In [1]:

```
from autogluon.tabular import TabularDataset, TabularPredictor

```

In [10]:

```
data = TabularDataset('数据集/card_transdata.csv')
subsample_size = 500  # subsample subset of data for faster demo, try setting this to much larg
train_data = data.iloc[:subsample_size,:]#.sample(n=subsample_size, random_state=0)
train_data.head()
```

```
Loaded data from: 数据集/card_transdata.csv | Columns = 8 / 8 | Rows = 1000000 -> 10
00000
```

Out[10]:

	distance_from_home	distance_from_last_transaction	ratio_to_median_purchase_price	repeat_
0	57.877857	0.311140	1.945940	
1	10.829943	0.175592	1.294219	
2	5.091079	0.805153	0.427715	
3	2.247564	5.600044	0.362663	
4	44.190936	0.566486	2.222767	

In [11]:

```
label = 'fraud'
print("Summary of class variable: \n", train_data[label].describe())
```

```
Summary of class variable:
 count    500.000000
mean       0.082000
std        0.274639
min        0.000000
25%        0.000000
50%        0.000000
75%        0.000000
max        1.000000
Name: fraud, dtype: float64
```

In [12]:

```
save_path = 'agModels-predictClass'  # specifies folder to store trained models
predictor = TabularPredictor(label=label, path=save_path).fit(train_data)
```

```
Warning: path already exists! This predictor may overwrite an existing predictor!
path="agModels-predictClass"
Beginning AutoGluon training ...
AutoGluon will save models to "agModels-predictClass\"
AutoGluon Version:  0.4.0
Python Version:     3.9.7
Operating System:   Windows
Train Data Rows:    500
Train Data Columns: 7
Label Column: fraud
Preprocessing data ...
AutoGluon infers your prediction problem is: 'binary' (because only two unique la
bel-values observed).
        2 unique label values:  [0.0, 1.0]
        If 'binary' is not the correct problem_type, please manually specify the
problem_type parameter during predictor init (You may specify problem_type as one
of: ['binary', 'multiclass', 'regression'])
Selected class <--> label mapping:  class 1 = 1, class 0 = 0
Using Feature Generators to preprocess the data ...
Fitting AutoMLPipelineFeatureGenerator...
        Available Memory:                    2503.34 MB
        Train Data (Original)  Memory Usage: 0.03 MB (0.0% of available memory)
        Inferring data type of each feature based on column values. Set feature_m
etadata_in to manually specify special dtypes of the features.
        Stage 1 Generators:
                Fitting AsTypeFeatureGenerator...
                        Note: Converting 4 features to boolean dtype as they only
contain 2 unique values.
        Stage 2 Generators:
                Fitting FillNaFeatureGenerator...
        Stage 3 Generators:
                Fitting IdentityFeatureGenerator...
        Stage 4 Generators:
                Fitting DropUniqueFeatureGenerator...
        Types of features in original data (raw dtype, special dtypes):
                ('float', []) : 7 | ['distance_from_home', 'distance_from_last_tr
ansaction', 'ratio_to_median_purchase_price', 'repeat_retailer', 'used_chip',
...]
        Types of features in processed data (raw dtype, special dtypes):
                ('float', [])     : 3 | ['distance_from_home', 'distance_from_las
t_transaction', 'ratio_to_median_purchase_price']
                ('int', ['bool']) : 4 | ['repeat_retailer', 'used_chip', 'used_pi
n_number', 'online_order']
        0.0s = Fit runtime
        7 features in original data used to generate 7 features in processed dat
a.
        Train Data (Processed) Memory Usage: 0.01 MB (0.0% of available memory)
Data preprocessing and feature engineering runtime = 0.05s ...
AutoGluon will gauge predictive performance using evaluation metric: 'accuracy'
        To change this, specify the eval_metric parameter of Predictor()
Automatically generating train/validation split with holdout_frac=0.2, Train Row
s: 400, Val Rows: 100
Fitting 13 L1 models ...
Fitting model: KNeighborsUnif ...
        0.93     = Validation score   (accuracy)
```

```
        0.01s    = Training   runtime
        0.01s    = Validation runtime
Fitting model: KNeighborsDist ...
        0.93     = Validation score   (accuracy)
        0.0s     = Training   runtime
        0.01s    = Validation runtime
Fitting model: LightGBMXT ...
        0.99     = Validation score   (accuracy)
        0.21s    = Training   runtime
        0.0s     = Validation runtime
Fitting model: LightGBM ...
        1.0      = Validation score   (accuracy)
        0.3s     = Training   runtime
        0.0s     = Validation runtime
Fitting model: RandomForestGini ...
        0.97     = Validation score   (accuracy)
        0.51s    = Training   runtime
        0.04s    = Validation runtime
Fitting model: RandomForestEntr ...
        0.97     = Validation score   (accuracy)
        0.33s    = Training   runtime
        0.04s    = Validation runtime
Fitting model: CatBoost ...
        0.98     = Validation score   (accuracy)
        0.38s    = Training   runtime
        0.0s     = Validation runtime
Fitting model: ExtraTreesGini ...
        0.96     = Validation score   (accuracy)
        0.32s    = Training   runtime
        0.04s    = Validation runtime
Fitting model: ExtraTreesEntr ...
        0.96     = Validation score   (accuracy)
        0.42s    = Training   runtime
        0.04s    = Validation runtime
Fitting model: NeuralNetFastAI ...
        0.97     = Validation score   (accuracy)
        0.48s    = Training   runtime
        0.01s    = Validation runtime
Fitting model: XGBoost ...
        0.97     = Validation score   (accuracy)
        0.19s    = Training   runtime
        0.01s    = Validation runtime
Fitting model: NeuralNetTorch ...
        0.99     = Validation score   (accuracy)
        2.92s    = Training   runtime
        0.0s     = Validation runtime
Fitting model: LightGBMLarge ...
        0.99     = Validation score   (accuracy)
        0.45s    = Training   runtime
        0.0s     = Validation runtime
Fitting model: WeightedEnsemble_L2 ...
        1.0      = Validation score   (accuracy)
        0.27s    = Training   runtime
        0.0s     = Validation runtime
AutoGluon training complete, total runtime = 7.6s ... Best model: "WeightedEnsemb
le_L2"
TabularPredictor saved. To load, use: predictor = TabularPredictor.load("agModels
-predictClass\")
```

In [13]:

```
test_data = data.iloc[subsample_size:800,:]#.sample(n=subsample_size, random_state=0)
print(test_data)
y_test = test_data[label]  # values to predict
test_data_nolab = test_data.drop(columns=[label])  # delete label column to prove we're not che
test_data_nolab.head()
```

```
     distance_from_home  distance_from_last_transaction  \
500            1.750684                        0.580885
501            1.869959                        2.514885
502            1.753528                        3.749890
503          203.614980                        4.801743
504            6.702776                        1.505302
..                  ...                             ...
795            7.200425                        1.901058
796            8.446890                        0.538515
797           10.673741                      417.453868
798           14.504727                        1.862790
799            1.069143                        0.124392

     ratio_to_median_purchase_price  repeat_retailer  used_chip  \
500                        0.247285              0.0        0.0
501                        1.394792              0.0        0.0
502                        1.143069              0.0        0.0
503                        0.481828              1.0        0.0
504                        0.699937              1.0        1.0
..                              ...              ...        ...
795                        0.638770              1.0        1.0
796                        0.442399              1.0        0.0
797                        0.475115              1.0        0.0
798                        0.334947              1.0        1.0
799                        0.787226              0.0        1.0

     used_pin_number  online_order  fraud
500              0.0           0.0    0.0
501              0.0           1.0    0.0
502              0.0           1.0    0.0
503              0.0           1.0    1.0
504              0.0           1.0    0.0
..               ...           ...    ...
795              0.0           1.0    0.0
796              0.0           1.0    0.0
797              0.0           1.0    1.0
798              0.0           0.0    0.0
799              0.0           1.0    0.0

[300 rows x 8 columns]
```

Out[13]:

	distance_from_home	distance_from_last_transaction	ratio_to_median_purchase_price	re
500	1.750684	0.580885	0.247285	
501	1.869959	2.514885	1.394792	
502	1.753528	3.749890	1.143069	
503	203.614980	4.801743	0.481828	

	distance_from_home	distance_from_last_transaction	ratio_to_median_purchase_price	re
504	6.702776	1.505302	0.699937	

In [14]:

```
predictor = TabularPredictor.load(save_path)  # unnecessary, just demonstrates how to load prev

y_pred = predictor.predict(test_data_nolab)
print("Predictions:  \n", y_pred)
perf = predictor.evaluate_predictions(y_true=y_test, y_pred=y_pred, auxiliary_metrics=True)
```

```
Evaluation: accuracy on test data: 0.9933333333333333
Evaluations on test data:
{
    "accuracy": 0.9933333333333333,
    "balanced_accuracy": 0.9615384615384616,
    "mcc": 0.9572815468117103,
    "f1": 0.9600000000000001,
    "precision": 1.0,
    "recall": 0.9230769230769231
}

Predictions:
 500    0.0
501    0.0
502    0.0
503    1.0
504    0.0
      ...
795    0.0
796    0.0
797    0.0
798    0.0
799    0.0
Name: fraud, Length: 300, dtype: float64
```

In [15]:

```
predictor.leaderboard(test_data, silent=True)
```

Out[15]:

	model	score_test	score_val	pred_time_test	pred_time_val	fit_time	pred_
0	LightGBM	0.993333	1.00	0.003988	0.004984	0.304152	
1	WeightedEnsemble_L2	0.993333	1.00	0.005984	0.004984	0.576394	
2	ExtraTreesGini	0.993333	0.96	0.052410	0.039888	0.321642	
3	RandomForestGini	0.993333	0.97	0.073796	0.041136	0.506712	
4	LightGBMLarge	0.990000	0.99	0.014959	0.004987	0.448746	
5	ExtraTreesEntr	0.990000	0.96	0.057999	0.039890	0.417498	
6	RandomForestEntr	0.990000	0.97	0.093741	0.035901	0.331518	
7	NeuralNetFastAI	0.983333	0.97	0.026925	0.009973	0.475679	
8	NeuralNetTorch	0.980000	0.99	0.017951	0.004987	2.923846	
9	LightGBMXT	0.976667	0.99	0.007978	0.003988	0.210413	
10	XGBoost	0.973333	0.97	0.004986	0.009972	0.192462	
11	CatBoost	0.966667	0.98	0.003986	0.000998	0.376950	
12	KNeighborsUnif	0.933333	0.93	0.005985	0.010589	0.006981	
13	KNeighborsDist	0.926667	0.93	0.006629	0.009967	0.004995	

In [ ]:

```

```