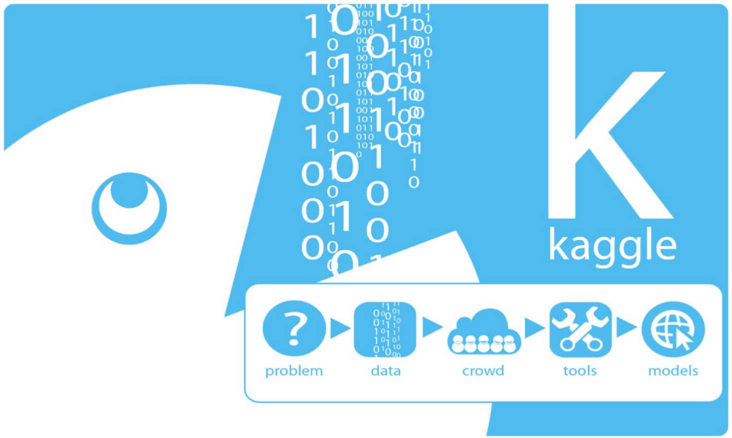
Santander Customer Transaction Prediction

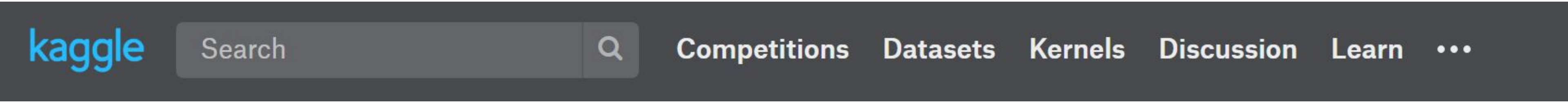
Kameron MacKenzie & Sarah Yurick

Introduction

- Compete in competition hosted by Kaggle, a large online data science community
- Goal to predict customer transactions using data provided by Santander Bank
- **Our DSCI 133 project goals:**
  - Analyze public approaches to task (Kernels)
  - Learn new machine learning and data analysis methods
  - Develop unique, competitive solutions
  - Reflect, learn, and improve



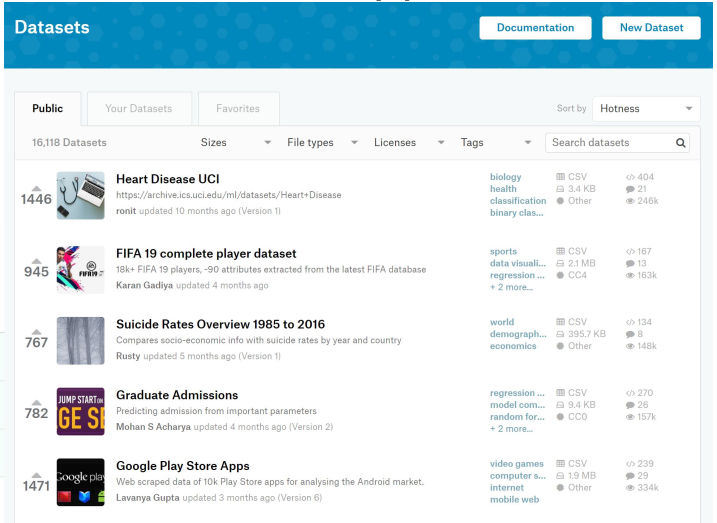
Kaggle



- **Competitions:** Constantly being introduced, offer leaderboards & rewards
- **Datasets:** Enormous database of tagged, sortable, and public access datasets
- **Kernels:** Environment, input, code, and output all in one (Similar to ipynb)
- **Discussion & Learn:** Active forums, data science courses, community engagement

In the money · Gold · Silver · Bronze

#	△pub	Team Name	Kernel	Team Members	Score	Entries	Last
1	—	**Wizardry**			0.92573	91	14d
2	—	三人寄れば文殊の知恵（本当か？			0.92552	45	14d
3	—	**Rock, Physics, Science!**			0.92467	101	14d



Santander

- Santander Bank, subsidiary of the Spanish Santander Group
- Offering $65,000 in prize money
- Competition data has similar structure to actual data Santander data science team works with

Problem

- **Binary Classification -** Common issue in banking: Will a customer buy a product? Will a customer be able to repay a loan? Is a customer satisfied with their service?
- Offering $65,000 in prize money, limited to 2 final submissions per team
- **“Identify which customers will make a specific transaction in the future, irrespective of the amount of money transacted”**

[0] - No transaction

[1] - Transaction

The Data

- Three files provided by Santander and Kaggle
  - train.csv, test.csv, Sample_submission.csv
- Anonymized data: 200 columns identified only by number
- 200,000 rows, each representing an individual & containing:
  - "ID_code" - String
  - "var_0" - "var_199" - Numerics

Using .head() on training set

	ID_code	target	var_0	var_1	var_2	var_3	var_4	var_5	var_6	var_7	var_8	var_9
0	train_0	0	8.9255	-6.7863	11.9081	5.0930	11.4607	-9.2834	5.1187	18.6266	-4.9200	5.7470
1	train_1	0	11.5006	-4.1473	13.8588	5.3890	12.3622	7.0433	5.6208	16.5338	3.1468	8.0851
2	train_2	0	8.6093	-2.7457	12.0805	7.8928	10.5825	-9.0837	6.9427	14.6155	-4.9193	5.9525
3	train_3	0	11.0604	-2.1518	8.9522	7.1957	12.5846	-1.8361	5.8428	14.9250	-5.8609	8.2450
4	train_4	0	9.8369	-1.4834	12.8746	6.6375	12.2772	2.4486	5.9405	19.2514	6.2654	7.6784

Exploratory Analysis

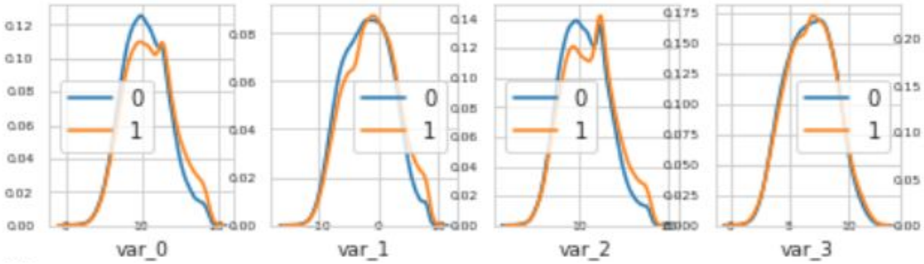
- **Data cleaning:** No missing values, no unreasonable outliers, all types match
- **Descriptive statistics:** Hard to draw any conclusions, variables cover broad distribution of means and deviations

var_0	var_1	var_2	var_3	var_4	var_5	var_6	var_7
200000.000000	200000.000000	200000.000000	200000.000000	200000.000000	200000.000000	200000.000000	200000.
10.679914	-1.627622	10.715192	6.796529	11.078333	-5.065317	5.408949	16.5458
3.040051	4.050044	2.640894	2.043319	1.623150	7.863267	0.866607	3.41807
0.408400	-15.043400	2.117100	-0.040200	5.074800	-32.562600	2.347300	5.34970
8.453850	-4.740025	8.722475	5.254075	9.883175	-11.200350	4.767700	13.9438
10.524750	-1.608050	10.580000	6.825000	11.108250	-4.833150	5.385100	16.4568
12.758200	1.358625	12.516700	8.324100	12.261125	0.924800	6.003000	19.1029
20.315000	10.376800	19.353000	13.188300	16.671400	17.251600	8.447700	27.6918

Using .describe() function on training set

Exploratory Analysis

- Distributions
  - Compare variables using their distribution separated by training target



Created by separating rows based on target and generating subplots for their respective features

- Correlations
  - Look for obvious connections, appears there's nothing so simple in this dataset

39790	var_183	var_189	0.009359
39791	var_189	var_183	0.009359
39792	var_174	var_81	0.009490
39793	var_81	var_174	0.009490
39794	var_81	var_165	0.009714
39795	var_165	var_81	0.009714
39796	var_53	var_148	0.009788
39797	var_148	var_53	0.009788
39798	var_26	var_139	0.009844
39799	var_139	var_26	0.009844

Created by using the .corr() function contained in pandas, then sorting the list of correlations

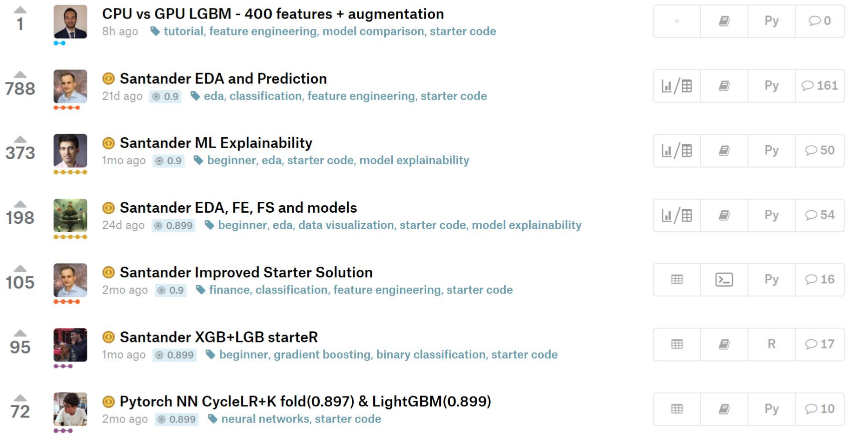
Exploratory Analysis

- Determine appropriate model based on insight gained from EDA
- Key points:

Insight	Reasoning	Conclusion
Variables are highly independent of one another	Low values for all pearson pairwise correlations	Naive Bayes - Model assumes independence of features
Features are of varying importance to target outcome	Variables range from identical to vastly different distributions when stratified based on target	Random Forest Algorithm - Decision tree structure, simpler implementation
Significant amounts of clean data, lacking labels or clear causation	200,000 rows, anonymized data makes manual insight difficult	Light Gradient Boost - Works well with high quantities of data

Researching An Approach

- Our initial, independent attempts performed poorly
  - Massive script runtimes
  - Huge number of parameters to establish
  - Errors in training output
- Move to competition forum to investigate successful strategies
- Find starting points that we can refine and reshape into our own solution



1) Demonstrative Naive Bayes

- Relies on Bayes theorem, conditional probabilities, and approximate distributions
- Uses Naive Bayes classifier, similar to what was done in class/lecture

$$P(A|B) = \frac{P(A \bigcap B)}{P(B)}$$

$$P(B|A) = \frac{P(A \bigcap B)}{P(A)}$$

$$\Rightarrow P(A|B) = \frac{P(B|A) \cdot P(A)}{P(B)}$$

Bayes Theorem

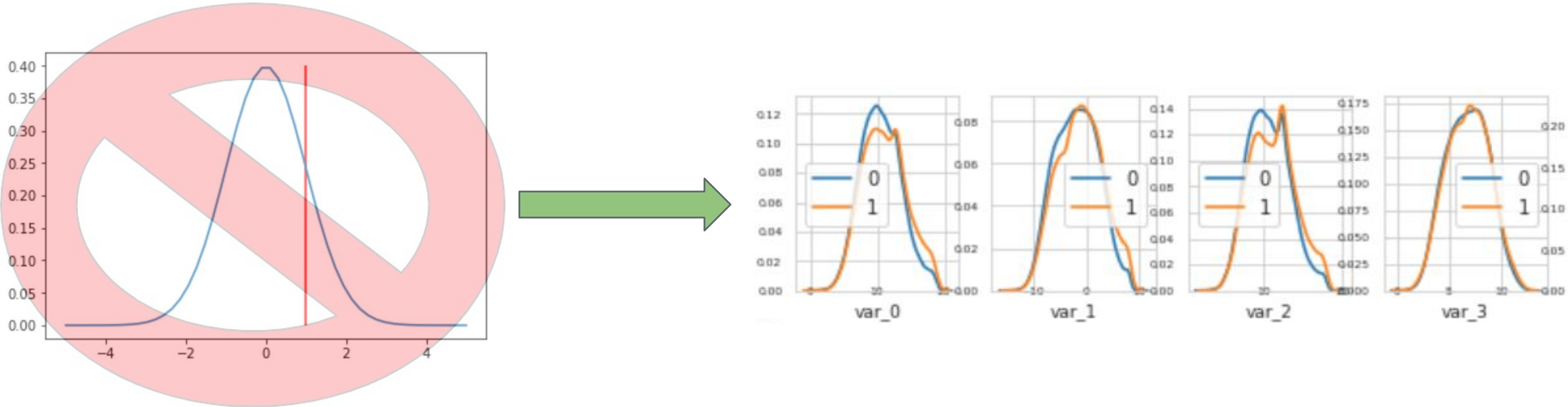
```
In [12]:
# priori probability
ppos = pos_idx.sum() / len(pos_idx)
pneg = neg_idx.sum() / len(neg_idx)

def get_proba(x):
    # we use odds P(target=1|X=x)/P(target=0|X=x)
    return (ppos * norm.pdf(x, loc=stats_df.pos_mean, scale=stats_df.pos_sd).prod()) /\
           (pneg * norm.pdf(x, loc=stats_df.neg_mean, scale=stats_df.neg_sd).prod())
```

```
In [13]:
tr_pred = train.apply(get_proba, axis=1)
```

Bayes probability function in Python

Naive Bayes - Improvements

- Not reliant on set parameters, so less that can be tweaked and modified
- Possible improvement by calculating probabilities using unique distributions



2) Random Forest Algorithm

- Relies on building and merging decision trees using random subsets of features
- Heavy reliance on sklearn and RandomForestClassifier object functions significantly reduces programming struggles

```
train_X, val_X, train_y, val_y = train_test_split(X, y, random_state=1)
rfc_model = RandomForestClassifier(random_state=0).fit(train_X, train_y)
```

```
y_pred_rfc = rfc_model.predict(X_test)
```

Random Forest - Improvements

- High number of parameters that can be modified
- Possible improvement by dropping more features
    - RandomForestClassifier in sklearn provides methods for easily determining feature importance, we could drop the features with lowest performance and eliminate possible overfitting

`feature_importances_`

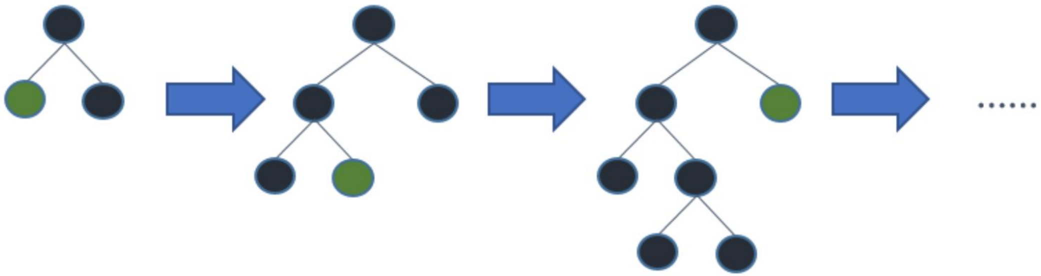
Return the feature importances (the higher, the more important the feature).

Returns:	**feature_importances_ :** ***array, shape = [n_features]***

class `sklearn.ensemble.` **`RandomForestClassifier`** (*n_estimators='warn', criterion='gini', max_depth=None, min_samples_split=2, min_samples_leaf=1, min_weight_fraction_leaf=0.0, max_features='auto', max_leaf_nodes=None, min_impurity_decrease=0.0, min_impurity_split=None, bootstrap=True, oob_score=False, n_jobs=None, random_state=None, verbose=0, warm_start=False, class_weight=None*)

3) Light Gradient Boost

- Uses leaf-wise tree growth for a faster gradient boosting algorithm
- Fast training, low memory usage, compatibility with large datasets



Leaf-wise tree growth

```
prediction = np.zeros(len(X_test))
for fold_n, (train_index, valid_index) in enumerate(folds.split(X,y)):
    print('Fold', fold_n, 'started at', time.ctime())
    X_train, X_valid = X.iloc[train_index], X.iloc[valid_index]
    y_train, y_valid = y.iloc[train_index], y.iloc[valid_index]

    train_data = lgb.Dataset(X_train, label=y_train)
    valid_data = lgb.Dataset(X_valid, label=y_valid)

    model = lgb.train(params,train_data,num_boost_round=20000,
                    valid_sets = [train_data, valid_data],verbose_eval=300,early_stopp
ing_rounds = 200)

    #y_pred_valid = model.predict(X_valid)
    prediction += model.predict(X_test, num_iteration=model.best_iteration)/5
```

LightGBM - Improvements

- High number of parameters to learn, tweak, and improve, however documentation and resources are limited
- Fast learn time means lower opportunity cost for each attempted kernel

- **task** : default value = train ; options = train , prediction ; Specifies the task we wish to perform which is either train or prediction.

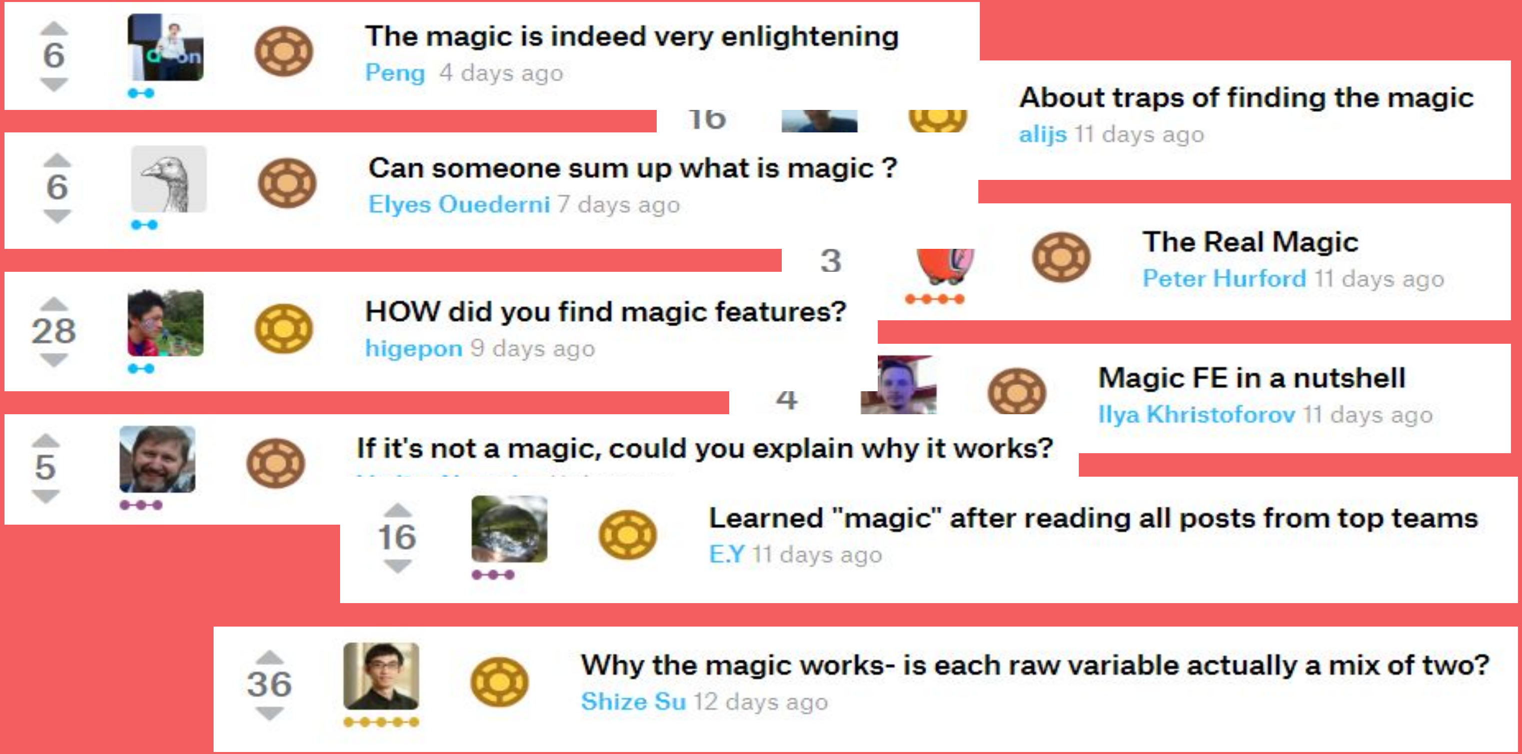
- **application**: default=regression, type=enum, options= options :
  - regression : perform regression task
  - binary : Binary classification
  - multiclass: Multiclass Classification
  - lambdarank : lambdarank application

- **data**: type=string; training data , LightGBM will train from this data
- **num_iterations**: number of boosting iterations to be performed ; default=100; type=int
- **num_leaves** : number of leaves in one tree ; default = 31 ; type =int
- **device** : default= cpu ; options = gpu,cpu. Device on which we want to train our model. Choose GPU for faster training.

- **max_depth**: Specify the max depth to which tree will grow. This parameter is used to deal with overfitting.
- **min_data_in_leaf**: Min number of data in one leaf.
- **feature_fraction**: default=1 ; specifies the fraction of features to be taken for each iteration
- **bagging_fraction**: default=1 ; specifies the fraction of data to be used for each iteration and is generally used to speed up the training and avoid overfitting.
- **min_gain_to_split**: default=.1 ; min gain to perform splitting
- **max_bin** : max number of bins to bucket the feature values.
- **min_data_in_bin** : min number of data in one bin
- **num_threads**: default=OpenMP_default, type=int ;Number of threads for Light GBM.

- **label** : type=string ; specify the label column
- **categorical_feature** : type=string ; specify the categorical features we want to use for training our model
- **num_class**: default=1 ; type=int ; used only for multi-class classification

The "Magic" and 0.901 Barrier



Chris Deotte: The magic is adding 200 new variables where you frequency encode each of the original 200 variables.

Model each variable separately and then combine the 200 models. We will do the same here after adding a “magic” feature to each variable. For each variable, we make a new feature (column) whose value is the number of counts of the corresponding variable. The new feature interacts with the original, so you must set feature_fraction to 1.0. Then you will gain the benefit from the new feature but you also have the detrimental effect of modeling spurious original variable interactions. We will then ensemble the 200 models with logistic regression.

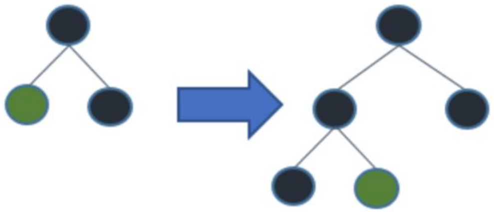
To maximize the gain of the "magic" feature (and climb from LB 0.910 to 0.920), we must allow the new feature to interact with the original variables while preventing the original variables from interacting with each other. Here are 3 ways to do that:

- Use Data Augmentation (as shown in Jiwei's awesome kernel here). You must keep original and new feature in same row.
- Use 200 separate models as shown in this kernel below.
- Merge new feature and original feature into one feature. In original data, simply add 200 to each unique value. (And don't add new columns)

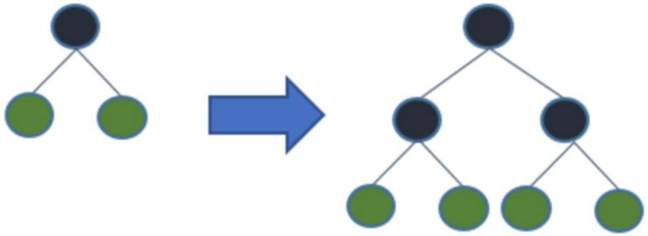
Our New Approach:
LightGBM

Explanation of LightGBM

- Boosting - Ensemble of predictors created sequentially, meaning each subsequent predictor learns from the errors of the previous
- Gradient Boosting - Uses gradient descent to minimize loss function, leverages patterns in residuals
- Light Gradient Boosting - Basic modeling decision trees now grow leaf-wise, allowing algorithm to choose to develop leaf with greatest loss



Leaf-wise Growth



Level-wise Growth

ok.

```
param = {
    #for better accuracy, try dart
    #to deal with over-fitting, try lambda_11, lambda_12
        #and min_gain_to_split for regularization

    #for faster speed
    #to deal with over-fitting
    'bagging_freq': 5,

    #for faster speed
    #to deal with over-fitting
    'bagging_fraction': 0.38,
    'boost_from_average':'false',
    'boost': 'gbdt',
    'feature_fraction': 0.045,

    #for better accuracy, use small learning_rate with large num_iterations
    'learning_rate': 0.0095,

    #limit the tree depth explicitly
    #to deal with over-fitting, try max_depth to avoid growing deep tree
    'max_depth': 7,
    'metric':'auc',

    #this is a very important parameter to prevent overfitting
    #its optimal value depends on the number of training samples and num_leaves
    #setting it to a large value can avoid growing too deep a tree,
    #but may cause under-fitting
    #it practice, setting it to hundreds or thousands is enough for a large dataset
    'min_data_in_leaf': 200,

    #to deal with over-fitting, use min_data_in_leaf and min_sum_hessian_in_leaf
    'min_sum_hessian_in_leaf': 10.0,

    #this is the main parameter to control the complexity of the tree model
    #should be smaller than 2^(max_depth)
    #for better accuracy, use large num_leaves (may cause over-fitting)
    #to deal with over-fitting, use small num_leaves
    'num_leaves': 7, #4 has been used with success

    #chose to add myself
    #can be used to speed up training
    #can be used to deal with over-fitting
    #will randomly selection part of features on each iteration
        #if feature_fraction smaller than 1.0
        #for example, if you set it to 0.8, LightGBM will select
        #80% of features before training each tree
    'feature_fraction': 0.77,

    'num_threads': 8,
    'tree_learner': 'serial',
    'objective': 'binary',
    'verbosity': 1
}
```

```
2551.1s   Fold 3
2556.6s   [LightGBM] [Warning] Starting from the 2.1.2 version, default value for the "boost_from_average" parameter in "binary" objective is true.
          This may cause significantly different results comparing to the previous versions of LightGBM.
          Try to set boost_from_average=false, if your old models produce bad results
2556.6s   [LightGBM] [Info] Number of positive: 18423, number of negative: 164910
          [LightGBM] [Info] Total Bins 51131
2556.6s   [LightGBM] [Info] Number of data: 183333, number of used features: 200
2556.8s   Training until validation scores don't improve for 3500 rounds.
2602.3s   [1000]  training's auc: 0.862346        valid_1's auc: 0.854397
2649.8s   [2000]  training's auc: 0.891907        valid_1's auc: 0.881747
2697.8s   [3000]  training's auc: 0.904741        valid_1's auc: 0.892688
2745.3s   [4000]  training's auc: 0.912543        valid_1's auc: 0.898684
2793.4s   [5000]  training's auc: 0.917693        valid_1's auc: 0.902054
2841.0s   [6000]  training's auc: 0.921587        valid_1's auc: 0.904012
2886.4s   [7000]  training's auc: 0.924774        valid_1's auc: 0.904974
2930.1s   [8000]  training's auc: 0.927702        valid_1's auc: 0.905418
2973.5s   [9000]  training's auc: 0.930494        valid_1's auc: 0.905753
3016.1s   [10000] training's auc: 0.933326        valid_1's auc: 0.90585
3058.3s   [11000] training's auc: 0.936105        valid_1's auc: 0.906009
3100.8s   [12000] training's auc: 0.9388  valid_1's auc: 0.905932
3142.8s   [13000] training's auc: 0.941425        valid_1's auc: 0.905957
3184.1s   [14000] training's auc: 0.944019        valid_1's auc: 0.905745
3226.3s   [15000] training's auc: 0.946597        valid_1's auc: 0.905726
3268.0s   [16000] training's auc: 0.948929        valid_1's auc: 0.905547
3273.8s   Early stopping, best iteration is:
          [12638] training's auc: 0.940477        valid_1's auc: 0.906039
```

CV score: 0.89960

```python
# Build the Light GBM Model
param = {
'bagging_freq': 5,
'bagging_fraction': 0.335,
'boost_from_average':'false',
'boost': 'gbdt',
'feature_fraction': 0.041,
'learning_rate': 0.0083,
'max_depth': -1,
'metric':'auc',
'min_data_in_leaf': 80,
'min_sum_hessian_in_leaf': 10.0,
'num_leaves': 14,
'num_threads': 8,
'tree_learner': 'serial',
'objective': 'binary',
'lambda_l1': 0.50,
'lambda_l2': 0.25,
'verbosity': -1
}
```

```python
num_folds = 11
features = [c for c in train.columns if c not in ['ID_code', 'target']]

folds = KFold(n_splits=num_folds, random_state=2319)
oof = np.zeros(len(train))
getVal = np.zeros(len(train))
predictions = np.zeros(len(target))
feature_importance_df = pd.DataFrame()
```

```
13191.1s  Fold idx:11
13203.3s  Training until validation scores don't improve for 4000 rounds.
13526.4s  [5000]  training's auc: 0.913165       valid_1's auc: 0.897963
13848.1s  [10000] training's auc: 0.923198       valid_1's auc: 0.900723
14162.5s  [15000] training's auc: 0.931137       valid_1's auc: 0.900864
14330.6s  Early stopping, best iteration is:
          [13736] training's auc: 0.929215       valid_1's auc: 0.900958
14475.2s

           >> CV score: 0.90118
```

```python
# Build the Light GBM Model
param = {
'bagging_freq': 5,
'bagging_fraction': 0.335,
'boost_from_average':'false',
'boost': 'gbdt',
'feature_fraction': 0.041,
'learning_rate': 0.0078,
'max_depth': -1,
'metric':'auc',
'min_data_in_leaf': 80,
'min_sum_hessian_in_leaf': 10.0,
'num_leaves': 14,
'num_threads': 8,
'tree_learner': 'serial',
'objective': 'binary',
'lambda_l2': 0.50,
'verbosity': -1
}
```

```
13491.2s   Fold idx:11
13502.8s   Training until validation scores don't improve for 4000 rounds.
13840.9s   [5000]  training's auc: 0.912485        valid_1's auc: 0.897469
14174.1s   [10000] training's auc: 0.922404        valid_1's auc: 0.900732
14489.8s   [15000] training's auc: 0.929973        valid_1's auc: 0.901119
14808.8s   [20000] training's auc: 0.936818        valid_1's auc: 0.900993
14878.2s   Early stopping, best iteration is:
           [17134] training's auc: 0.932955        valid_1's auc: 0.901217
15116.1s
            >> CV score: 0.90107
```

```python
# Build the Light GBM Model
param = {
'bagging_freq': 5,
'bagging_fraction': 0.335,
'boost_from_average':'false',
'boost': 'gbdt',
'feature_fraction': 0.041,
'learning_rate': 0.0083,
'max_depth': -1,
'metric':'auc',
'min_data_in_leaf': 80,
'min_sum_hessian_in_leaf': 10.0,
'num_leaves': 14,
'num_threads': 8,
'tree_learner': 'serial',
'objective': 'binary',
'lambda_l1': 0.50,
'min_gain_to_split': 0.50,
'verbosity': -1
}
```

```
11699.3s   Fold idx:11
11711.7s   Training until validation scores don't improve for 4000 rounds.
12027.7s   [5000]  training's auc: 0.912947        valid_1's auc: 0.898013
12337.0s   [10000] training's auc: 0.923045        valid_1's auc: 0.900899
12634.0s   [15000] training's auc: 0.931013        valid_1's auc: 0.901147
12925.6s   [20000] training's auc: 0.93822 valid_1's auc: 0.901056
12963.4s   Early stopping, best iteration is:
           [16663] training's auc: 0.933477        valid_1's auc: 0.901228
13129.2s
            >> CV score: 0.90123
```

```python
# Build the Light GBM Model
param = {
'bagging_freq': 5,
'bagging_fraction': 0.333,
'boost_from_average':'false',
'boost': 'gbdt',
'feature_fraction': 0.044,
'learning_rate': 0.0077,
'max_depth': -1,
'metric':'auc',
'min_data_in_leaf': 80,
'min_sum_hessian_in_leaf': 10.0,
'num_leaves': 14,
'num_threads': 8,
'tree_learner': 'serial',
'objective': 'binary',
'lambda_l1': 0.51,
'verbosity': -1
}
```

```
14083.4s   Fold idx:11
14096.4s   Training until validation scores don't improve for 4000 rounds.
14420.9s   [5000]  training's auc: 0.912192       valid_1's auc: 0.897213
14742.4s   [10000] training's auc: 0.921971       valid_1's auc: 0.900554
15060.4s   [15000] training's auc: 0.929495       valid_1's auc: 0.901029
15189.4s   Early stopping, best iteration is:
           [13052] training's auc: 0.926664       valid_1's auc: 0.901055
15361.9s
            >> CV score: 0.90138
```

Private Score	Public Score
0.89981	0.90115

Reflection

Difficulties:

- LightGBM documentation is really bad!
- Limited to 3 submissions per day
- Submissions could take 2-4 hours to run on Kaggle cloud

Things we would have liked to do:

- Use neural nets to do ensembling
- “Ensemble methods use multiple learning algorithms to obtain better predictive performance than could be obtained from any of the constituent learning algorithms alone.”

Conclusions

Santander Customer Trans...
15 days ago·Top 21%

1,832nd
of 8802

- Should have stuck with the models with more protection against overfitting!

1	—	**Wizardry**	0.92573	91	15d
2	—	三人寄れば文殊の知恵（本当か？	0.92552	45	15d
3	—	**Rock, Physics, Science!**	0.92467	101	15d
4	▲2	**alijs & Evgeny & ZFTurbo**	0.92460	102	15d
5	▼1	**KKM**	0.92440	107	15d

Sources

- https://medium.com/@pushkarmandot/https-medium-com-pushkarmandot-what-is-lightgbm-how-to-implement-it-how-to-fine-tune-the-parameters-60347819b7fc
- https://www.analyticsvidhya.com/blog/2017/06/which-algorithm-takes-the-crown-light-gbm-vs-xgboost/
- https://towardsdatascience.com/the-random-forest-algorithm-d457d499ffcd
- https://www.kaggle.com/mjbahmani/santander-ml-explainability#6--Model-Development
- https://www.kaggle.com/dromosys/sctp-working-lgb
- https://www.kaggle.com/gpreda/santander-eda-and-prediction/notebook#Feature-engineering
- https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html
- https://lightgbm.readthedocs.io/en/latest/_modules/lightgbm/sklearn.html
- https://www.kaggle.com/cdeotte/200-magical-models-santander-0-920
- https://towardsdatascience.com/l1-and-l2-regularization-methods-ce25e7fc831c