A Case Study on Deep Learning for Classification with Imbalanced Finance Data

Paul F. Rodriguez
San Diego Supercomputer Center

PI James F. Albertus
Tepper School of Business, Carnegie Mellon University

Bryon Gill
Pittsburgh Supercomputer Center,

XSEDE SOC20003 ECSS NAIP
Goals

- Explore the association between bankruptcy risk and equity returns through machine learning/deep learning models to address some discrepancies between theory and predictions
Modeling Strategy

- Evaluate deep feed forward neural networks
  - Deep NN have not had as big an impact in tabular data (ie structured data)
- Compare it to other ML models
- Evaluate sample weighting for imbalanced data
- Examine variable effects (Explainable AI)
- Evaluate recurrent neural networks
Challenges

• Positive case occur around 0.7% (highly imbalanced)
• Neural networks often require more data than other models.
• A good case study to explore regularization, performance, data representations, and variable importance with deep learning.
• Help address questions about financial variables and more generally nonlinear effects in data.
The Data

- We transformed the raw data into both non-sequential 2D datasets and sequential (time-series) 3D datasets for modeling.

- There are 76 real valued measurements, and several indicator variables. One indicator for type of firm consisted of about 100 mutually exclusive binary indicators.

- All real valued variables standardized (using training data mean, std)
The Data

- 1 row for each year of data for each firm
- Upto 38 years from 1980 to 2017
- 7932 firms
- 81295 total rows
- 180 columns
- 612 total number of bankruptcies

Inputs: 76 Real Variables (normalized); 103 Indicator variables (binary 0 or 1)

<table>
<thead>
<tr>
<th>Year</th>
<th>Col. 1</th>
<th>Col. 2</th>
<th>....</th>
<th>....</th>
<th>Indictor1</th>
<th>Indictor2</th>
<th>....</th>
<th>....</th>
</tr>
</thead>
<tbody>
<tr>
<td>1980</td>
<td>3.2</td>
<td>5.22</td>
<td>-3.2</td>
<td>-1.1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1985</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>
The Task, predict bankruptcy 1 year ahead

- Keep model ‘causal’

![Time line diagram showing training and validation years](image)
The Task, predict bankruptcy 1 year ahead

• Keep model ‘causal’

  Train on all years up to a ‘split year’
  Validation year is the next year

• 10 split years, 1997 to 2006, for training/validation to select hyperparameters
• 10 split years, 2007 to 2016, for training/test to estimate generalization error
• Low positive rate suggests we pool all predictions for each 10-year set
The Data as Sequence

• 3D array for sequences: N (num firms) x 38 years x 179 features

Inputs: Reals and Indicators

<table>
<thead>
<tr>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>....</th>
<th>T</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.11</td>
<td>0.53</td>
<td>0.34</td>
<td>0.725</td>
<td>0.824</td>
<td>0.45</td>
<td>0.61</td>
<td></td>
<td>0.06</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td></td>
<td>0</td>
</tr>
</tbody>
</table>

Time

N samples

• Note: most firms have data < 38 years, with different start/stop years

• 833 (of 8023) firms had 976 missing segments; 79 out 976 were after a bankruptcy [so removed 91 firms, 8 bankruptcies, with too much missing]
Data Description

Number Of Bankruptcies Across Years

Correlation of Variables to Bankruptcy Target
Model scripts

• All neural networks were implemented in Tensorflow/Keras 2.0+ (run with singularity image on Bridges2)

• All training was performed on Bridges2 using single regular CPU nodes.

• All other models implemented using sci-kit learn

• Run times for all model varied depending on number of hyperparameters, from 2-6 hours for regression models, 12-18 for nonlinear, and ensemble models
Workflow

- Run one Batch job for each model, or each NN seed, activation, etc..
- For each training/valid dataset
  - For each model configuration
    - Run Model, save predictions
- Pool predictions across 10 train/valid years, get best parameters
- Use best parameters on the 10 train/test years, pool predictions to get final metrics

Note: All predictions saved in one file with key values to indicate configuration
NN Hyperparameters Choosing

• Systematically searched: layers, units, learning rate
• Small search with some better configurations:
  • relu (rectified lin unit) vs selu (scaled exponential) activation
  • dropout vs batch normalization (batch norm better)
• Quick check for published recommendations
  • bias initialization to class base rate
  • selu with alpha dropout
  • Adam optimizer
• Explored a bit with skip connections from input (will revisit)
• Number of epochs were based on train/valid early stopping
Grid Search and Hypertuner (hyperband)

- Keras Hypertuner class implements several search strategies

- Hyperband is like a competition of network configurations, with incremental training, to weed out worse ones

- Future work to test:
  Bayesian optimization is like function approximation to pick out next configuration
Grid Search and Hypertuner (hyperband)

- Test: hyperband vs grid search among same discrete choices of layers, units, learning rates (225 combinations), 1 training/validation dataset
  - Both ran in about 40 minutes, 1 compute node
- Among top performing configurations (all datasets):
  - Small networks typically better than deeper networks but sometimes deep networks with smaller units perform near top (depending on hyperband parameters)
  - Both searches came up with 2 layers, 64 to 256, 0.004 to 0.01 range of better configurations
- Use one final grid search over this more focused range to get best hyperparameters
Study 1  Metrics

• The AUC ROC (area-under-curve of receiver operating characteristic) is a measure of true positive and false positive rates. 
  higher is better, 1.0 max 
  baseline is 0.5 

• The AUC PRC (precision-recall curve) is a measure of precision versus recall curve 
  higher is better, 1.0 max 
  baseline is percent positive case rate 

Results will show generalization performance on test years only
# Generalization Performance

<table>
<thead>
<tr>
<th>Model</th>
<th>ROCAUC</th>
<th>PRCAUC</th>
<th>TP</th>
<th>FP</th>
<th>FN</th>
</tr>
</thead>
<tbody>
<tr>
<td>Logistic Reg</td>
<td>0.95243</td>
<td>0.30445</td>
<td>25</td>
<td>26</td>
<td>126</td>
</tr>
<tr>
<td>Lasso Reg</td>
<td>0.80328</td>
<td>0.15351</td>
<td>11</td>
<td>41</td>
<td>140</td>
</tr>
<tr>
<td>Ridge Reg</td>
<td>0.94927</td>
<td>0.22636</td>
<td>0</td>
<td>0</td>
<td>151</td>
</tr>
<tr>
<td>Gradient Boosting</td>
<td>0.96082</td>
<td>0.24687</td>
<td>18</td>
<td>26</td>
<td>133</td>
</tr>
<tr>
<td>Random Forest</td>
<td>0.95986</td>
<td><strong>0.32071</strong></td>
<td>9</td>
<td>4</td>
<td>142</td>
</tr>
<tr>
<td>Neural Network Layer 2, Units 64</td>
<td>0.95833</td>
<td>0.2740</td>
<td><strong>52</strong></td>
<td>91</td>
<td>99</td>
</tr>
<tr>
<td>Neural Network ensemble average</td>
<td>0.96012</td>
<td>0.2720</td>
<td><strong>52</strong></td>
<td>78</td>
<td>99</td>
</tr>
</tbody>
</table>
Predictions unpooled over time

Over 20 yrs from NN ensemble

Blue: Number of Bankruptcies
Red: Number of Predicted Bankruptcies (TP+FP)

Validation Years 1998-2007  Test Years 2008-2017

First set of years has some trend
Second set of years seems to capture some information about levels
Might consider added variable for ‘previous-num-bankruptcies’
Study1  Program Note

- Neural network ensemble helped regularize false positives (less overfitting)

- Random forest with minimum 1 leaf size ran very long – ended up trying 3 to 5 as a hyperparameter

- Also tried Kernel Ridge regression but jobs were very long – might revisit
Study 2: Using sample weighting for imbalance

- All models using gradient descent had cases weighted by $1/(\text{class probability})$

- Random Forest uses a ‘balance’ option for weighing split improvements by class probability

- New model: Balanced Random Forest under-samples the negative cases so that each bootstrap sample has equal pos/neg cases (used Python virtual environment)
<table>
<thead>
<tr>
<th>Model</th>
<th>ROCAUC Valid set</th>
<th>PRCAUC</th>
<th>TP</th>
<th>FP</th>
<th>FN</th>
</tr>
</thead>
<tbody>
<tr>
<td>Logistic Reg</td>
<td>0.9420</td>
<td>0.253</td>
<td>126</td>
<td>1600</td>
<td>25</td>
</tr>
<tr>
<td>Lasso Reg</td>
<td>0.863982</td>
<td>0.43613</td>
<td>123</td>
<td>2068</td>
<td>28</td>
</tr>
<tr>
<td>Ridge Reg</td>
<td>0.957404</td>
<td>0.287</td>
<td>136</td>
<td>2256</td>
<td>15</td>
</tr>
<tr>
<td>Gradient Boosting</td>
<td>0.961471</td>
<td>0.26021</td>
<td>136</td>
<td>2056</td>
<td>15</td>
</tr>
<tr>
<td>Random Forest</td>
<td>0.9629</td>
<td>0.253</td>
<td>24</td>
<td>19</td>
<td>127</td>
</tr>
<tr>
<td>Balanced Random Forest</td>
<td><strong>0.96328</strong></td>
<td>0.2249</td>
<td>142</td>
<td>2880</td>
<td>9</td>
</tr>
<tr>
<td>Neural Network Layer 2, Units 512, Lrate 0.01</td>
<td>0.95856</td>
<td>0.262</td>
<td><strong>144</strong></td>
<td>4099</td>
<td>7</td>
</tr>
<tr>
<td>Neural Net ten ensemble average</td>
<td>0.955165</td>
<td>0.283</td>
<td>143</td>
<td>4053</td>
<td>8</td>
</tr>
</tbody>
</table>
Study 3 Custom Loss with Market Capitalization

• What if the sample weighting could take into account the firm size?

• We built custom loss function to explore such possibilities
## Generalization Performance with weighted Loss by market cap

<table>
<thead>
<tr>
<th>Model</th>
<th>ROCAUC (Valid set)</th>
<th>PRCAUC</th>
<th>TP</th>
<th>FP</th>
<th>FN</th>
<th>False Neg Loss</th>
</tr>
</thead>
<tbody>
<tr>
<td>Logistic Reg</td>
<td>0.91460</td>
<td>0.168</td>
<td>32</td>
<td>107</td>
<td>119</td>
<td>511.93</td>
</tr>
<tr>
<td>Lasso Reg</td>
<td>0.64179</td>
<td>0.191</td>
<td>44</td>
<td>483</td>
<td>107</td>
<td>495.95</td>
</tr>
<tr>
<td>Ridge Reg</td>
<td>0.9182</td>
<td>0.181</td>
<td>31</td>
<td>125</td>
<td>120</td>
<td>509.65</td>
</tr>
<tr>
<td>Gradient Boosting</td>
<td><strong>0.94869</strong></td>
<td>0.182</td>
<td>31</td>
<td>84</td>
<td>120</td>
<td>506.17</td>
</tr>
<tr>
<td>Neural Network</td>
<td>0.937924</td>
<td>0.147</td>
<td><strong>81</strong></td>
<td>654</td>
<td>70</td>
<td><strong>438.24</strong></td>
</tr>
<tr>
<td>Neural Net ten ensemble average</td>
<td>0.935432</td>
<td>0.126</td>
<td>72</td>
<td>480</td>
<td>79</td>
<td>450.52</td>
</tr>
</tbody>
</table>

More False-Neg than sample weighting, and worse loss (not shown)
May need to re-scale smaller bankrupt cases
In Keras, you can substitute your own loss function into `model.fit()`

In Keras, the fit function only passes: `(True_Target, Predictions)` but it does not indicate if its training, validation, or batch subset so if custom function needs to use other data, is has to figure out which rows it is processing

2 ways to approach it –

A. use shuffle=False, and batch size = size of input and `tf.shape` (not `object.shape`) to figure out which rows

B. add extra columns into True Target, but then all your metrics will have to be customized
Study 4 Recurrent Network

- Recurrent means that network states are recurrently fed back at the next time step

- Recurrent Network can handle different length sequences ("ragged sequences")

- So far: A small search around best feed forward parameters using GRU, SRN type of layers, tanh or selu activation
Results Recurrent Networks

- So far, RNNs have weak positive case predictions.
- Plan to try fuller grid search and/or hyperparameter tuning.
Recurrent Network program note

• Keras provides a Masking layer option:
  Masking(mask_value=0., input_shape=[None, nvar])

• pick a mask value (eg all input=0 means mask)

• It’s a little like padding at the end or beginning, but what about the middle?

  Missing years or Masked years just carry forward last state
Keras has ‘gradient tape’ functions to track $\frac{dF}{dx}$
Often used for images
Correlates well with raw correlations-to-target

Explainable AI: Integrated Gradients to measure dependency

$$\sum \frac{dF}{dx} \quad \text{for F model, x input}$$

Correlation is 0.456
Explainable AI, leveraging other models

Trained Network(s)

⇒ Raw Prediction scores on training set

<table>
<thead>
<tr>
<th>Int Gradients</th>
<th>Var. Import</th>
<th>Var. Import</th>
</tr>
</thead>
<tbody>
<tr>
<td>LnAt</td>
<td>4.3</td>
<td>PiNaive</td>
</tr>
<tr>
<td>PiNaive</td>
<td>3.9</td>
<td>MveDebt</td>
</tr>
<tr>
<td>ReLct</td>
<td>3.5</td>
<td>WcapAt</td>
</tr>
<tr>
<td>ChSale</td>
<td>3.4</td>
<td>LctCheAt</td>
</tr>
<tr>
<td>Sigma</td>
<td>3.2</td>
<td>LctAt</td>
</tr>
</tbody>
</table>

Train 1 Regression Tree – no pruning necessary – it can reproduce Network classifications

Get Variable Importance scores
Browse Tree
Modeling Practical Experience

- Very deep NN not seeming necessary, but 2 hidden layers often most effective
- Ensembles of non-linear models compare well with networks
- Integrated Gradients may help measure variable effects
- Ongoing: systematic hyperparameter search with recurrent neural networks, explore skip connections
- Future work possibly includes a new dataset using financial distress
Quick check on GPU vs CPU (RM node)

Train time in seconds, 25 epochs

Number of layers

- CPU 1024 units per layer
- CPU 512 units per layer
- GPU 1024 units per layer
- GPU 512 units per layer
References

[12] Saito, T., & Rehmsmeier, M. 2015. The precision-recall plot is more informative than the ROC plot when evaluating binary classifiers on imbalanced datasets. PloS one, 10(3), e0118432. https://doi.org/10.1371/journal.pone.0118432