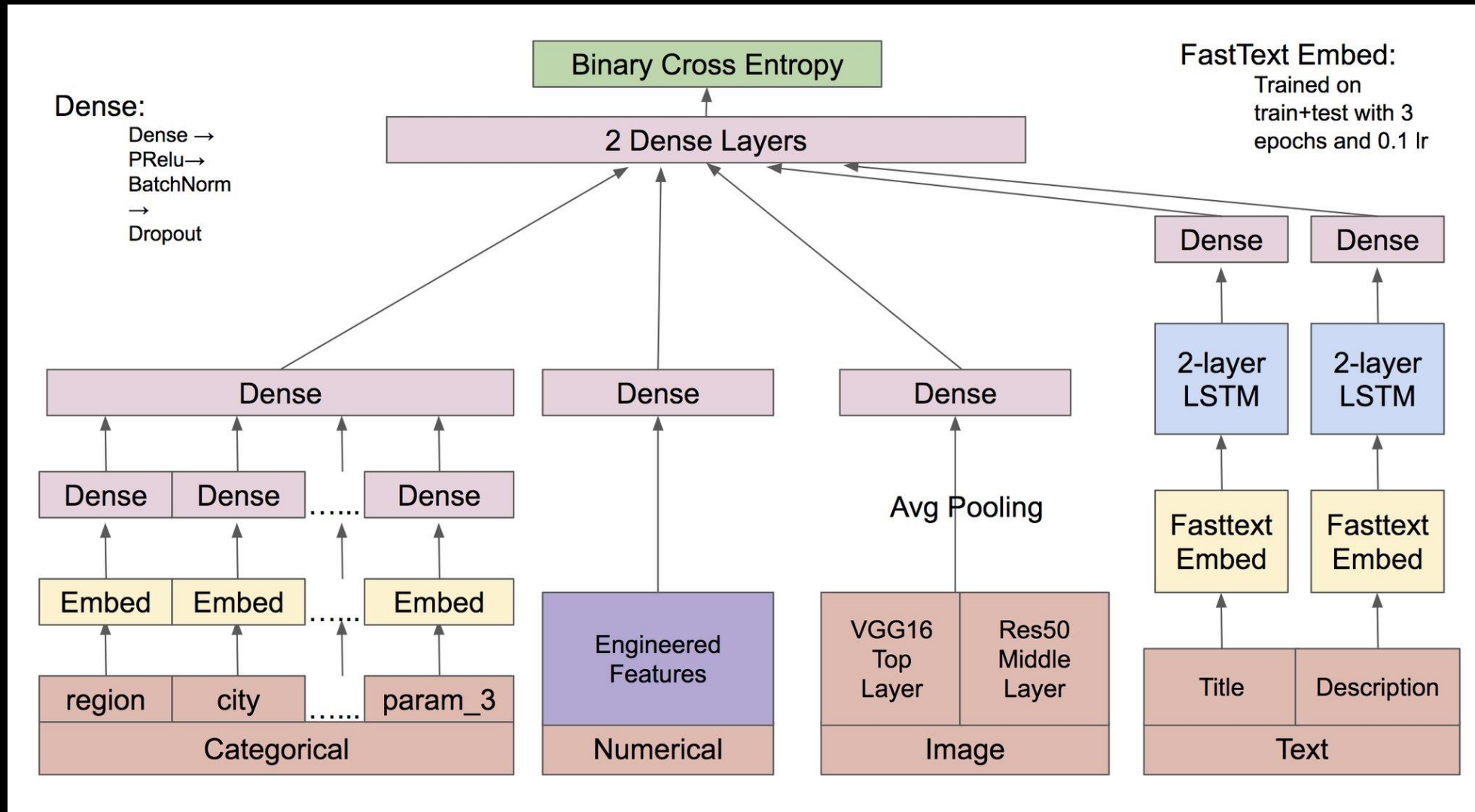


Deep learning approach

- Using CNN models as a building block
 - Freeze CNN layers: using pretrained CNN models as-is
 - Unfreeze CNN layers: incrementally re-train CNN models



Deep learning network architecture for Avito



NN architecture inspired by and retrieved from Little Boat's sharing at <https://www.kaggle.com/c/avito-demand-prediction/discussion/59880>



Santander customer transaction prediction

CAN YOU IDENTIFY WHO WILL MAKE A TRANSACTION?

Challenge

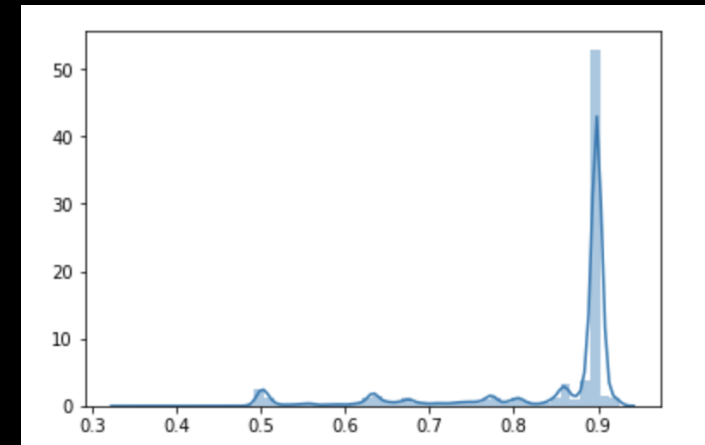
At Santander our mission is to help people and businesses prosper. We are always looking for ways to help our customers understand their financial health and identify which products and services might help them achieve their monetary goals.

Our data science team is continually challenging our machine learning algorithms, working with the global data science community to make sure we can more accurately identify new ways to solve our most common challenge, binary classification problems such as: is a customer satisfied? Will a customer buy this product? Can a customer pay this loan?

In this challenge, we invite Kagglers to help us identify **which customers will make a specific transaction in the future**, irrespective of the amount of money transacted. The data provided for this competition has the same structure as the real data we have available to solve this problem.

The **most** participated competition, and arguably, this is the **easiest** competition and the **hardest** competition in Kaggle's history. in Kaggle's history:

- **8,802** teams (10,000+ individuals) participated.
- A simple logistic regression or naïve Bayesian classifier without tuning can get you an ROC-AUC 0.88.
- 62% teams crowded between 0.89~0.90.
- But only **5%** teams could go beyond ROC-AUC 0.90.
- Our team (rapids.ai) won a Gold medal (#17)



Distribution of AUC-ROC on leaderboard

Dataset


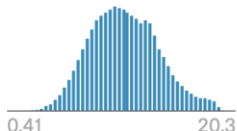
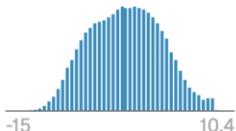
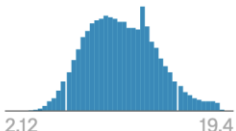
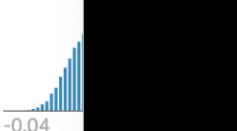
You are provided with an anonymized dataset containing 200 numeric feature variables, the binary target column, and a string ID_code column. The task is to predict the value of target column in the test set.

Everything sounds pretty straightforward so far until you found that:

- There's **NO missing values**
- All the features are almost perfectly **normally distributed**.
- All the features are almost **non-correlated** with each other.
- Train and test follow the same distributions.

After two months Kaggle got crazily frustrated by the facts that:

- Simple algorithms such as Logistic Regression and Naïve Bayesian performed almost as well as complex ones such as XGBoost and LightGBM (0.88~0.89).
- None of the feature engineering tricks worked.
- Best LightGBM model only requires 2 leaves.

A ID_code	# target	# var_0	# var_1	# var_2	# var_3
200000 unique values					
train_0	0	8.9255	-6.7863	11.9081	
train_1	0	11.5006	-4.1473	13.8588	
train_2	0	8.6093	-2.7457	12.0005	
train_3	0	11.0604	-2.1518	8.9522	
train_4	0	9.8369	-1.4834	12.8746	

Is this going to be another *"stacking" war*?

What's more confusing (or scary).....

There are **SYNTHETIC** samples in the **test** data set!

Observations:

The distribution of the number of unique values (across features)

Assumptions:

"**Faked**" samples were generated by a sort of simulation process that uses distribution estimated from some samples which were not included in train or test.

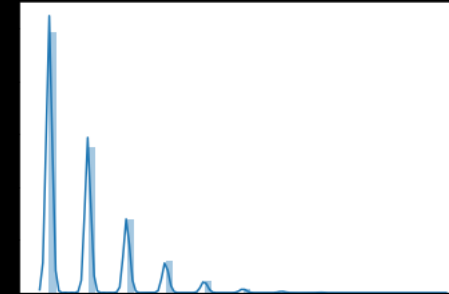
If this holds true, then given a sample we can go over its features and check if the feature value is unique. If **at least one of the sample's features is unique**, then the sample must be a **real** sample.

Experiments:

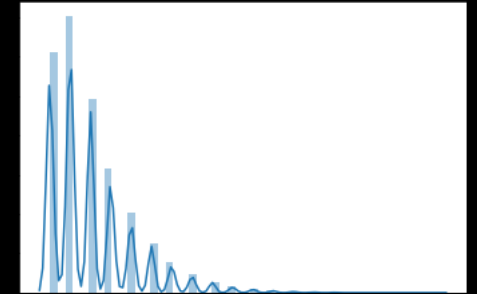
The following codes show how to count "faked" samples and "real" samples:

```
unique_samples = []
unique_count = np.zeros_like(df_test)
for feature in tqdm(range(df_test.shape[1])):
    _, index_, count_ = np.unique(df_test[:, feature], return_counts=True, return_index=True)
    unique_count[index_[count_ == 1], feature] += 1
# Samples which have unique values are real the others are fake
real_samples_indexes = np.argwhere(np.sum(unique_count, axis=1) > 0)[: , 0]
synthetic_samples_indexes = np.argwhere(np.sum(unique_count, axis=1) == 0)[: , 0]
print(len(real_samples_indexes))
print(len(synthetic_samples_indexes))
```

Distributions of unique values: train



test



Results:

Using this approach we can identify that there are **100,000** "faked" samples **100,000** "real" samples in the test dataset. This cannot be a coincidence!!!

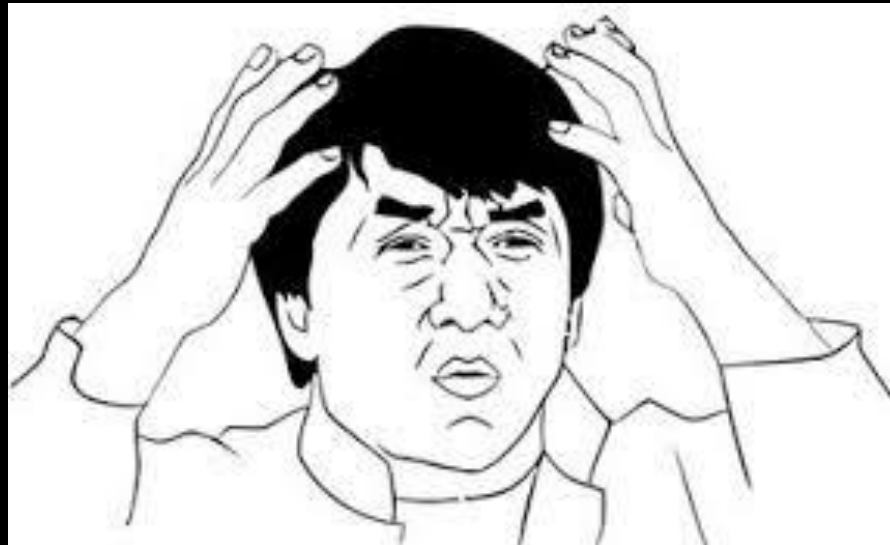
100000

100000

Now I started to be worried about my hairs because

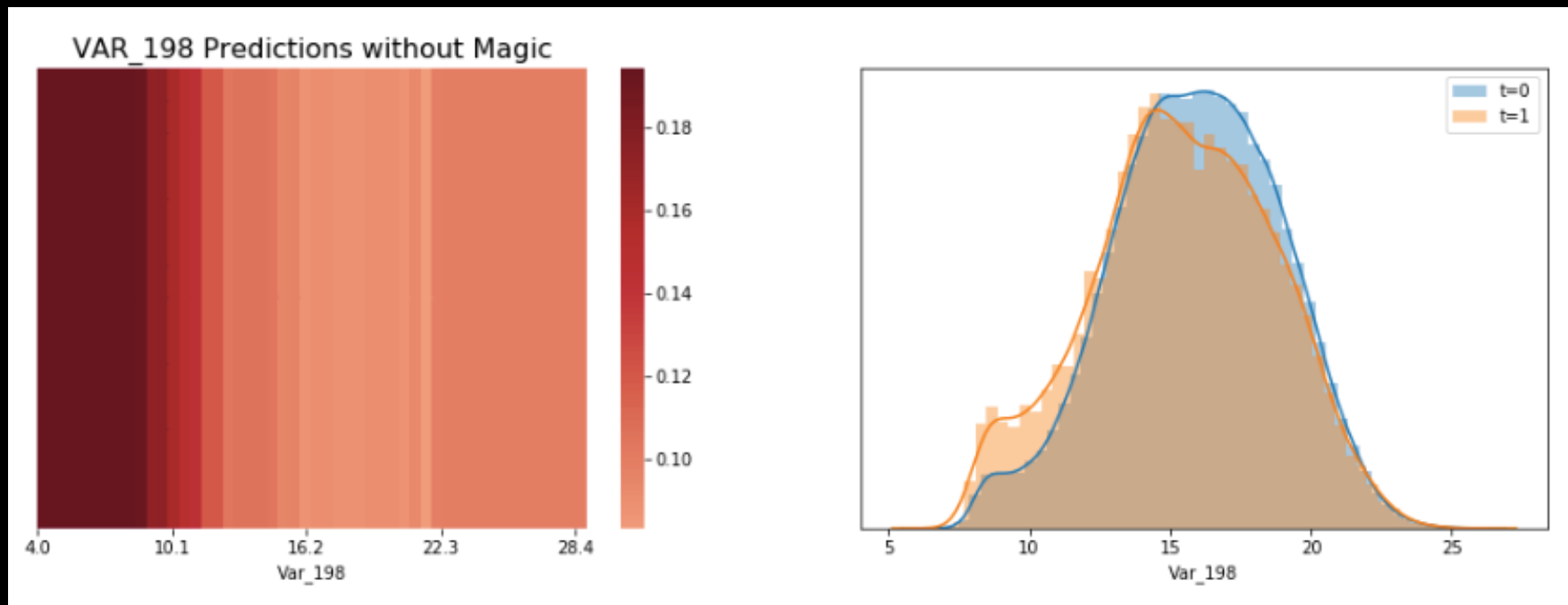
Randomly shuffling the data doesn't change the result!!!

For each sample in the train dataset, if we randomly shuffle the value of features, e.g. switch [var1, var2, .. var199] to [var 33, var 127.... var85], then train a model using the shuffled data instead of the original train data, guess what, we would end up with almost the same ROC-AUC! But a more important question is: how can we leverage this finding to improve our model?



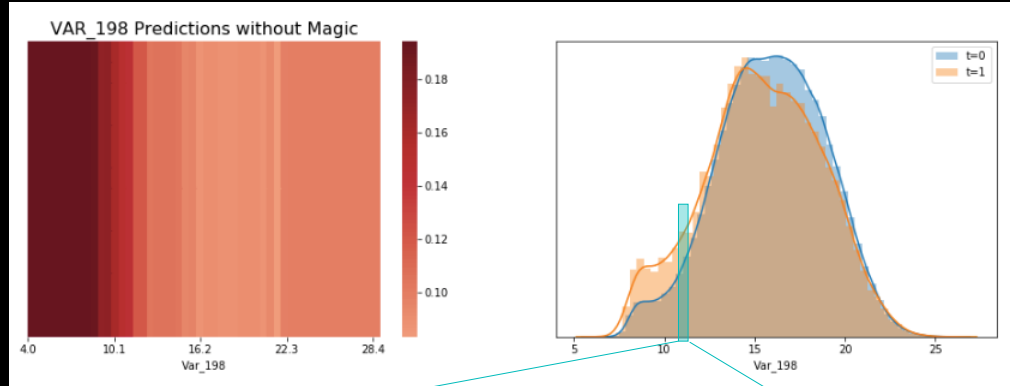
Let's a closer look at the data

Let's randomly pick up a feature, say var_198, and analyze it. As mentioned before, it follows an almost perfect normal distribution. However, if we color code the target with **orange** being 1 and **blue** being 0, then we would observe mixed gaussian distributions, which can also be observed on other features. If we simply draw a straight line to classify the data it would probably be around 13. In fact, a two-leaves LightGBM model would predict *target=0.18* for *var_198<13* and *target=0.10* otherwise.



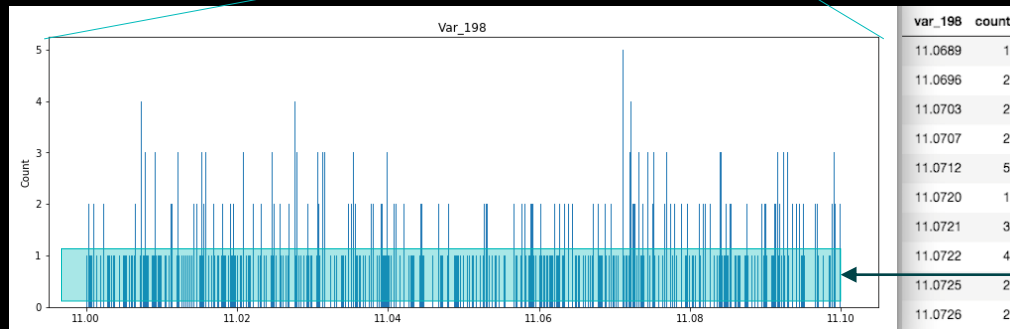
LGBM "divides" the histogram with **vertical lines** because LGBM does not see **horizontal** differences.

The "magic" feature



A histogram plot places multiple values into a single bin and produces a smooth picture. If you place every value in its own bin, you will have a jagged picture, where bars change heights from value to value. *Some values are unique, some values occur many times.*

If we create a histogram with one value per bin and zoom in on $11.0000 < x < 11.1000$, we will see that value 11.0712 occurs 5 times and its close neighbor 11.0720 occurs only once.



These counts on individual values are the "*magic*" feature in these competition.

How does the "magic" work?

For each variable, we make a new feature (column) whose value is the number of counts of the corresponding variable. An example of this new column is displayed above next to the histogram. When LGBM has this new feature, it can now "divide" the histogram with horizontal lines in addition to vertical.

Now that LGBM predicts *target=0.1 when $var_198 < 13$ AND $count=1$. When $var_198 < 13$ AND $count > 1$, it predicts $target=0.36$.*

This improvement (using the magic) causes validation AUC to become 0.551 as opposed to 0.547 when using var_198 alone.

Maximizing the “Magic” feature

With the magic feature only, our team boosted the AUC from 0.900 (top 400) to 0.910 (top 200). In the meantime, we’ve also applied a few additional tricks which eventually got us a Gold medal:

1. Reverse Missing Value Imputation

- ▶ Recall we made and proved an assumption that there are “fake” samples in test data set that were generated using distributions of other samples (out of train and test). Is it possible that some of the values, as opposed to samples, in the train data set were actually results of missing value imputations?
- ▶ With that assumption, for each original feature we added a new feature where it keeps the original value if the unique count of the value is greater than 1, or NULL if the unique count of the value is **equal to 1**. This helps boost the AUC from 0.910 to 0.918(top 100).

2. Removing “faked” samples

- ▶ Removing “faked” samples from the test data set when calculating the count of unique values. Helped AUC boosted to 0.920 (top 70)

3. Data Shuffling as an Augmentation

- ▶ Were you still puzzled and confused by why suffering the data would still work? STOP doing that and try to TAKE ADVANTAGE of the finding!
- ▶ A Data Augmentation scheme created by Grand Master Jiwei Liu helped the AUC further improved from 0.920 to 0.922 (top 50).
- ▶ The idea is quite straightforward: since shuffling the features wouldn’t affect the result much, let’s just perform the feature shuffling on the original train data for K times (we used 5 for K), then append the shuffled data to the original train data. Now we get an “augmented” dataset!

4. Multi-head 1D CNN model

- We also experimented quite a few deep learning models and ended up finding an 1D CNN with multi-head performed the best. Adding the CNN model for ensemble helped us get our final rank.

