Introduction:

The goal of the project is to solve the classification problem by forecasting if the loan will default or not. We are using last 1 year of Fannie-mae Loan performance data. Performance data is to monitor the 24 months performance of loan acquisition. If a loan defaults in any given month, it is categorized as default.
My work is mostly regarding designing neural network model and optimization the model based various layer configuration, transfer function, dropouts and feature selections.

Description of my work:

My contribution to the project is described as below

1. Model Configuration: Develop code for various model configuration.
2. Test and Validation: Enhance the framework to test and validate results.
3. Model Adjustments: Develop logic for output adjustment for SoftMax and sigmoid based model
4. Confusion matrix: Create confusion matrix to fine tune models

Model Configuration: We have tried below model configuration to test the performance of loan dataset.

- 2 layer model (2 Relu Transfer function)
    - Model: Builder(
    - (layers): ModuleList(
    - (0): Linear(in_features=101000, out_features=1000, bias=True)
    - (1): ReLU()
    - (2): Linear(in_features=1000, out_features=100, bias=True)
    - (3): ReLU()
    -)
    -)

- 2 layer model (1 Relu and 1 Sigmoid transfer function)
    - Model: Builder(
    - (layers): ModuleList(
    - (0): Linear(in_features=101000, out_features=1000, bias=True)
    - (1): ReLU()
    - (2): Linear(in_features=1000, out_features=100, bias=True)
    - (3): Sigmoid()
    -)
    -)
- 3 layer model (3 Relu transfer function)
    - Model: Builder(
    - (layers): ModuleList(
    - (0): Linear(in_features=101000, out_features=1000, bias=True)
    - (1): ReLU()
    - (2): Linear(in_features=1000, out_features=1000, bias=True)
    - (3): ReLU()
    - (4): Linear(in_features=1000, out_features=100, bias=True)
    - (5): ReLU()

    -)
    -)
- 2 layer model (1 Relu and 1 Softmax transfer function)
    - Model: Builder(
    - (layers): ModuleList(
    - (0): Linear(in_features=101000, out_features=1000, bias=True)
    - (1): ReLU()
    - (2): Linear(in_features=1000, out_features=100, bias=True)
    - (3): Sigmoid()
    -)
    -)
- 2 layer model with 0.2 dropout
    - Model: Builder(
    - (layers): ModuleList(
    - (0): Linear(in_features=101000, out_features=1000, bias=True)
    - (1): ReLU()
    - (2): Linear(in_features=1000, out_features=100, bias=True)
    - (3): ReLU()
    -)
    -)

Test and Validation:

The application is reading the data in configurable chunk of 100 using data loader. The data is then pass thru network and results output of chunk size 100. The test and validation code checks for true positive, true negative, false positive and false negative. It also creates accuracy percentage on correct/total of processed data.

Model Adjustment:

The sigmoid and SoftMax model produces the probabilistic output for loan default. Develop an classification logic for converting the probabilistic output to loan default values. Further provide the overriding feature in models to provide this value to improve the accuracy of these models.

Confusion matrix:

Used the test and validation results to create the confusion matrix for each model results . Below are the results on the sample dataset.

```
Model:  Builder(
  (layers): ModuleList(
    (0): Linear(in_features=101000, out_features=1000, bias=True)
    (1): ReLU()
    (2): Linear(in_features=1000, out_features=100, bias=True)
    (3): ReLU()
  )
)
```

Predicted False True __all__
Actual
False   55992   506   56498
True   3268   34   3302
__all__   59260   540   59800

** TEST: 2.0_layer | # Layers: 2.0 | Optimizer: Adam **

Model: Builder(
  (layers): ModuleList(
    (0): Linear(in_features=101000, out_features=1000, bias=True)
    (1): ReLU()
    (2): Linear(in_features=1000, out_features=100, bias=True)
    (3): Sigmoid()
)
)
2 Layer SIGMOID Model
Predicted   0.0 1.0 __all__
Actual
0.0   57872   0   57872
1.0   1928   0   1928
__all__   59800   0   59800

** TEST: 3.0_layer | # Layers: 3.0 | Optimizer: Adam **

Model: Builder(
  (layers): ModuleList(
    (0): Linear(in_features=101000, out_features=1000, bias=True)
    (1): ReLU()
    (2): Linear(in_features=1000, out_features=1000, bias=True)
    (3): ReLU()
    (4): Linear(in_features=1000, out_features=100, bias=True)
    (5): ReLU()
)
)
3 Layer RELu Model
Predicted False True __all__
Actual
False   55794   50   55844
True   3956   0   3956
__all__   59750   50   59800

** TEST: 2.0_layer | # Layers: 2.0 | Optimizer: Adam **

```
Model:  Builder(
  (layers): ModuleList(
    (0): Linear(in_features=101000, out_features=1000, bias=True)
    (1): ReLU()
    (2): Linear(in_features=1000, out_features=100, bias=True)
    (3): Softmax()
  )
)
2 Layer Softmax  Model
Predicted    0.0  1.0  __all__
Actual
0.0       56483    0    56483
1.0        3317    0     3317
__all__    59800    0    59800

** TEST: 2.0_layer | # Layers: 2.0 | Optimizer: Adam **

Model:  Builder(
  (layers): ModuleList(
    (0): Linear(in_features=101000, out_features=1000, bias=True)
    (1): ReLU()
    (2): Linear(in_features=1000, out_features=100, bias=True)
    (3): ReLU()
  )
)
2 Layer Relu  Model with droput
Predicted  False  True  __all__
Actual
False     59511   165    59676
True        123     1      124
__all__    59634   166    59800
```

General description of my work

The entire application is divided into four parts:

- Neural network design using Pytorch
- Error Validation and further optimization

Neural Network Design using pytorch

I have used pytorch framework to design the model. Have refined the modelBuilder class (originally created by Luke) to create the model with different configuration.

Below is the sample code for creating various network configuration use for testing

```
f __name__ == '__main__':
    layers = [
        nn.Linear(INPUT_SIZE, NUERONS_l1),
        nn.ReLU(),
        nn.Linear(NUERONS_l1, CHUNK_SIZE),
        nn.ReLU(),
    ]

## Sigmoid model
if __name__ == '__main__':
    layers2 = [
        nn.Linear(INPUT_SIZE, NUERONS_l1),
        nn.ReLU(),
        nn.Linear(NUERONS_l1, CHUNK_SIZE),
        nn.Sigmoid()
    ]


## Relu model 3 layer
if __name__ == '__main__':
    layers3 = [
        nn.Linear(INPUT_SIZE, NUERONS_l1),
        nn.ReLU(),
        nn.Linear(NUERONS_l1, NUERONS_l2),
        nn.ReLU(),
        nn.Linear(NUERONS_l2, CHUNK_SIZE),
        nn.ReLU()
    ]


## Softmax model
if __name__ == '__main__':
    layers4 = [
        nn.Linear(INPUT_SIZE, NUERONS_l1),
        nn.ReLU(),
        nn.Linear(NUERONS_l1, CHUNK_SIZE),
        nn.Softmax(dim=1)
    ]

## dropout model

if __name__ == '__main__':
    layers5 = [
        nn.Linear(INPUT_SIZE, NUERONS_l1),
        nn.ReLU(),
        nn.Linear(NUERONS_l1, CHUNK_SIZE),
        nn.ReLU(),
    ]
```

Error Validation and further optimization:

Wrote logic to translate tensor data given chunk size and validate against the target value. Also use it for both in-sample and out-sample testing. Further wrote logic to convert probabilistic output classified value which was use in adjusting the model accuracy.

Sample code for the Error validation is below:

```
for features, labels in test_loader:
    if self.stop_early_at and (counter + 1) % self.stop_early_at == 0:
```

```
            break
        #print("feature ",features)
        #print("label ",labels)
        features, labels = self._to_device(features, labels)
        features = features if not self.is_image else features.view(-1, self.dimensions)
        outputs = self.model(features)
        #print("PreOutputs ", outputs)
        if(self.adjustment == 1 ):
            #print("ADJUSTING")
            outputs = torch.ceil(outputs*self.adjustment)
            outputs = torch.clamp(outputs, min=0, max=1)
        else:
            if(self.adjustment!=0):
                outputs = torch.clamp(outputs, min=0, max=self.adjustment)
                scalar_multi=1/self.adjustment
                outputs = torch.round(outputs*scalar_multi)
            else:
                outputs = torch.round(outputs)

        #print("Outputs ",outputs)
        #_, predicted = torch.max(outputs.data, 1)
        total += labels.numel()
        #correct += (predicted == labels.type(torch.long)).sum()
        predicted = (outputs == labels).sum()
        correct += predicted
```

Summary and Conclusions:

The Neural network using pytorch gave us great flexibility to design this model framework to predict loan default model.

Our framework basically produces three-layer architecture for model engineering with first layer as data loader which control the fetching, massaging and feature selection of data; second layer being network design and third layer is error and validation logic.

With above architecture it is very easy to train model over large dataset and also optimized model over various feature selection and network designs.

There can be further improvement in this model by training it over larger dataset to improve the performance of true positive and also trying out various feature selection

Code Percentage

Code for internet: 1 lines
Code modified: 1 lines
Line written by me :150
Code percentage from internet: (2-2)/150 * 100 = 0%

References:

https://pytorch.org

Appendix: Sample Code Output

```
** INFO **
DATA_LEN: 638326
INPUT_SIZE: 101000

** TEST: 2.0_layer | # Layers: 2.0 | Optimizer: Adam **

Model:  Builder(
  (layers): ModuleList(
    (0): Linear(in_features=101000, out_features=1000, bias=True)
    (1): ReLU()
    (2): Linear(in_features=1000, out_features=100, bias=True)
    (3): ReLU()
  )
)
Epoch [1/1], Step [100/638326], Loss: 3.0000
Epoch [1/1], Step [200/638326], Loss: 10.0000
Epoch [1/1], Step [300/638326], Loss: 5.0000
Time Spent: 21.08156156539917
Time Spent: 37.01350545883179
Accuracy of the network: 92 %
```

END TEST: 2.0_layer

2 Layer Relu Model

Predicted False True __all__

Actual

False 55992 506 56498

True 3268 34 3302

__all__ 59260 540 59800

** TEST: 2.0_layer | # Layers: 2.0 | Optimizer: Adam **

Model: Builder(

 (layers): ModuleList(

  (0): Linear(in_features=101000, out_features=1000, bias=True)

  (1): ReLU()

  (2): Linear(in_features=1000, out_features=100, bias=True)

  (3): Sigmoid()

)

)

Epoch [1/1], Step [100/638326], Loss: 0.0065

Epoch [1/1], Step [200/638326], Loss: 0.0021

Epoch [1/1], Step [300/638326], Loss: 0.0008

Time Spent: 21.07642149925232

Time Spent: 37.07498121261597

Accuracy of the network: 97 %

END TEST: 2.0_layer

/usr/local/lib/python3.5/dist-packages/pandas/core/indexing.py:1494: FutureWarning:

Passing list-likes to .loc or [] with any missing label will raise

KeyError in the future, you can use .reindex() as an alternative.

See the documentation here:

https://pandas.pydata.org/pandas-docs/stable/indexing.html#deprecate-loc-reindex-listlike

return self._getitem_tuple(key)

2 Layer SIGMOID Model

Predicted 0.0 1.0 __all__

Actual

0.0 57872 0 57872

1.0 1928 0 1928

__all__ 59800 0 59800

** TEST: 3.0_layer | # Layers: 3.0 | Optimizer: Adam **

Model: Builder(

(layers): ModuleList(

(0): Linear(in_features=101000, out_features=1000, bias=True)

(1): ReLU()

(2): Linear(in_features=1000, out_features=1000, bias=True)

(3): ReLU()

(4): Linear(in_features=1000, out_features=100, bias=True)

(5): ReLU()

)

)

Epoch [1/1], Step [100/638326], Loss: 4.0000

Epoch [1/1], Step [200/638326], Loss: 2.0947

Epoch [1/1], Step [300/638326], Loss: 5.0000

Time Spent: 21.32636594772339

Time Spent: 37.35735464096069

Accuracy of the network: 92 %

END TEST: 3.0_layer

3 Layer RELu Model

Predicted False True __all__

Actual

False 55794 50 55844

True 3956 0 3956

__all__ 59750 50 59800

** TEST: 2.0_layer | # Layers: 2.0 | Optimizer: Adam **

Model: Builder(

 (layers): ModuleList(

  (0): Linear(in_features=101000, out_features=1000, bias=True)

  (1): ReLU()

  (2): Linear(in_features=1000, out_features=100, bias=True)

  (3): Softmax()

)

)

Epoch [1/1], Step [100/638326], Loss: 0.0100

Epoch [1/1], Step [200/638326], Loss: 0.0100

Epoch [1/1], Step [300/638326], Loss: 0.0100

Time Spent: 21.232450485229492

Time Spent: 37.57780337333679

Accuracy of the network: 88 %

END TEST: 2.0_layer

2 Layer Softmax Model

Predicted 0.0 1.0 __all__

Actual

0.0 56483 0 56483

1.0 3317 0 3317

__all__ 59800 0 59800

** TEST: 2.0_layer | # Layers: 2.0 | Optimizer: Adam **

Model: Builder(

 (layers): ModuleList(

  (0): Linear(in_features=101000, out_features=1000, bias=True)

  (1): ReLU()

  (2): Linear(in_features=1000, out_features=100, bias=True)

  (3): ReLU()

)

)

Epoch [1/1], Step [100/638326], Loss: 0.0000

Epoch [1/1], Step [200/638326], Loss: 0.0000

Epoch [1/1], Step [300/638326], Loss: 0.0000

Time Spent: 21.131920099258423

Time Spent: 37.608861684799194

Accuracy of the network: 99 %

END TEST: 2.0_layer

2 Layer Relu Model with droput

Predicted False True __all__

Actual

False 59511 165 59676

True 123 1 124

__all__ 59634 166 59800

Process finished with exit code 0