Final Report

Author: Tao Li (u0941298); Yupeng Zhang (u0939637)

Abstract

The intention of this project is to develop a model that can recognize if a non-profit organization will face financial default given its historical tax report. The data are provided by GuideStar who has tons of tax forms (990 Form). We exam on different learning models and check out their performance on such time series analysis problem, it turns out neural networks is more powerful than linear models (e.g. Perceptron and SVM). Also, we observe that feature engineering is more easier on linear models and less effective on neural networks.

Introduction

The data consist of financial statements from 2009 to 2013 for each organization. Each organization has detailed statement on five categories: revenue, expenses, assets, liabilities, and balances. The objective is to predict if a specific organization will have financial risk in the future. We view this as a time series analysis problem, in which we have a label y_i for each year i's statement x_i, and the goal is to predict y_{i+1} given $x_{i,i-1,...}$ and $y_{i,i-1,...}$. Note, x_{i+1} is not involved here since if we have x_{i+1} we would have already know y_{i+1} from balance information.

We choose to exam various models and find out which one is more suitable for this problem by comparing best performance of cross validations. Our choices are Perceptron, SVM, Feed-forward Neural Networks, and Long Short-Term Memory recurrent networks. The first two have simpler model setting and human-readable weights, while the second two are more robust, highly nonlinear, and naturally more expressive for time-series analysis. We will seperately compare their testing results.

Perceptron

Features and Model Setup

We opt to use Aggressive Perceptron with Margin [3]. Featues we used are

1. ratio of revenue over expenses
2. difference between revenue and expenses
3. ratio of increase in $revenue - expenses$
4. ratio of assets over liabilities
5. difference between assets and liabilities
6. ratio of increase in $assets - liabilities$

We extract feature $1, 2, 4, 5$ on each year's statement in $[2009, 2012]$, and feature $3, 6$ on two adjacent year's statement, giving us 23 dimensional feature. Taking 2013's balance as final label (eigher positive or negative), we take 50 positive examples and 50 negative examples for cross validation.

Cross Validation

With training epoch set to 20, our 3-fold cross validations on margin μ are

μ	1	2	3	4	5	6	7	8
Error Rate	0.488	0.477	0.476	0.478	0.486	**0.438**	0.470	0.476

Table 1.1: Table of Cross Validation on μ

With the best value of mu, we test on different epochs

epoch	5	10	20	30	40	50
Error Rate	0.453	0.464	0.427	0.431	0.427	0.421

Table 1.2: Table of Cross Validation on epoch

It seems Perceptron works not so good on our time-series analysis.

SVM

Features and Model Setup

We use the same feature as in Perceptron and standard form of SVM [4] with learning rate decay factor. It's loss function for SGD is

$$J_i(w) = \frac{1}{2}w^2 + C\max(0, 1 - y_i w^T x_i)$$

whose update rule is $w = w - r\frac{\partial J_i(w)}{\partial w}$, and $r = \frac{\rho_0}{1+\rho_0 t/C}$

Cross Validation

Cross validations are done on ρ_0 and C

$\rho_0 \backslash C$	0.01	0.1	1
hline 1	0.59	0.56	0.43
0.5	0.54	**0.30**	0.46
0.1	0.39	0.48	0.60
0.01	0.38	0.47	0.32

Table 1.3: Table of Cross Validation on ρ_0 and C

Fixing $rho_0 = 0.5$ and $C = 0.1$, we do another CV on epochs

epoch	5	10	20	30	40	50
Error Rate	0.544	0.336	0.370	0.467	0.327	0.348

Table 1.4: Table of Cross Validation on epoch

One observation is that there are certain correlation between future financial default and historical balance statement. We can see SVM performs much better than Perceptron.

Feed-forward Neural Networks

Features and Model Setup

Unlike previous models, we take all raw data as direct input of the model. We opt to use window-based approach to model time-series analysis that, we flatten the history from 2009 to 2012 as input to feed-forward neural nets and the output is the a binary label indicating if 2013 will have financial default.

We used PyBrain ([2]) to build a neural nets with three layers. First layer is a linear layer consisting of $91 \cdot 4 = 364$ dimension (which is the dimension of concatenating data from 2009 to 2012). Hidden layer consists of twice the units of the first layer, and has non-linear hyperbolic tangent unit. Output layer consists of two units upon which we use an extra Softmax layer to do binary prediction. The loss function is of the "standard" form of feedforward neural nets ([1])

Cross Validation

We intetionally organized the 100 examples in which every positive example is followed by a negative example. Doing this will allow us to split the small amount of data into several folds and guarantee each fold contains roughly equivalent number of positive and negative examples, and thus know if our model performs better than random guess.

We split the data into 4 folds and do cross validation on learning rate α and weight decay factor λ. Since the model uses SGD which has noisy convergence path, so for each validation set (and respective training set), we do the learning and testing for 3 times and calculate the mean of all 3 testing error rate. Note, this is important to have reproducible results, since

our experience shown that the final prediction error rate could vary a lot even on the same validation setting (from 0.25 to 0.50). Also we fix the number of passes for blackbox SGD solver to be 50. Experience shown this is sufficient even for very low learning rate.

$\lambda \backslash \alpha$	10^{-5}	10^{-4}	10^{-3}	10^{-2}	10^{-1}
10^{-5}	39.67	43.67	42.87	41.86	43.00
10^{-4}	45.67	42.00	45.67	44.00	48.93
10^{-3}	44.92	43.91	42.01	39.33	41.03
10^{-2}	38.00	**37.99**	38.03	42.87	39.96
10^{-1}	41.06	40.08	43.00	41.06	43.11

Table 1.5: Table of Cross Validation Error Rates on Detailed Balance Data

One observation is that the model is of high variance in that, with uniformly random initial weights in net connections, there is no guarantee that we can have the same performance even on the same training set and testing set. This is predictable since neural networks are highly nonlinear and thus have more complicated hypothesis landscape than linear models, so cases are the model converges to different local optima.

LSTM

Features and Model Setup

Still we use the same features as in feed-forward nets, but with quite different model setup. We extract a time-seriese sequence for each organization from its history (2009-2012). For each year i in $[2009, 2011]$, we build a training sequence as $[(x_i, y_i), y_{i+1}]$, where (x_i, y_i) are training features and y_{i+1} is the label. Thus for each example, we have a sequence of 3 time stamps. The testing set is build from 2012's data and 2013's label as $[(x_{2012}, y_{2012}), y_{2013}]$.

Again, we use Pybrain's built in LSTM model, in which we set input layer to be $91 + 1 = 92$ dimension, twice recurrent memory units, and 1 output unit.

Cross Validation

Still we use 4-fold cross validation to test learning rate α and weight decay factor λ. With detailed balance data, we run 100 passes of SGD and results are

$\lambda \backslash \alpha$	10^{-5}	10^{-4}	10^{-3}	10^{-2}	10^{-1}
10^{-5}	39.01	37.97	38.98	38.98	38.00
10^{-4}	38.92	**36.96**	39.01	38.98	39.99
10^{-3}	39.13	39.98	37.97	40.94	38.12
10^{-2}	40.91	40.00	37.12	38.98	40.00
10^{-1}	39.99	40.00	37.97	38.92	39.93

Table 1.6: Table of Cross Validation Error Rates on Detailed Balance Data

Interesting thing is LSTM is more stable than feed-forward neural nets in that results merely change with different parameters and random initial weights.

Interesting Things

With slight feature engineering, simple linear model is able to compete with much more complicated neural networks. Further, we observe that with linear model (e.g. SVM), it's much easier to have an intuitive sense on the usefulness of features, since we can identify weights on important features and useless ones. However, in neural networks, the weights are much less readable.

Future Work

Given more time in future, we could do more test on LSTM. Since currently we only use Pybrain blackbox implementation, there are something we don't fully understand, such as why LSTM is more stable than feed-forward nets. Also, we could extract more structural features that reflects more changes with respect to time. Further, we could do detailed analysis on the pros and cons of each model with respect to this problem.

Bibliography

[1] A Ng. Ufldl tutorial on neural networks. *Ufldl Tutorial on Neural Networks*, 2011.

[2] Tom Schaul, Justin Bayer, Daan Wierstra, Yi Sun, Martin Felder, Frank Sehnke, Thomas Rückstieß, and Jürgen Schmidhuber. Pybrain. *The Journal of Machine Learning Research*, 11:743–746, 2010.

[3] K Singer. Online classification on a budget. *Advances in neural information processing systems*, 16:225, 2004.

[4] Johan AK Suykens and Joos Vandewalle. Least squares support vector machine classifiers. *Neural processing letters*, 9(3):293–300, 1999.