Project Description

In this project I used the Kaggle Customer Churn data to determine whether the customer will churn (leave the company) or not. I split the kaggle training data into train and test (80%/20%) and fitted the models using train data and evaluated model results in test data.

I used mainly the semi automatic learning module `pycaret` for this project. I also used usual boosting modules (`xgboost,lightgbm,catboost`) and regular sklearn models.

In real life the cost of misclassifying leaving customer and not-leaving customer is different. In this project I defined the PROFIT metric as following:

```
profit = +$400  for TP : incentivize the customer to stay, and sign a new contract.
profit = 0      for TN : nothing is lost
profit = -$100  for FP : marketing and effort used to try to retain the user
profit = -$200  for FN : revenue lost from losing a customer

TP = true positive
FP = false positive
FN = false negative
```

After testing various models with extensive feature engineering, I found that the xgboost algorithm gave the best profit.

Data description

	Feature	Type	N	Count	Unique	Missing	MissingPct	Zeros	ZerosPct
2	SeniorCitizen	int64	7043	7043	2	0	0.00	5901	83.79
5	tenure	int64	7043	7043	73	0	0.00	11	0.16
0	customerID	object	7043	7043	7043	0	0.00	0	0.00
1	gender	object	7043	7043	2	0	0.00	0	0.00
3	Partner	object	7043	7043	2	0	0.00	0	0.00
4	Dependents	object	7043	7043	2	0	0.00	0	0.00
6	PhoneService	object	7043	7043	2	0	0.00	0	0.00
7	MultipleLines	object	7043	7043	3	0	0.00	0	0.00
8	InternetService	object	7043	7043	3	0	0.00	0	0.00
9	OnlineSecurity	object	7043	7043	3	0	0.00	0	0.00
10	OnlineBackup	object	7043	7043	3	0	0.00	0	0.00
11	DeviceProtection	object	7043	7043	3	0	0.00	0	0.00
12	TechSupport	object	7043	7043	3	0	0.00	0	0.00
13	StreamingTV	object	7043	7043	3	0	0.00	0	0.00
14	StreamingMovies	object	7043	7043	3	0	0.00	0	0.00
15	Contract	object	7043	7043	3	0	0.00	0	0.00
16	PaperlessBilling	object	7043	7043	2	0	0.00	0	0.00
17	PaymentMethod	object	7043	7043	4	0	0.00	0	0.00
18	MonthlyCharges	float64	7043	7043	1585	0	0.00	0	0.00
19	TotalCharges	object	7043	7043	6531	0	0.00	0	0.00
20	Churn	object	7043	7043	2	0	0.00	0	0.00

Data Processing

- Missing Value imputation for `TotalCharges` with 0.
- Label Encoding for features having 5 or less unique values.
- Binning Numerical Features.
- Combination of features. e.g `SeniorCitizen + Dependents`.
- Boolean Features. e.g. Does someone have Contract or not.
- Aggregation features. eg. Mean of `TotalCharges` per `Contract`.

Sklearn Methods: LogisticRegression and LogisticRegressionCV

- Used raw data with new features from EDA.
- Used SMOTE oversampling since data is imbalanced.
- Used `yeo-johnson` transformers instead of standard scaling since the numerical features were not normal.

- Tuned the model using hyperband library.

```
      Accuracy  Precision Recall    F1-score    AUC
LR    0.4450    0.3075    0.8717    0.4547    0.5812

                                                               Predicted

                    Predicted-noChurn  Predicted-Churn    0    1
Original no-Churn     [[301              734]             TN   FP
Original Churn        [ 48               326]]            FN   TP


Let's make following assumptions
TP = +$400
TN = 0
FP = -$100
FN = -$200

profit = tn*0 + fp*(-100) + fn*(-200) + tp*400
       = 400*tp - 200*fn - 100*fp

tn,fp,fn,tp = confusion_matrix(y_true,y_pred)

             LAST+    2ndrow  1strow
profit = 400*326 - 200*48 - 100*734
       = 47400


============================ LogisticRegressoinCV=====================
        Accuracy  Precision Recall    F1-score    AUC
LRCV    0.7367    0.5024    0.8396    0.6286    0.7695

[[724 311]
 [ 60 314]]

profit = 82,500
```

Boosting: Xgboost, lightgbm and catboost

- Used custom data cleaning.
- Used xgb classifier with custom scoring function from Hyperband.

```
--------------------------------- xgboost -------------------------------
           Accuracy  Precision    Recall    F1-score    AUC
xgboost    0.7097    0.4749       0.8850    0.6181    0.7657

[[669 366]
 [ 43 331]]
```

```
Profit = $87,200
--------------------------------- lightgbm ------------------------------
                 Accuracy  Precision Recall    F1-score  AUC       profit
lgb+hyperband    0.7069    0.4651    0.6952    0.5573    0.7031    $51,300
lgb+hyperopt     0.64088   0.419903  0.925134  0.577629  0.731649  $85,000

I did a lot of hyperparameter tuning of lgb with hyperopt for multiple
days.
I got following results
             5-foldCV   TestProfit
params_lgb1  68,900     83,000
params_lgb2  69,340     82,700
params_lgb3  69,420     87,900 ** This has largest test profit, but less
cv
params_lgb4  69,480     85,000 ** We never see the test data, we only see
train data


So, we must choose the parameters with highest cross validation score.
Here params_lgb4 gives higher profit than params3, this means it is
possible to
get higher score but we need to get it along with higher validation score.
We can try about 10k hyperopt trials but so far I have tried only upto 5k
trials.

Note about hyperopt:
When I dumped the hyperopt trial to a file and load again and used in
hyperopt
then, it gave me the same results in microseconds even if I run further
thousands
of trials. This means using old trials does not work. Always use new
trials but
we can pickle dump it so that we can see the trials history.


--------------------------------- catboost -------------------------------
                   Accuracy     Precision Recall    F1-score    AUC
catboost+optuna    0.6955       0.4618    0.8877    0.6075      0.7569
[[648 387]
 [ 42 332]]

 profit = $85,700
```

Modelling Pycaret

- Used detailed cleaned data.
- Pycaret uses gpu for xgboost and lightgbm in colab.
- Pycaret does not have model interpretation (SHAP) for non-tree based models.
- Simple model comparison gave naive bayes as the best model.
- Used additional metrics `MCC and LogLoss`.

- Used `tune-sklearn` algorithm to tune logistic regression.
- The model calibration in pycaret DID NOT improve the metric.

	Model	Accuracy	AUC	Recall	Prec.	F1	Kappa	MCC	LogLoss	TT (Sec)
nb	Naive Bayes	0.7069	0.8243	0.8345	0.4722	0.6027	0.3983	0.4389	10.1234	0.0160
qda	Quadratic Discriminant Analysis	0.6102	0.6244	0.6548	0.3697	0.4717	0.2001	0.2210	13.4642	0.0440
lda	Linear Discriminant Analysis	0.8107	0.8455	0.5493	0.6780	0.6067	0.4839	0.4887	6.5368	0.0280
gbc	Gradient Boosting Classifier	0.8123	0.8496	0.5418	0.6861	0.6053	0.4843	0.4903	6.4831	0.6760
ada	Ada Boost Classifier	0.8107	0.8498	0.5402	0.6840	0.6028	0.4808	0.4872	6.5368	0.2220
lr	Logistic Regression	0.8063	0.8507	0.5376	0.6684	0.5957	0.4703	0.4753	6.6900	1.3160
lightgbm	Light Gradient Boosting Machine	0.7959	0.8345	0.5326	0.6401	0.5808	0.4475	0.4512	7.0502	0.9700
catboost	CatBoost Classifier	0.8028	0.8437	0.5318	0.6612	0.5888	0.4611	0.4663	6.8127	8.8400
xgboost	Extreme Gradient Boosting	0.7854	0.8250	0.5301	0.6106	0.5672	0.4256	0.4276	7.4104	1.1940
rf	Random Forest Classifier	0.7972	0.8258	0.5109	0.6512	0.5723	0.4419	0.4477	7.0041	0.2640
et	Extra Trees Classifier	0.7779	0.8017	0.5076	0.5968	0.5483	0.4024	0.4048	7.6709	0.3120
dt	Decision Tree Classifier	0.7362	0.6644	0.5059	0.5054	0.5050	0.3254	0.3258	9.1116	0.0620
ridge	Ridge Classifier	0.8087	0.0000	0.5042	0.6940	0.5837	0.4635	0.4737	6.6057	0.0360
knn	K Neighbors Classifier	0.7752	0.7729	0.4624	0.5993	0.5216	0.3781	0.3837	7.7630	0.0780
svm	SVM - Linear Kernel	0.7307	0.0000	0.3277	0.5969	0.3813	0.2308	0.2614	9.3025	0.0700

	Model	Description	Accuracy	AUC	Recall	Precision	F1	Kappa	MCC	LogLoss
0	nb	default	0.706000	0.824100	0.833600	0.471200	0.601600	0.396700	0.437300	10.154100
1	lr	tuned,tune-sklearn,n_iter=100	0.752600	0.848300	0.805200	0.523400	0.634100	0.460100	0.484500	8.544700
2	lr	tuned,scikit-optimize	0.754200	0.849700	0.799400	0.525500	0.633800	0.460800	0.483800	8.491000
3	lr	tuned,custom_grid	0.805900	0.848900	0.536800	0.669000	0.594800	0.469200	0.474700	6.705400
4	lr	default	0.805600	0.849100	0.531800	0.669900	0.592200	0.466900	0.472700	6.713100
5	lightgbm	default	0.792800	0.835600	0.527700	0.631300	0.574200	0.438900	0.442400	7.157400
6	xgboost	default	0.788600	0.826500	0.518500	0.621700	0.565100	0.427000	0.430300	7.303000

```
Pycaret Logistic Regression
==============================================================
            Accuracy Precision Recall    F1-score    AUC
pycaret_lr    0.7509 0.5199     0.8021    0.6309      0.7673

[[758 277]
 [ 74 300]]

profit = 400*300 - 200*74 - 100*277
       = 77,500

Pycaret Naive Bayes
==============================================================
            Accuracy  Precision Recall    F1-score    AUC
pycaret_nb  0.7296    0.4943    0.8102    0.6140      0.7553
[[725 310]
 [ 71 303]]
```

```
profit = 400*303 - 200*71 - 100*310
       = 76,000

Pycaret Xgboost (Takes long time, more than 1 hr)
===============================================================
                  Accuracy Precision Recall  F1-score  AUC
pycaret_xgboost    0.7601  0.5342    0.7513  0.6244    0.7573

[[790 245]
 [ 93 281]]

profit = 400*281 - 200*93 - 100*245
       = 69,300

 Pycaret LDA (Takes medium time, 5 minutes)
================================================================
- Used polynomial features and fix imbalanced data.

               Accuracy  Precision    Recall    F1-score  AUC
pycaret_lda    0.7062    0.4704       0.8503    0.6057    0.7522


[[677 358]
 [ 56 318]]

profit = 400*318 - 200*56 - 100*358
       = 80,200
```

EvalML method

- Minimal data processing (dropped gender and make some features numeric)
- evalml itself deals with missing values and categorical features.

```
          Accuracy  Precision Recall    F1-score  AUC
evalml    0.7977    0.6369    0.5535    0.5923    0.7197

[[917 118]
 [167 207]]

profit = 400*207 - 200*167 - 100*118
       = 37,600
```

Deep Learning models

- Used minimal data processing.
- Dropped `customerID` and `gender`.

- Imputed `TotalCharges` with 0.
- Created dummy variables from categorical features.
- Used standard scaling to scale the data.
- Used `class_weight` parameter to deal with imbalanced data.
- Tuned keras model with scikitlearn `GridSearchCV`

```
Model parameters
{'activation': 'sigmoid',
 'batch_size': 128,
 'epochs': 30,
 'n_feats': 43,
 'units': (45, 30, 15)}

NOTE: The result changes each time even if I set SEED for everything.

      Accuracy  Precision Recall    F1-score    AUC
keras 0.6849    0.4422    0.7166    0.5469    0.6950
[[697 338]
 [106 268]]

profit = 400*268 - 200*106 - 100*338
       = 52,200
```

Model Comparison

```
This is a imbalanced binary classification.
The useful metrics are F2-score and Recall.
AUC is useful only when dataset is balanced.
F1 is useful when precision and recall is equally important.
Here I defined a custom metric "profit" based on confusion matrix
elements.

- Logistic regression cv algorithm gave me the best profit.
- I used custom feature engineering of the data.
- SMOTE oversampling gave worse result than no resampling.
  (note: I have used class_weight='balanced')
- Elasticnet penalty gave worse result than l2 penalty.
- Make custom loss scorer instead of default scoring such as
f1,roc_auc,recall.

Profit = 400*TP  - 200*FN - 100*FP

TP = +$400 ==> incentivize the customer to stay, and sign a new contract.
TN = 0
FP = -$100 ==> marketing and effort used to try to retain the user
FN = -$200 ==> revenue lost from losing a customer

Some Notes about comparing models:
```

```
- We should never directly compare test dataset, we may simply overfit the
test
  data. It's like training test data and overfitting by best hyperparams.
- We should compare validation splits and validation splits must have very
small
  standard deviation, then, after we get hyperparams from
training/validation,
  we use these hyperparams to see how it does in test.
  We can not change hyperparameter based on test results, but we can
change
  based on validation results.
- Here I have reported the test profit, but for model comparison we can
report
  cross-validation profit.

                 Accuracy   Precision Recall       F1-score  AUC
Profit
---------------------------------------------------------------------------
-----
xgboost          0.7097     0.4749    0.8850       0.6181    0.7657
$87,200
catboost+optuna  0.6955     0.4618    0.8877       0.6075    0.7569
$85,700
lgb+hyperopt     0.64088   0.419903   0.925134     0.577629  0.731649
$85,000
LRCV             0.7367     0.5024    0.8396       0.6286    0.7695
$82,500
pycaret_lda      0.7062     0.4704    0.8503       0.6057    0.752200
$80,200
pycaret_lr       0.750887   0.519931  0.802139     0.630915  0.767253
$77,500
pycaret_nb       0.729595   0.494290  0.810160     0.613982  0.755322
$76,000
pycaret_xgboost  0.760114   0.534221  0.751337     0.624444  0.757311
$69,300
keras            0.684883   0.442244  0.716578     0.546939  0.695004
$52,200
lgb+hyperband    0.7069     0.4651    0.6952       0.5573    0.7031
$51,300
LR               0.444996   0.307547  0.871658     0.454672  0.581240
$47,400
evalml           0.7977     0.6369    0.5535       0.5923    0.719700
$37,600
lgb+optuna       0.7473     0.5262    0.4840       0.5042    0.6632
$17,500
```