American Express - Default Prediction
MIDS - W207
Spring 2023

Jim Zhu

Rick Chen

Julian Rippert

Berkeley
UNIVERSITY OF CALIFORNIA

Agenda

1. Problem Motivation
2. Dataset Description
3. EDA
4. Models
   a. Clustering
   b. Random Forest
   c. Transformer
5. Results
6. Takeaways

Problem Motivation

- Goal of a credit issuer is to ensure credit is repaid back with interest to compensate risk
- Default events are unfavorable for a lender

Dataset Description

- Dataset dimensions
  - (5531451,191)
- Time Series Data
- Default is when a customer does not pay due amount in 120 days after their latest statement date.
- *Features are anonymized and normalized*

Feature Categories:

D_* = Delinquency variables

S_* = Spend variables

P_* = Payment variables

B_* = Balance variables

R_* = Risk variables

EDA

- Validated the data was standardized
- Categoricals Columns
  - Reindexed to 0
  - One-Hot Encoded
  - Dropped due to interpretability

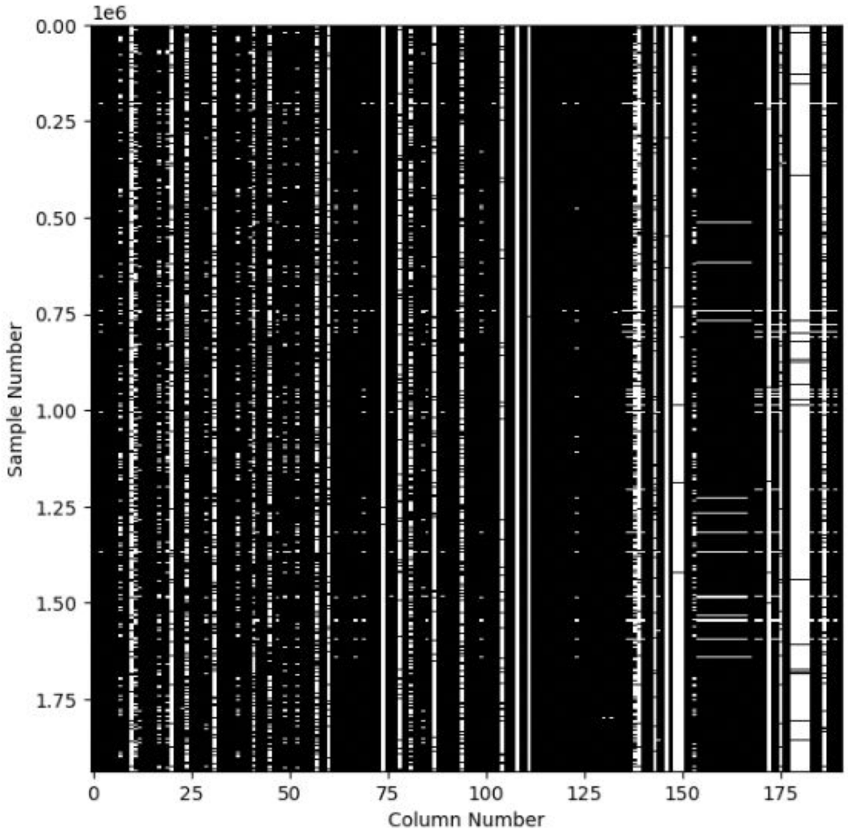
	count	mean	std	min	25%	50%	75%	max
D_42	283,952.000000	0.000000	0.000000	-0.000372	0.040314	0.123779	0.258057	4.187500
D_49	171,995.000000	0.000000	0.000000	0.000007	0.062439	0.131104	0.245972	19.953125
D_53	477,308.000000	0.000000	0.000000	0.000000	0.005875	0.011810	0.039185	7.902344
R_4	1,936,007.000000	0.000000	0.000000	0.000000	0.002539	0.005081	0.007626	1.009766
R_5	1,936,007.000000	0.000000	0.000000	0.000000	0.002550	0.005096	0.007645	17.515625

```
2.0      726993
3.0      442452
1.0      390498
5.0      147417
4.0       94695
7.0       81702
6.0       51298
Name: B_38, dtype: int64
```

```
CO      1428972
CR       336769
CL       154494
XZ         9405
XM         3672
XL         2695
Name: D_63, dtype: int64
```

```
O       1022626
U        514145
R        294146
          69495
-1        35595
Name: D_64, dtype: int64
```

EDA - NaNs



- Set max 40% threshold of NaNs per feature
  - Length before filtering out high NaN count cols: 191
  - Number of columns to drop: 62
- Imputed -1 for all NaNs

Train/Val/Test Split

```
Training Features dimensions:  (3318870, 190)
Training Labels dimensions:  (3318870,)
Validation Features dimensions:  (1106290, 190)
Validation Labels dimensions:  (1106290,)
Test Features dimensions:  (1106291, 190)
Test Labels dimensions:  (1106291,)

Training Percent of Positive Labels:  0.2440

Validation Percent of Positive Labels:  0.2553
```

- Implemented a 60/20/20 split
- To maintain integrity of the time series no shuffling was used
- Yet, similar target label distribution

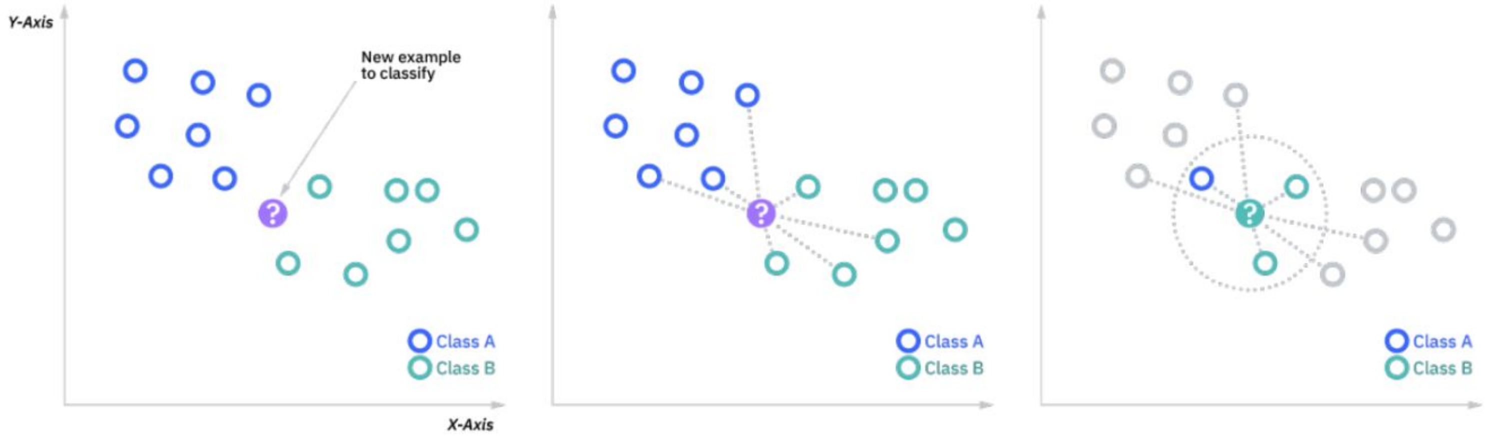
Models & Experiments

- Baseline Model - always predict no default event
    - Accuracy Score = 75.60%

Model - KNN

KNN models are relatively simple models great for both classification and regression problems

- Known as a lazy learner, and makes a great baseline model
- Calculates the distance between similar data points, and makes predictions based on the frequency of different classes



KNN Baseline Model

- Build a baseline KNN model with hyperparameters
  - n_neighbors = 5, p = 2, leaf_size = 30
- Resulted in an initial accuracy of 61% on the training data

	precision	recall	f1-score	support
0.0	0.74	0.74	0.74	82208
1.0	0.24	0.24	0.24	28421
accuracy			0.61	110629
macro avg	0.49	0.49	0.49	110629
weighted avg	0.61	0.61	0.61	110629

KNN Hyperparameter Tuning

- Use Grid Search CV to tune the hyperparameters
- Find the hyperparameters for the highest accuracy model

```
# Build a dictionary for the hyperparameters we are looking to tune
# ignore p value and leaf_size as gridsearchcv takes too long, so just tune n_neighbors
# p = [1, 2]
#leaf_size = list(range(1, 50))
n_neighbors = list(range(1, 50))
hyperparameters = dict(n_neighbors = n_neighbors)
```

```
# initiate another KNN model class
knn_2 = KNeighborsClassifier()

#run gridsearchcv on the hyperparameters for a KNN model with 10 folds
clf = GridSearchCV(knn_2, hyperparameters, cv = 10, n_jobs = -1)

# fit the model
best_model = clf.fit(X_train_pca, Y_train)
```

```
print('Best leafsize:' , best_model.best_estimator_.get_params()['leaf_size'])
print('Best n_neighbors:' , best_model.best_estimator_.get_params()['n_neighbors'])
#print('Best p:' , best_model.best_estimator_.get_params()['p'])
```

```
Best leafsize: 30
Best n_neighbors: 27
```

KNN Model Results

- Using the tuned hyperparameters, a KNN model fit on training data results in an accuracy on the test data of 67%
- Precision, recall, and F1-scores are 74%, 84%, and 79% respectively

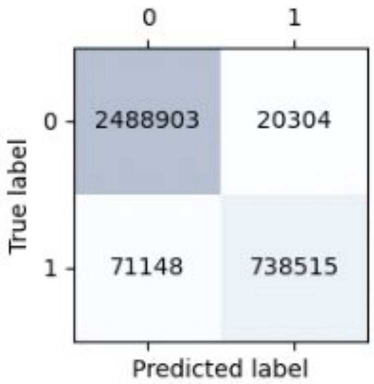
	precision	recall	f1-score	support
0.0	0.74	0.84	0.79	82208
1.0	0.27	0.17	0.21	28421
accuracy			0.67	110629
macro avg	0.51	0.50	0.50	110629
weighted avg	0.62	0.67	0.64	110629

Model - Random Forest

Building a random forest

```
%%time
random_forest = RandomForestClassifier(criterion='entropy', #function to measure the quality of a feature split
                                       n_estimators=10, #number of trees in the forest
                                       bootstrap = True, #different sample for each estimator
                                       max_samples = 0.6, #max number of samples is 50% of training data
                                       n_jobs=2) # The number of jobs to run in parallel.
clf = random_forest.fit(X_train, Y_train)
```

```
Accuracy on Training Data: 0.9724
Accuracy on Validation Data: 0.8727
Validation Precision score: 0.8360
Validation Recall score: 0.8239
Validation F1 score 0.8296
```



Random Forest - Hyperparameter Tuning

- Establish random forest parameter distribution
  - SKlearn’s RandomizedSearchCV will randomly sample from these

```python
criterion = ['entropy'] #what will be used to choose features at node splits
n_estimators = [i for i in range(10,18,2)] # number of trees in the random forest
bootstrap = [True] # drawing samples from our source data with replacement
max_samples = [i/10 for i in range(2,10,2)] #percentage of training samples to train each tree with
n_jobs=[-1]
max_features = ['sqrt']  #could add in 'log2' but took too long
max_depth = [i for i in range(110,160,10)] #max amount of layers in a tree
max_depth.append(None)  #if None, then nodes are expanded until all leaves are pure
min_samples_split = [2, 6, 10] # minimum sample number to split a node, need at least two to split
min_samples_leaf = [1,2, 3] # A split point at any depth will only be considered if it leaves at least
                             #min_samples_leaf training samples in each of the left and right branches.
                              #This may have the effect of smoothing the model, especially in regression

random_grid = { 'criterion' : criterion,
               'n_estimators' : n_estimators,
               'bootstrap' : bootstrap,
               'max_samples' : max_samples,
               'n_jobs' : n_jobs,
               'max_features' : max_features,
               'max_depth' : max_depth,
               'min_samples_split': min_samples_split,
               'min_samples_leaf' : min_samples_leaf
              }
```

Random Forest - Hyperparameter Tuning

- Initiate the tuning

```
%%time
#first we need to initiate a new base tree estimator
base_tree =  RandomForestClassifier()

#initialize the random search CV hyperparameter tuner
Random_Forest_HyperTuned = RandomizedSearchCV(estimator = base_tree,
                                              param_distributions = random_grid,#params defines above
                                              n_iter = 8, #Number of parameter settings that are sampled from distr
                                              cv = 3, #default is5-fold cross validation, so we will have 30 trees
                                              verbose = 1, #the higher, the more messages
                                              return_train_score = True # attribute will include training scores
                                             )
#now we need to train the model using our training data and labels
Random_Forest_HyperTuned.fit(X_train,Y_train)
```

```
Fitting 3 folds for each of 8 candidates, totalling 24 fits
CPU times: user 21min 23s, sys: 2min 17s, total: 23min 40s
Wall time: 12h 50min 54s
```

```
▸ RandomizedSearchCV
▸ estimator: RandomForestClassifier
    ▸ RandomForestClassifier
```

Best parameters

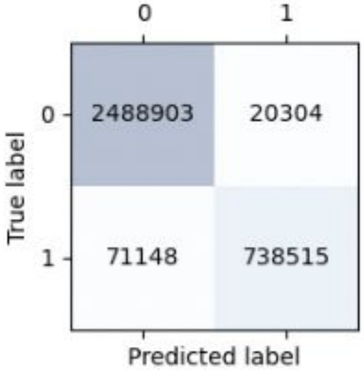
```
{'n_jobs': -1,
 'n_estimators': 16,
 'min_samples_split': 2,
 'min_samples_leaf': 3,
 'max_samples': 0.8,
 'max_features': 'sqrt',
 'max_depth': 150,
 'criterion': 'entropy',
 'bootstrap': True}
```

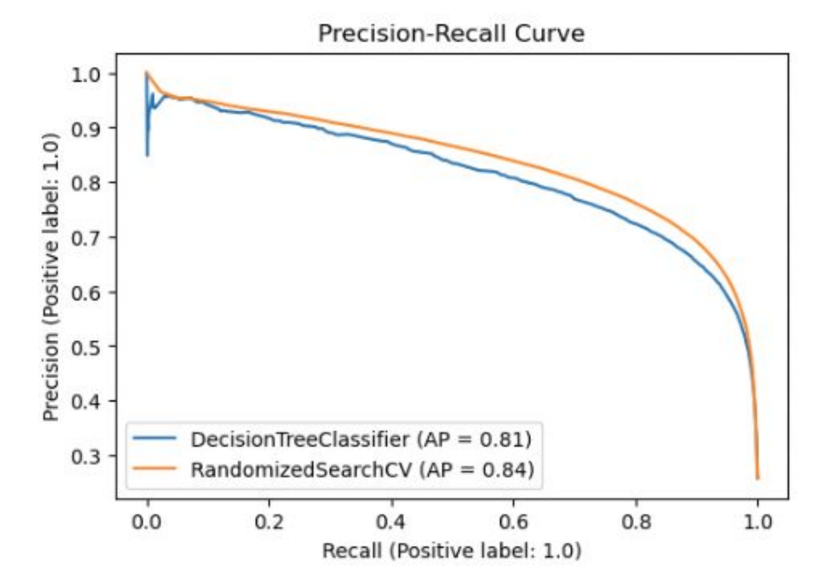
Feature Importances

	Feature	importance
0	P_2	0.076980
1	D_44	0.038740
2	D_48	0.032910
3	D_45	0.029680
4	D_61	0.027010
5	B_20	0.025950
6	B_3	0.021830
7	B_2	0.020580
8	B_10	0.019860
9	B_18	0.018480

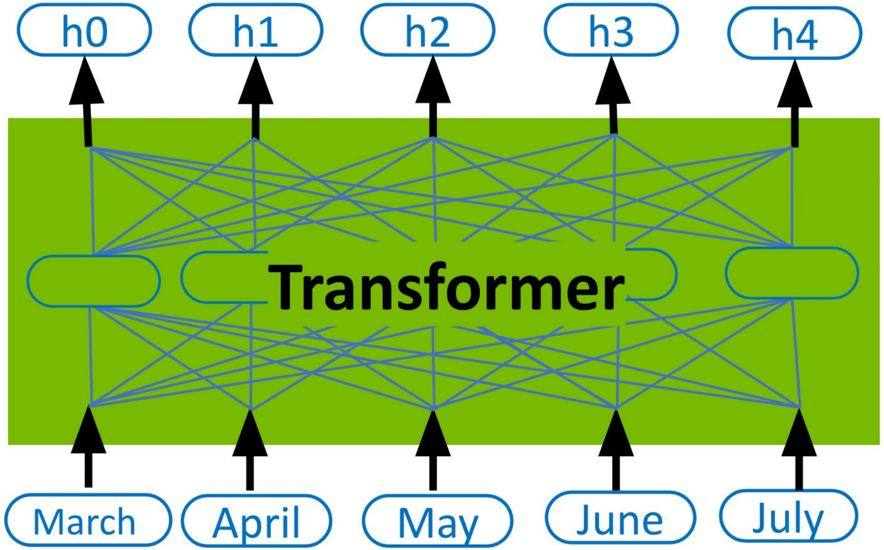
Random Forest - Test Set Evaluation

```
Accuracy on Test Data: 0.8829
Test Precision score: 0.8422
Test Recall score: 0.8641
Test F1 score 0.8520
```





Model Transformer

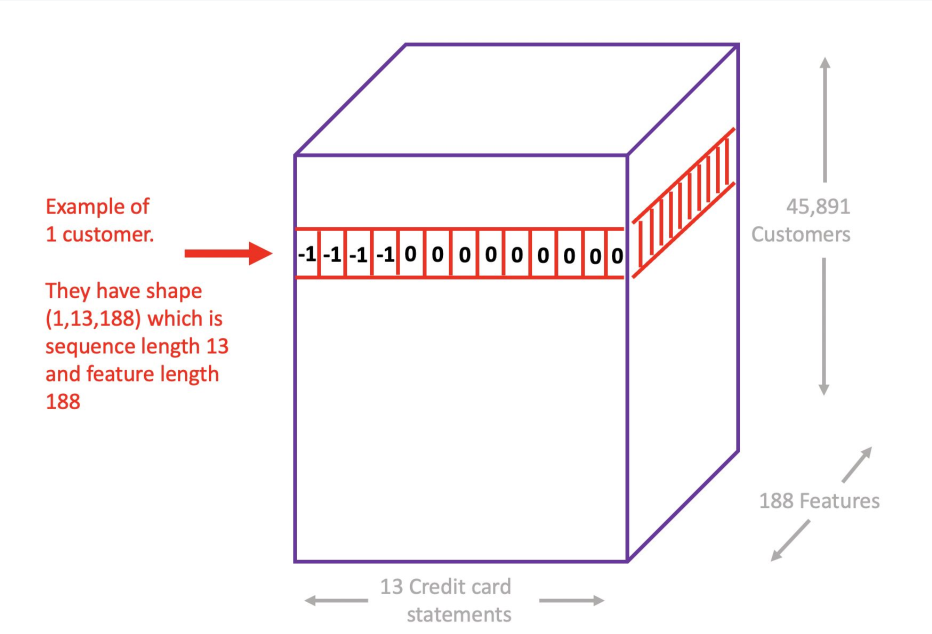


Time series data:

- Most of customers have 13 months credit history
- Transform data from 2d to 3d
- Implement transformer encoder and connect its output to a classification head using binary cross-entropy

Model Transformer - Pre-Processing and Feature Engineering

- Load and split training data:
  - 10 files
  - 45891 customers
  - 13 credit card statements
- Add padding
  - For customers has less than 13 credit card history, pad with -1
- Feature Engineer
  - Output shape (45891, 13, 188)



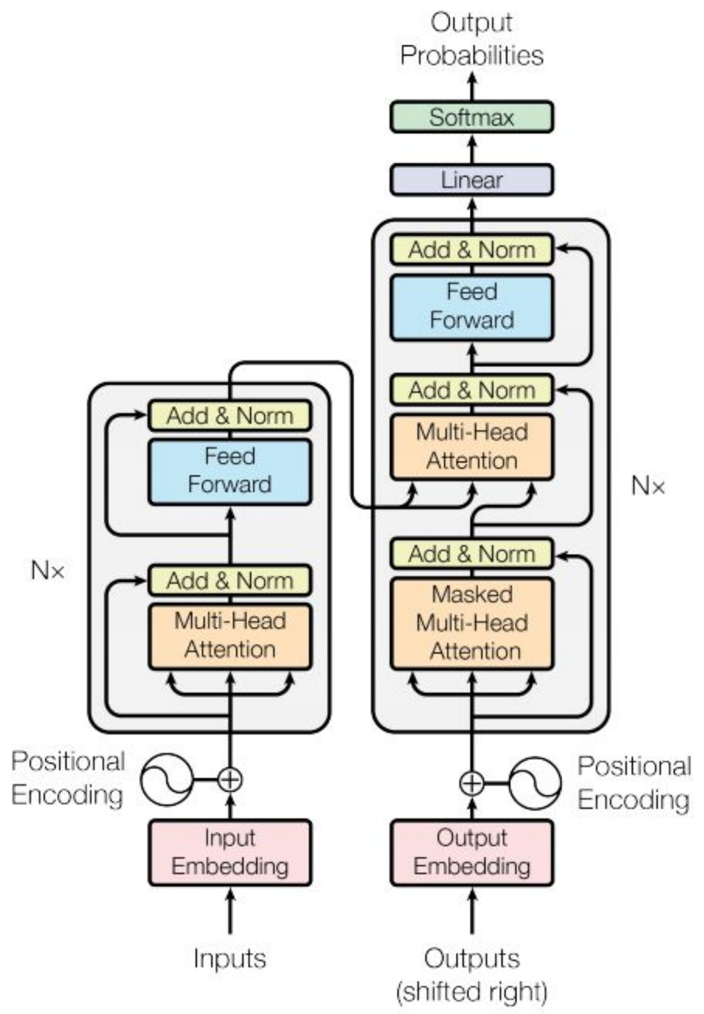
Transformer Block

▾ Hyper-parameters

```
feat_dim = 188
embed_dim = 64   # Embedding size for attention
num_heads = 4   # Number of attention heads
ff_dim = 128   # Hidden layer size in feed forward network inside transformer
dropout_rate = 0.3
num_blocks = 2
```

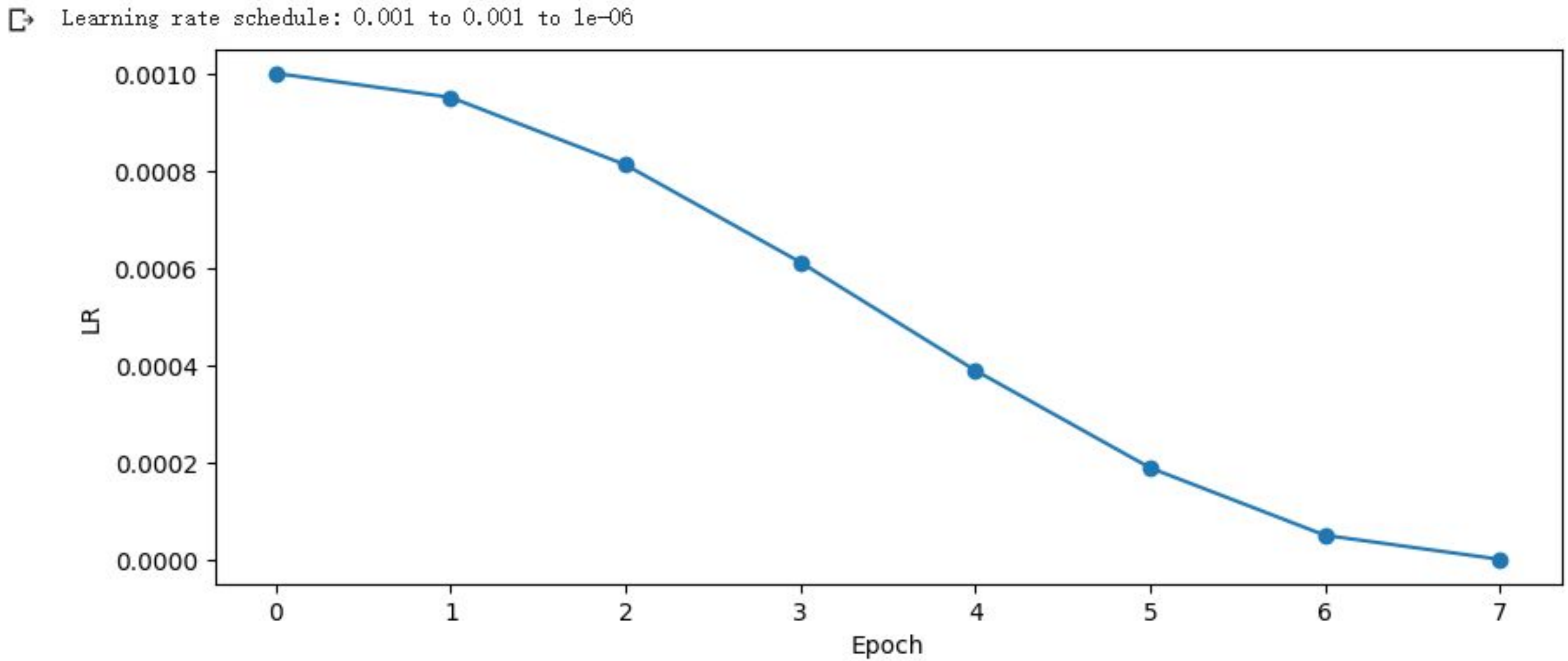
```
[ ]  class TransformerBlock(layers.Layer):
         def __init__(self, embed_dim, feat_dim, num_heads, ff_dim, rate=0.1):
             super(TransformerBlock, self).__init__()
             self.att = layers.MultiHeadAttention(num_heads=num_heads, key_dim=embed_dim)
             self.ffn = keras.Sequential(
                 [layers.Dense(ff_dim, activation="gelu"), layers.Dense(feat_dim),]
             )
             self.layernorm1 = layers.LayerNormalization(epsilon=1e-6)
             self.layernorm2 = layers.LayerNormalization(epsilon=1e-6)
             self.dropout1 = layers.Dropout(rate)
             self.dropout2 = layers.Dropout(rate)

         def call(self, inputs, training):
             attn_output = self.att(inputs, inputs)
             attn_output = self.dropout1(attn_output, training=training)
             out1 = self.layernorm1(inputs + attn_output)
             ffn_output = self.ffn(out1)
             ffn_output = self.dropout2(ffn_output, training=training)
             return self.layernorm2(out1 + ffn_output)
```



Transformer (cont.)

- Learning schedule



Transformer - Training

- We train 5 folds of model using cross validation
  - Use 8 files to trian, 2 files for validation
- 8 epochs for each fold
- Save 5 output transformer model into files

```
#########################
### Fold 5 with valid files [9, 10]
### Training data shapes (367128, 13, 188) (367128,)
### Validation data shapes (91785, 13, 188) (91785,)
#########################

Epoch 1: LearningRateScheduler setting learning rate to 0.001.
Epoch 1/8
718/718 - 24s - loss: 0.2424 - val_loss: 0.2307 - lr: 0.0010 - 24s/epoch - 34ms/step

Epoch 2: LearningRateScheduler setting learning rate to 0.0009505339495172585.
Epoch 2/8
718/718 - 19s - loss: 0.2312 - val_loss: 0.2370 - lr: 9.5053e-04 - 19s/epoch - 27ms/step

Epoch 3: LearningRateScheduler setting learning rate to 0.0008119331560284375.
Epoch 3/8
718/718 - 19s - loss: 0.2279 - val_loss: 0.2269 - lr: 8.1193e-04 - 19s/epoch - 27ms/step
```

Model Transformer - Performance

- Kaggle Evaluation Metrics
  - The metics M is defined as the the mean of two measures
  - G: Normalized Gini Coefficient
  - D: Default rate captured at 4%

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

Cross Validation Score

```
Fold 1 CV= 0.7843216315379575
```

```
Fold 2 CV= 0.7821227060205045
```

```
Fold 3 CV= 0.7852668054694807
```

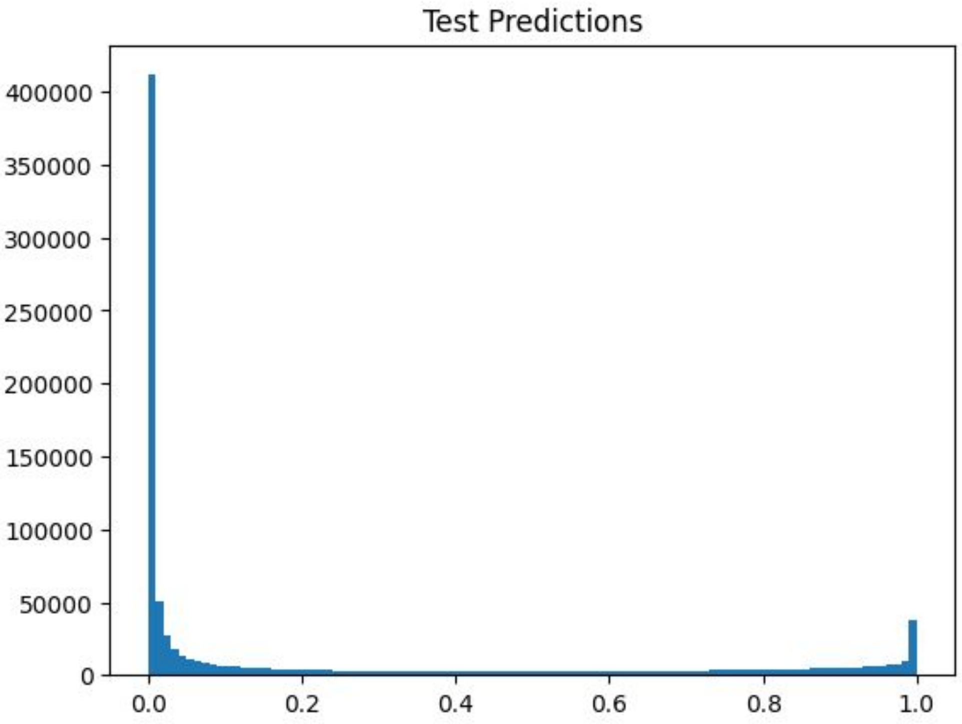
```
Fold 4 CV= 0.7878325830028957
```

```
Fold 5 CV= 0.7903108125684797

#########################
Overall CV = 0.7859054438364497
```

Model Transformer - Test Inference

- Process test data the same way as train data, the output will be 20 files with each file having shape (46231, 13, 188) as (customer x statement x feature)
- Ensemble 5 models from training and create predictions by averaging each model outputs
- Graph the predictions



Model Transformer - Kaggle submission

- Private score: 0.798, Public Score: 0.789

Submission and Description	Private Score	Public Score	Selected
submission.csv Complete (after deadline) · 3d ago · Transformer model	**0.79831**	**0.78911**	☐

Prize Winners

#	△	Team	Members	Score	Entries	Last	Solution
1	▲ 3	lucky shake		0.80977	91	8mo	
2	▲ 5	byDefault [JuneHomes]		0.80938	340	8mo	
3	▲ 2	AIBANK		0.80925	50	8mo	
4	▲ 6	GRE		0.80900	186	8mo	
5	▼ 3	VISA		0.80881	387	8mo	

Takeaways

- **No PII data in features**
  - No insight into categorical features
- **Models may be learning abstract biases about certain demographics**
- **Last 13 bank statements may introduce recency bias**

Contributions:

-EDA (**All**)

-Data Pre Processing (**All**)

-KNN (**Jim**)

-Random Forest (**Julian**)

-Transformer (**Rick**)