American Express Default Predictions

By: The Quagga Group
Ethan Silvas, Naomy Velasco, Karim Bouzina, and Jeff Crabill

Mission Statement

Develop a machine learning model that predicts credit defaults using real-world data from American Express to better manage risk in a consumer lending business.

Competition Description

- "Predict the probability that a customer does not pay back their credit card balance amount in the future based on their monthly customer profile"
- Credit default binary classification
- Industry-size data
- Potentially influence AMEX's model

Evaluation

The evaluation metric, M, for this competition is the mean of two measures of rank ordering: Normalized Gini Coefficient, G, and default rate captured at 4%, D.

$$M = 0.5 \cdot (G + D)$$

- Calculates with test set predictions
- Max possible score of 1.0
- Top leaderboard score of 0.80977

Project Goals

- Impute and encode our own dataset
- Use new models like XGBoost and LGBM
- Test techniques like dropout and regularization for neural networks
- Score as close to 0.80 as possible

Outline

Collect

customer_ID	S_2	P_2	D_39	B_1
458913 unique values	28Feb17 30Mar18	-0.46 1.01	0 5.39	-7.59 1.32
0000099d6bd597052cdcda90ffabf56573fe9d7c79be5fbac11a8ed792feb62a	2017-03-09	0.9384687191272548	0.0017333390041739	0.0087244509498605
0000099d6bd597052cdcda90ffabf56573fe9d7c79be5fbac11a8ed792feb62a	2017-04-07	0.9366646050988444	0.0057754430691282	0.0049233526310337
0000099d6bd597052cdcda90ffabf56573fe9d7c79be5fbac11a8ed792feb62a	2017-05-28	0.9541802771951844	0.0915053967544593	0.0216546637740555
0000099d6bd597052cdcda90ffabf56573fe9d7c79be5fbac11a8ed792feb62a	2017-06-13	0.9603835924391524	0.0024552237550715	0.0136833350682097
0000099d6bd597052cdcda90ffabf56573fe9d7c79be5fbac11a8ed792feb62a	2017-07-16	0.9472483832103192	0.0024830136942964	0.0151934639447788

Baseline

	precision	recall	f1-score	support
0	0.92	0.94	0.93	340085
1	0.82	0.78	0.80	118828
accuracy			0.90	458913
macro avg	0.87	0.86	0.86	458913
weighted avg	0.90	0.90	0.90	458913

Tune

Private Score	Public Score
0.79361	**0.78462**

Data Collection

test_data.csv (33.82 GB)

# D_46	# D_47	# D_48	# D_49	# B_6
-17.3 – 16.3	-0.03 – 1.64	-0.01 – 8.97	0 – 45.8	-0.01 – 1.21k
0.3585865793715965	0.525351040810055	0.255736073902975		0.0639022133803909
0.35362955018564	0.5213112572080865	0.223328868696034		0.0652610579665619
0.3346501402648452	0.5245677277623807	0.1894239790446447		0.066981923963344з
0.3232707585815574	0.5309292044162731	0.1355861611744148		0.0837202553007994
0.2310086756150568	0.5293047211041928			0.0758999199008079
0.2759629064725732	0.5297616727419061			0.0957843776472023

Data Collection: Features

- D_* = Delinquency variables
- S_* = Spend variables
- P_* = Payment variables
- B_* = Balance variables
- R_* = Risk variables

with the following features being categorical:

```
['B_30', 'B_38', 'D_114', 'D_116', 'D_117', 'D_120', 'D_126', 'D_63', 'D_64', 'D_66', 'D_68']
```

Data Collection: Aggregate

	B_10_last	B_10_max	B_10_mean	B_10_min	B_10_std	B_11_last
customer_ID						
0000099d6bd597052cdcda90ffabf56573fe9d7c79be5fbac11a8ed792feb62a	0.326172	0.741699	0.270264	0.096191	0.181835	0.010262
00000fd6641609c6ece5454664794f0340ad84dddce9a267a310b5ae68e9d8e5	0.297119	0.302734	0.298828	0.293945	0.003044	0.014572
00001b22f846c82c51f6e3958ccd81970162bae8b007e80662ef27519fcc18c1	0.296387	0.302734	0.273682	0.162109	0.052867	0.005093
000041bdba6ecadd89a52d11886e8eaaec9325906c9723355abb5ca523658edc	0.411621	0.431885	0.306641	0.192993	0.079525	0.005489
00007889e4fcd2614b6cbe7f8f3d2e5c728eca32d9eb8ad51ca8b8c4a24cefed	0.125244	0.260742	0.100342	0.044739	0.074579	0.001000

Data Collection: Methods

- Original vs. aggregate
  - 232 vs. 927 features
- One-hot encode
- Impute NaN values (already normalized)
  - Numerical = mean
  - Categorial = most common
- TensorFlow pipeline
  - tf.Input()
  - float32

Data Collection: Code Samples

```python
# turn each column of the dataframe into a tf.keras.Input() object
inputs = {}
for name, column in X_train.items():
  if (name in binary_feature_names):
    dtype = tf.int64
  else:
    dtype = tf.float32

  inputs[name] = tf.keras.Input(shape=(), name=name, dtype=dtype)
```

```python
numeric_inputs = {}
for name in numeric_feature_names:
  numeric_inputs[name] = inputs[name]

# preprocess numeric inputs by stacking them and converting to float32
numeric_inputs = stack_dict(numeric_inputs)
preprocessed.append(numeric_inputs)

preprocessed
```

[9]:

```
[<KerasTensor: shape=(None, 1) dtype=float32 (created by layer 'tf.cast')>,
 <KerasTensor: shape=(None, 1) dtype=float32 (created by layer 'tf.cast_1')>,
 <KerasTensor: shape=(None, 1) dtype=float32 (created by layer 'tf.cast_2')>,
 <KerasTensor: shape=(None, 1) dtype=float32 (created by layer 'tf.cast_3')>,
 <KerasTensor: shape=(None, 1) dtype=float32 (created by layer 'tf.cast_4')>,
 <KerasTensor: shape=(None, 1) dtype=float32 (created by layer 'tf.cast_5')>,
 <KerasTensor: shape=(None, 1) dtype=float32 (created by layer 'tf.cast_6')>,
 <KerasTensor: shape=(None, 1) dtype=float32 (created by layer 'tf.cast_7')>,
 <KerasTensor: shape=(None, 1) dtype=float32 (created by layer 'tf.cast_8')>,
 <KerasTensor: shape=(None, 1) dtype=float32 (created by layer 'tf.cast_9')>,
 <KerasTensor: shape=(None, 1) dtype=float32 (created by layer 'tf.cast_10')>,
 <KerasTensor: shape=(None, 916) dtype=float32 (created by layer 'tf.stack')>]
```

Baseline Models

- Scikit-learn models
  - LogisticRegression
  - DecisionTreeClassifier
  - RandomForestClassifier
  - SGD Classifier
  - KNN Classifier
- Shallow neural network (1 hidden layer)
- LGBM
- XGBoost

Baseline Decision Tree Classifier

```
M = 1.0
```

```
print(classification_report(y, preds))
```

```
              precision    recall  f1-score   support

           0       1.00      1.00      1.00   4153582
           1       1.00      1.00      1.00   1377869

    accuracy                           1.00   5531451
   macro avg       1.00      1.00      1.00   5531451
weighted avg       1.00      1.00      1.00   5531451
```

→ **0.34586**

Baseline Random Forest Classifier

M = 0.6340978403530687

```
print(classification_report(y, preds))
```

	precision	recall	f1-score	support
0	0.89	0.92	0.91	4153582
1	0.74	0.65	0.69	1377869
accuracy			0.86	5531451
macro avg	0.81	0.79	0.80	5531451
weighted avg	0.85	0.86	0.85	5531451

0.71774

Baseline Logistic Regression

```
M = 0.7747117851405895
```

```
print(classification_report(y, preds))
```

	precision	recall	f1-score	support
0	0.92	0.94	0.93	340085
1	0.82	0.78	0.80	118828
accuracy			0.90	458913
macro avg	0.87	0.86	0.86	458913
weighted avg	0.90	0.90	0.90	458913

→ **0.77498**

Baseline KNN Classifier

```
M = 0.7040079562040782
```

```
print(classification_report(y, knn_preds))
```

	precision	recall	f1-score	support
0	0.92	0.97	0.95	340085
1	0.89	0.77	0.83	118828
accuracy			0.92	458913
macro avg	0.91	0.87	0.89	458913
weighted avg	0.92	0.92	0.91	458913

0.57919

Baseline SGDClassifier

M = 0.7404786983804272

```
print(classification_report(y, preds))
```

	precision	recall	f1-score	support
0	0.90	0.95	0.93	340085
1	0.84	0.70	0.76	118828
accuracy			0.89	458913
macro avg	0.87	0.83	0.84	458913
weighted avg	0.88	0.89	0.88	458913

0.73017

Baseline Shallow Neural Network

M = 0.7930589925674986

```
print(classification_report(target, (preds > 0.5).astype("int32")))
```

	precision	recall	f1-score	support
0	0.94	0.93	0.93	340085
1	0.81	0.82	0.82	118828
accuracy			0.90	458913
macro avg	0.87	0.88	0.88	458913
weighted avg	0.90	0.90	0.90	458913

0.78462

Baseline LGBM Classifier

```
M = 0.588492095944125
```

```
print(classification_report(y, preds))
```

	precision	recall	f1-score	support
0	0.94	0.94	0.94	340085
1	0.82	0.82	0.82	118828
accuracy			0.91	458913
macro avg	0.88	0.88	0.88	458913
weighted avg	0.91	0.91	0.91	458913

→ **0.78137**

Baseline XGB Classifier

M = 0.8517121892133738

```
print(classification_report(y, preds))
```

	precision	recall	f1-score	support
0	0.95	0.95	0.95	340085
1	0.85	0.85	0.85	118828
accuracy			0.92	458913
macro avg	0.90	0.90	0.90	458913
weighted avg	0.92	0.92	0.92	458913

→ **0.77002**

Baseline XGB Regressor

M = 0.8392985836076163

```
print(classification_report(y, predict))
```

	precision	recall	f1-score	support
0	0.94	0.95	0.94	340085
1	0.85	0.83	0.84	118828
accuracy			0.92	458913
macro avg	0.90	0.89	0.89	458913
weighted avg	0.92	0.92	0.92	458913

0.76512

Baseline Models: Takeaways

- Aggregate data was always better
- 90% accuracy “cap”
- Easy to overfit
- Try to beat 0.78462
- How to improve by 0.01-0.02?

Tuned Models: Approach

- Add validation set
- Tune individual model hyperparameters
  - max_iter, max_depth, etc.
- Neural networks
  - Activation functions
  - Number of layers and nodes
  - Optimizer
  - L1 or L2 Regularization
  - Dropout

Tuned Logistic Regression

- Validation size = .25
- max_iter = 290
- C = 100

M = 0.7844621833516792

	precision	recall	f1-score	support
0	0.93	0.94	0.93	85180
1	0.82	0.79	0.80	29549
accuracy			0.90	114729
macro avg	0.87	0.86	0.87	114729
weighted avg	0.90	0.90	0.90	114729

0.77498

0.77868

Tuned Random Forest Classifier

- Validation size = .25
- max_depth = 20

```
M = 0.761545259608168
```

	precision	recall	f1-score	support
0	0.93	0.93	0.93	67899
1	0.80	0.79	0.80	23884
accuracy			0.89	91783
macro avg	0.86	0.86	0.86	91783
weighted avg	0.89	0.89	0.89	91783

0.71774

0.75896

Tuned XGB Classifier

M = 0.8525676894753856

```
xgb_classifier = xgb.XGBClassifier(n_estimators = 200, max_depth = 5, subsample = 0.75)
```

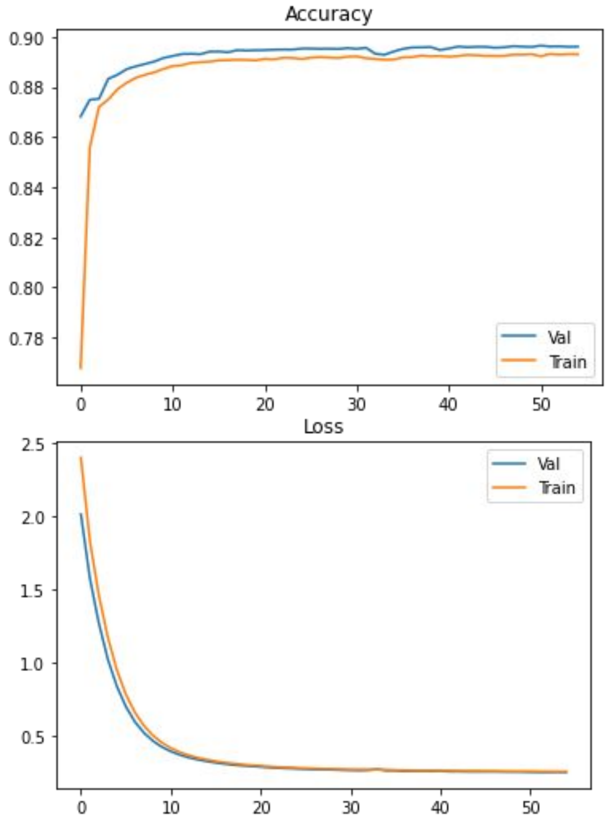
0.77002

0.76803

	precision	recall	f1-score	support
0	0.95	0.95	0.95	340085
1	0.85	0.85	0.85	118828
accuracy			0.92	458913
macro avg	0.90	0.90	0.90	458913
weighted avg	0.92	0.92	0.92	458913

Tuned Shallow Neural Network

- Validation size = .20
- 1 hidden layer, 116 nodes
- Dropout = 0.1
- L2 regularization = 0.01
- 55 epochs
- 50k batch size

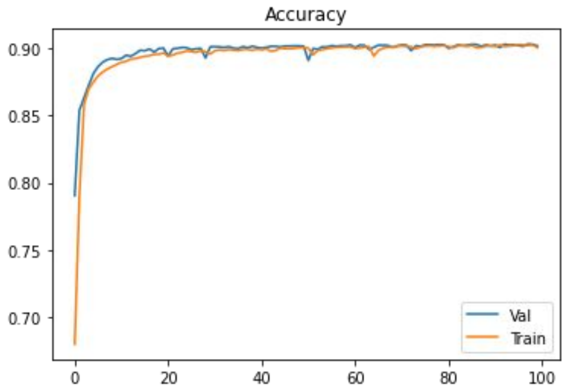


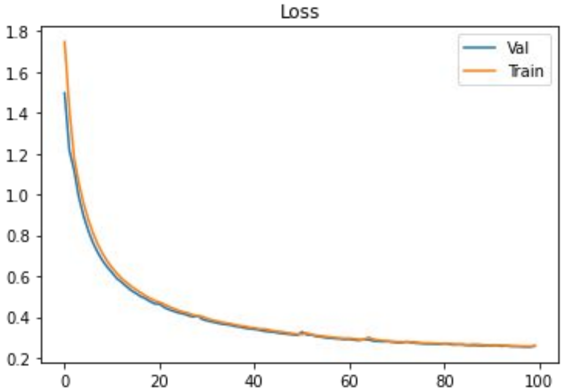
0.78462

0.76586

Tuned Deep Neural Network

```
# hidden layers
nn.add(Dense(hidden_nodes_l1, input_dim=num_features, activation='relu', kernel_regularizer=l2(0.00
1)))
nn.add(Dropout(0.25))
nn.add(Dense(hidden_nodes_l2, activation='tanh', kernel_regularizer=l2(0.001)))
nn.add(Dropout(0.25))
nn.add(Dense(hidden_nodes_l3, activation='relu', kernel_regularizer=l2(0.001)))
nn.add(Dropout(0.25))
nn.add(Dense(hidden_nodes_l4, activation='tanh', kernel_regularizer=l2(0.001)))
```





0.78462

0.78412

Tuned Logistic Regression + StratifiedKFold

- Validation size = .25
- max_iter = 290
- C = 100

M score: 0.7789007070595015

	precision	recall	f1-score	support
0	0.93	0.94	0.93	340085
1	0.82	0.78	0.80	118828
accuracy			0.90	458913
macro avg	0.87	0.86	0.87	458913
weighted avg	0.90	0.90	0.90	458913

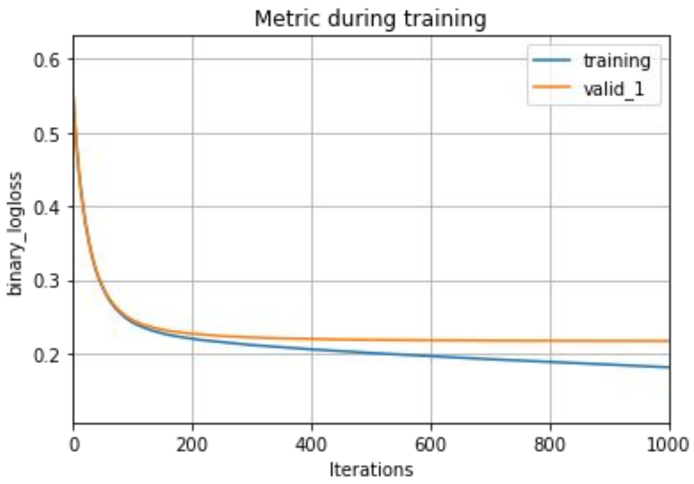
0.77868

0.7789

Tuned LGBM + StratifiedKFold

```
model = lgb.LGBMClassifier(
    objective='binary',
    n_estimators=1000,
    num_leaves=50,
    learning_rate=0.03,
    colsample_bytree=0.1,
    min_child_samples=2000,
    max_bins=500,
    reg_alpha=2,
    random_state=25
)
```

```
M score: 0.7920069386936385
```



0.78137

0.79185

Results and Conclusions

- Top model and score
  - LGBM + SKFold
  - 0.79185
- Balancing classes with SKFold improved performance
- More tunings/complexity != better performance
- Importance of data collection, imputing, encoding
- Look out for really good training performance

Project Difficulties

- Memory constraints
- Long training times
- Dealing with NaNs
- Trying to improve by 0.01-0.02
- 90% accuracy “cap” and overfitting

Next Steps

- Keep tuning hyperparameters
- Try models that can handle NaN values
- Label encoding instead of one-hot
- Try more variations of Dropout/Regularization
- AWS models
- Auto generate hyperparameters (ex: RandomizedSearchCV)

Thank you!