Using Neural Networks for Prediction of the US Economic Recessions

(with a Large Set of Predictors)

Abstract

Despite of its crucial importance, the prediction of economic recession continues to be a tough task for economists using traditional methods since the true underlying functional form between leading economic indicators and economic recessions is still unknown. As nonlinear and nonparametric models, Neural Networks are expected to perform well when there is little prior knowledge about the functional form of the relationship under investigation. Using a monthly data set on 21 economic indicators during 1959-2016, we examine the accuracy of neural network models in economic recession prediction. The preliminary results show that NNs yield high accuracy of over 95% and make lower errors compared to the conventional linear regression model.

Introduction

Economic growth affects almost everyone in the economy including lending institutions, stock brokers, and financial officers of corporations, and it dictates in many instances whether a corporation will accept a particular project or if banks will make a specific loan. Numerous financial and economic decisions depend on economic growth expectations, and this dependency continues to encourage economists to search out methods for accurately forecasting economic growth.

Economic theory does not always yield a specific functional form that is to be used for empirical verification of the theory. Researchers have traditionally relied on linear models because of their ease of estimation, but with the advent of modern computing technology, one can venture beyond the bounds of linearity.

The economy is evolving (rather slowly) over time. This feature cannot easily be captured by fixed specification linear models due to evolving coefficient estimate. Many factors interact in finance and economics including political events, general economic conditions, and traders' expectations. Therefore, predicting finance and economics movements is quite difficult.

Recently, there has been considerable interest in applications of Neural Networks in the economics literature, particularly in the areas of financial statistics and exchange rates.[1] In contrast, relatively few studies have applied neural network methods to macroeconomic time series. Time series in finance and economics has the following features: (1) Data intensity (2) Unstructured nature (3) High degree of uncertainty (4) Hidden relationships.

Neural networks (NNs) models have emerged as a powerful statistical modeling technique that can detect the underlying functional relationships within a set of data and perform tasks such as pattern recognition, classification, evaluation, modeling, and prediction. Neural networks are particularly well suited to finding accurate solutions in an environment characterized by complex, noisy, irrelevant or partial information which is the case in economic indicators of recessions.

---

1. For example, see recent work on financial applications of neural networks by Fernandez-Rodriguez et al. (2000) and Refenes and White (1998)

The network is a set of artificial neurons, connected like neurons in the brain. It learns associations by seeing lots of examples of each class, and learning the similarities and differences between them. This is thought to be a similar process to how the brain learns, with repeated exposure to a pattern forming a stronger association over time.

The application of neural network involves the interaction of many diverse variables that are highly correlated, frequently assumed to be nonlinear, not clearly related, and too complex to be described by a mathematical model. Several distinguishing features of NNs make them valuable and attractive in forecasting. First, ANNs are nonlinear data-driven. Thus they are more general and flexible modeling tools for forecasting. Second, NNs are universal functional approximators. It has been shown that a network can approximate any continuous function to any desired accuracy. Third, NNs can generalize. After learning the data presented to them, they can often correctly infer the unseen part of a population even if the sample data contain noisy information.

Any time series forecasting model assumes that there is an underlying process from which data are generated and the future value of a time series is solely determined by the past and current observations. Neural networks are able to capture the underlying pattern or autocorrelation structure within a time series even when the underlying law governing the system is unknown or too complex to describe. This is the case when we want to predict economic recessions since they do not happen based on a predetermined structure or clear underlying law.

Researchers such as Cover (1992), and Rhee and Rich (1995) have found empirically that expansionary monetary policy, as measured by either money or interest rates, has a marginally lower impact on output growth than contractionary policy in the United States. These findings lend credence to Friedman's proposition. Thus, non-linearities may very well exist in indicator models of output growth that are constructed using monetary and financial variables, and they may be representative of asymmetric effects of monetary policy on the real economy. In this paper, we apply neural network model on a set of economic indicators to examine whether it yields high precision and lower error compared to conventional linear regression.

Literature review

Kohzadi et al. (1995) is a good introduction to neural networks and their uses in economics, with an application to the forecast of corn futures. Comparing a neural network to an ARIMA model, they find that the forecast error of the neural net model is between 18 and 40 per cent lower than that of the ARIMA model, using different forecasting performance criteria.

In the finance literature, Angstenberger (1996) develops a neural net model that has some success at forecasting the S&P 500 index. Hutchinson, Lo, and Poggio (1994) find that neural network models can sometimes outperform the parametric Black-Scholes option pricing formula. However, Campbell, Lo, and MacKinlay (1997) caution that this finding is model specific, and that one should not conclude that neural network models are generally superior until they have been tested against additional parametric models.

Zhang, Yu, Wang and Xie (2010) examine the relevance of various financial and economic indicators in forecasting business cycle turning points using neural network (NN) models. They use a three-layer feed-forward neural network model to forecast turning points in the business cycle of China. The NN model uses 13 indicators of economic activity as inputs and produces the probability of a recession as its output. Results indicate that the asymmetry of business cycle can be verified using their NN method.

Nyman and Ormerod (2017) show that the machine learning technique of random forests has the potential to give early warning of recessions in the US. They use a small set of explanatory variables from financial markets which would have been available to a forecaster at the time of making the forecast. The algorithm they used never predicts a recession when one did not occur.

Data

Clearly, there is no single indicator, or even fixed set of indicators, that contain comprehensive information about the state of the economy 3, 6, or 12 months from now. Therefore, we consider a set of 21 monthly variables chosen to describe different aspects of the economy during 1959 to 2016. Broadly speaking, these indicators are measures of real economic activity such as labor market indicators, and forward-looking financial variables such as interest rates, Treasury yield curve and indicators of housing, stock and money markets. The data set was obtained from the Federal Reserve Economic Data (FRED) website of the Federal Reserve Bank of St. Louis.

The research division of the bank provides a macroeconomic database of 134 monthly US indicators. The data set is updated on a timely basis and can be downloaded for free from the website http://research.stlouisfed.org/econ/mccracken/sel/. We use this data set since it relieves researchers from the burden of handling data changes and revisions. Moreover, working with a more or less standard database should also facilitate replication and comparison of results. A full list of the data is given in the Appendix in Table 1. We choose 21 out of 134 economic indicators based on their availability since 1959 until 2016 to set up a data set as large as possible. Among the available variables during this period, we took those that are real measures of economic activity based on literature which are expected to help determine the existence of economic recession.

To examine whether NN models yield appropriate predictions on the economic recessions in the US, we use a dummy variable which takes value 1 if there's a recessionary period, and 0 if an expansionary period exists. Data on this dummy variable is provided by National Bureau of Economic research (NBER).

Algorithm and R Package

Artificial neural network (ANNs) are a class of typical intelligent learning paradigm, widely used in practical application domains including: pattern classification, function approximation, optimization, prediction and automatic control and many others. In this paper, resilient back-propagation algorithm is used for modeling the time series and predict recessions in the United States.

Topological structure of BP neural network consists of input layer, output layer and a number of hidden layers, each layer contains a number of neurons, and each neuron is connected with a different layer of neurons. There is no connection on neurons of the same layer. The learning algorithm of BP neural network is forward and backward propagation. Forward propagation allows the input information to be propagated to the output layer, and activation functions work on the corresponding weights. When the error between the output and the desired value is bigger than the given precision value, it will make the error propagate back. In the process of error propagation, the weights and thresholds of each layer are modified. So, with the repeated iterative operation, finally the error of the transmission signal will reach to the given precision.

neuralnet package in R

The package neuralnet focuses on multi-layer perceptrons (MLP, Bishop, 1995), which are well applicable when modeling functional relationships. The neurons are organized in layers, which are usually fully connected by synapses. neuralnet depends on two other packages: grid and MASS. In neuralnet, a synapse can only connect to subsequent layers. To each of the synapses, a weight is attached indicating the effect of the corresponding neuron, and all data pass the neural network as signals. The signals are processed first by the so-called integration function combining all incoming signals and second by the so-called activation function transforming the output of the neuron.

The simplest multi-layer perceptron consists of an input layer with n covariates and an output layer with one output neuron.

$$o(\mathbf{x}) = f\left(w_0 + \sum_{i=1}^{n} w_i x_i\right) = f\left(w_0 + \mathbf{w}^T \mathbf{x}\right),$$

It calculates the function where w_0 denotes the intercept, $w = (w_1,...,w_n)$ the vector consisting of all synaptic weights without the intercept, and $x = (x_1,. . ., x_n)$ the vector of all covariates. The function is mathematically equivalent to that of GLM with link function f^{-1}. Therefore, all calculated weights are in this case equivalent to the regression parameters of the GLM. An MLP with a hidden layer consisting of J hidden neurons calculates the following function:

$$\begin{aligned} o(\mathbf{x}) &= f\left(w_0 + \sum_{j=1}^{J} w_j \cdot f\left(w_{0j} + \sum_{i=1}^{n} w_{ij} x_i\right)\right) \\ &= f\left(w_0 + \sum_{j=1}^{J} w_j \cdot f\left(w_{0j} + \mathbf{w_j}^T \mathbf{x}\right)\right), \end{aligned}$$

where w_0 denotes the intercept of the output neuron and w_{0j} the intercept of the *jth* hidden neuron. Additionally, w_j denotes the synaptic weight corresponding to the synapse starting at the *jth* hidden neuron and leading to the output neuron, $w_j = (w_{1j},...,w_{nj})$ the vector of all synaptic weights corresponding to the synapses leading to the *jth* hidden neuron, and $x = (x_1,..., x_n)$ the vector of all covariates. This shows that neural networks are direct extensions of GLMs. However, the parameters, i.e. the weights, cannot be interpreted in the same way anymore. Here, $f: R \rightarrow R$ is the activation function.

All hidden neurons and output neurons calculate an output $f(g(z_0, z_1,. . ., z_k)) = f(g(z))$ from the outputs of all preceding neurons $z_0, z_1,..., z_k$, where $g : R^{k+1} \rightarrow R$ denotes the integration function and $f: R \rightarrow R$ the activation function. The neuron $z_0 \equiv 1$ is the constant one belonging to the intercept. The integration function is often defined as $g(z) = w_0z_0 + \sum w_iz_i = w_0 + w^Tz$ $(i=1$ to $k)$. The activation function f is usually a bounded non-decreasing nonlinear and differentiable function such as the logistic function $f(u) = (1+e^{-u})^{-1}$ or the hyperbolic tangent. It should be chosen in relation to the response variable as it is the case in GLMs. The logistic function is, for instance, appropriate for binary response variables since it maps the output of each neuron to the interval [0, 1]. At the moment, neuralnet uses the same integration as well as activation function for all neurons.

neuralnet focuses on supervised learning algorithms. These learning algorithms are characterized by the usage of a given output that is compared to the predicted output and by the adaptation of all parameters

according to this comparison. The parameters of a neural network are its weights. All weights are usually initialized with random values drawn from a standard normal distribution. During an iterative training process, the following steps are repeated:

The neural network calculates an output *o(x)* for given inputs *x* and current weights. If the training process is not yet completed, the predicted output *o* will differ from the observed output *y*. An error function *E*, like the sum of squared errors (SSE)

$$E = \frac{1}{2}\sum_{l=1}^{L}\sum_{h=1}^{H}(o_{lh} - y_{lh})^2$$

measures the difference between predicted and observed output, where $l = 1,..., L$ indexes the observations, i.e. given input-output pairs, and $h = 1,..., H$ the output nodes. All weights are adapted according to the rule of a learning algorithm. The process stops if a pre-specified criterion is fulfilled, e.g. if all absolute partial derivatives of the error function with respect to the weights $(\partial E/\partial w)$ are smaller than a given threshold. A widely used learning algorithm is the resilient backpropagation algorithm which we apply in this paper. It is based on the traditional backpropagation algorithm that modifies the weights of a neural network in order to find a local minimum of the error function. Therefore, the gradient of the error function (dE/dw) is calculated with respect to the weights in order to find a root. In particular, the weights are modified going in the opposite direction of the partial derivatives until a local minimum is reached.

Unlike the traditional backpropagation algorithm, a separate learning rate, which can be changed during the training process, is used for each weight in resilient backpropagation. This solves the problem of defining an over-all learning rate that is appropriate for the whole training process and the entire network. Additionally, instead of the magnitude of the partial derivatives only their sign is used to update the weights. This guarantees an equal influence of the learning rate over the entire network (Riedmiller and Braun, 1993).

Model

We use the data set *macro* which contains 21 economic indicators on a monthly basis during 1959-2016 (696 observations with data description in the Appendinx). To examine whether neural network yields better results compared to conventional linear regression, we proceed by randomly splitting the data into a train and a test set to fit a linear regression model and test it on the test set. Note that the gml() function is used instead of the lm() since this will become useful later when cross validating the linear model.

```
# Linear regression
index <- sample(1:nrow(macro),round(0.75*nrow(macro)))
train <- macro[index,]
test <- macro[-index,]
lm.fit <- glm(RECESSION~., data=train)
summary(lm.fit)
pr.lm <- predict(lm.fit,test)
MSE.lm <- sum((pr.lm - test$RECESSION)^2)/nrow(test)
pr.lm <- predict(lm.fit,test)
```

```
MSE.lm
## [1] 0.04448259
```

Since we are dealing with a regression problem, we use the mean squared error (MSE) as a measure of how much our predictions are far away from the real data (Estimates of coefficients in the Appendix).

Before fitting a neural network, we scale the data using min-max method. Without scaling or normalization, the algorithm may not converge before the number of maximum iterations allowed.

```
# scale the data in the interval [0,1]
maxs <- apply(macro, 2, max)
mins <- apply(macro, 2, min)
scaled <- as.data.frame(scale(macro, center = mins, scale = maxs - mins))
train_ <- scaled[index,]
test_ <- scaled[-index,]
n <- names(train_)
f <- as.formula(paste("RECESSION ~", paste(n[!n %in% "RECESSION"], collapse = " +
")))
```

In this section we run neuralnet function on the scaled data set to compare error term with that of linear regression. The most important arguments of the function are the following:

- *formula*, a symbolic description of the model to be fitted (see above). No default.
- *data*, a data frame containing the variables specified in formula. No default
- *hidden*, a vector specifying the number of hidden layers and hidden neurons in each layer.
- *threshold*, an integer specifying the threshold for the partial derivatives of the error function as stopping criteria. Default: 0.01.
- *rep*, number of repetitions for the training process. Default: 1.
- *startweights*, a vector containing prespecified starting values for the weights. Default: random numbers drawn from the standard normal distribution
- *algorithm*, a string containing the algorithm type. Possible values are "backprop", "rprop+", "rprop-", "sag", or "slr". "backprop" refers to traditional backpropagation, "rprop+" and "rprop-" refer to resilient backpropagation with and without weight backtracking and "sag" and "slr" refer to the modified globally convergent algorithm (grprop). Default: "rprop+"
- *err.fct*, a differentiable error function. The strings "sse" and "ce" can be used, which refer to ’sum of squared errors’ and ’cross entropy’. Default: "sse"
- *act.fct*, a differentiable activation function. The strings "logistic" and "tanh" are possible for the logistic function and tangent hyperbolicus. Default: "logistic"
- *linear.output*, logical. If act.fct should not be applied to the output neurons, linear.output has to be TRUE. Default: TRUE
- *likelihood*, logical. If the error function is equal to the negative log-likelihood function, likelihood has to be TRUE. Akaike’s Information Criterion (AIC, Akaike, 1973) and Bayes Information Criterion (BIC, Schwarz, 1978) will then be calculated. Default: FALSE

Since the response variable is binary, the activation function could be chosen as logistic function (default) and the error function as sum of squared errors (err.fct="sse"). Additionally, the item linear.output should be stated as FALSE since we have a binary output and want to do the classification. The number of hidden neurons should be determined in relation to the needed complexity of underlying relation.

The complexity of the calculated function increases with the addition of hidden layers or hidden neurons. The default value is one hidden layer with one hidden neuron. As far as the number of neurons is concerned, it should be between the input layer size and the output layer size, usually 2/3 of the input size as a rule of thumb. However, testing again and again is the best solution since there is no guarantee that any of these rules will fit the model best. After testing again and again, we found that two hidden layers each with 8 and 6 neurons respectively yield the best model fit.

```
# Neuralnet with two layers and total of 14 neurons
nn <- neuralnet(f,data=train_,hidden=c(8,6),linear.output=F)
head(nn$result.matrix,3)
```

```
##                                          1
## error                        0.011114059302
## reached.threshold            0.008729249665
## steps                     1729.000000000000
```

The training process needed 1729 steps until all absolute partial derivatives of the error function were smaller than 0.01 (the default threshold). The given data is saved in nn$covariate and nn$response as well as in nn$data for the whole data set inclusive non-used variables. The output of the neural network, i.e. the fitted values o(x), is provided by nn$net.result:

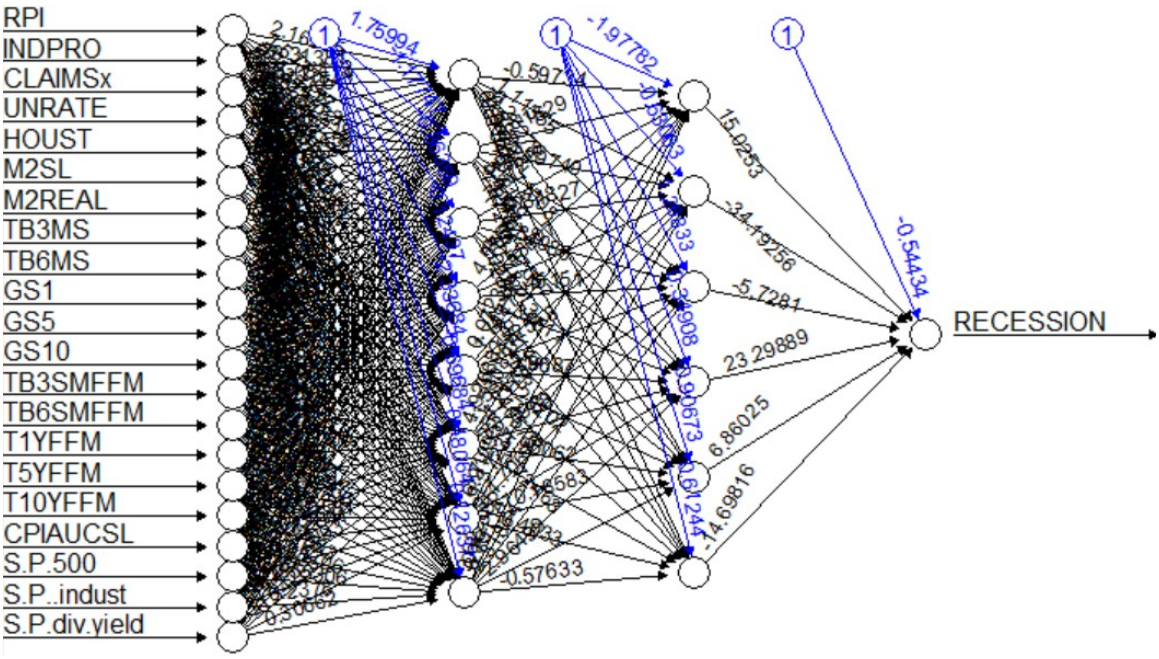
```
out <- cbind(nn$covariate,
nn$net.result[[1]])
dimnames(out) <- list(NULL,
c("RPI" , "INDPRO" , "CLAIMSx" , "UNRATE" , "HOUST" , "M2SL" , "M2REAL" ,
"TB3MS" , "TB6MS" , "GS1" , "GS5" , "GS10" , "TB3SMFFM" , "TB6SMFFM" , "T1YFFM" ,
"T5YFFM" , "T10YFFM" , "CPIAUCSL" , "S.P.500" , "S.P..indust" ,
"S.P.div.yield","nn-output"))[2]
```

In this case, the object nn$net.result is a list consisting of only one element relating to one calculated replication. If more than one replication were calculated, the outputs would be saved each in a separate list element.

The package neuralnet provides the plot function to visualize a built neural network and the gwplot function to visualize generalized weights. In the next section, we are going to visualize the results of neural network model.

2. The head of out is shown in the Appendix

```
plot(nn)
```



The black lines show the connections between each layer and the weights on each connection while the blue lines show the bias term added in each step. The bias can be thought as the intercept of a linear model. The net is essentially a black box so we cannot say that much about the fitting, the weights and the model. Suffice to say that the training algorithm has converged and therefore the model is ready to be used.

There are three types of neurons within the network: input neurons, hidden neurons, and output neurons. In the network, neurons are connected; the connection strength between neurons is called weights. If the weight is greater than zero, it is in an excitation status. Otherwise, it is in an inhibition status. Input neurons receive the input information; the higher the input value, the greater the activation. Then, the activation value is passed through the network in regard to weights and transfer functions in the graph. The hidden neurons (or output neurons) then sum up the activation values and modify the summed values with the transfer function. The activation value then flows through hidden neurons and stops when it reaches the output nodes. As a result, one can use the output value from the output neurons to classify the data.

Predicting Recession using NN

```
pr.nn <- compute(nn,test_[,1:21])
results <- data.frame(actual = test_$RECESSION, prediction = pr.nn$net.result)
results$prediction <- round(results$prediction)
a <- subset(results, results$actual== results$prediction)
nrow(a)/nrow(results)
## [1] 0.9827586207[3]
cm <- table(results$prediction, results$prediction)
cm
```

3. The results are indicated in the Appendix which show actual and prediction values of the RECESSION

```
##           prediction
## Actual       0   1
            0 154   1
            1   2  17
sum(diag(cm)) / sum(cm)
## [1] 0.98275862
```

To compare the results of two methods note that the net will output a normalized prediction, so we need to scale it back in order to make a meaningful comparison (or just a simple prediction). Now we can try to predict the values for the test set and calculate the MSE.

```
pr.nn_ <- pr.nn$net.result*(max(macro$RECESSION)-min(macro$RECESSION))
+min(macro$RECESSION)
test.r <- (test_$RECESSION)*(max(macro$RECESSION)-min(macro$RECESSION))
+min(macro$RECESSION)
MSE.nn <- sum((test.r - pr.nn_)^2)/nrow(test_)
print(paste(MSE.lm,MSE.nn))
## [1] "0.0444825853221814 0.0173965572123495"
```

Apparently the net is doing a better work than the linear model at predicting medv. However, this result depends on the train-test split performed above. So, we are going to perform a fast cross validation in order to be more confident about the results, using a for loop for the neural network and the cv.glm() function in the boot package for the linear model. Here is the 10 fold cross validated MSE for the linear model:

```
library(boot)
set.seed(200)
lm.fit <- glm(RECESSION~.,data=macro)
cv.glm(macro,lm.fit,K=10)$delta[1]
## [1] 0.05330968512
```

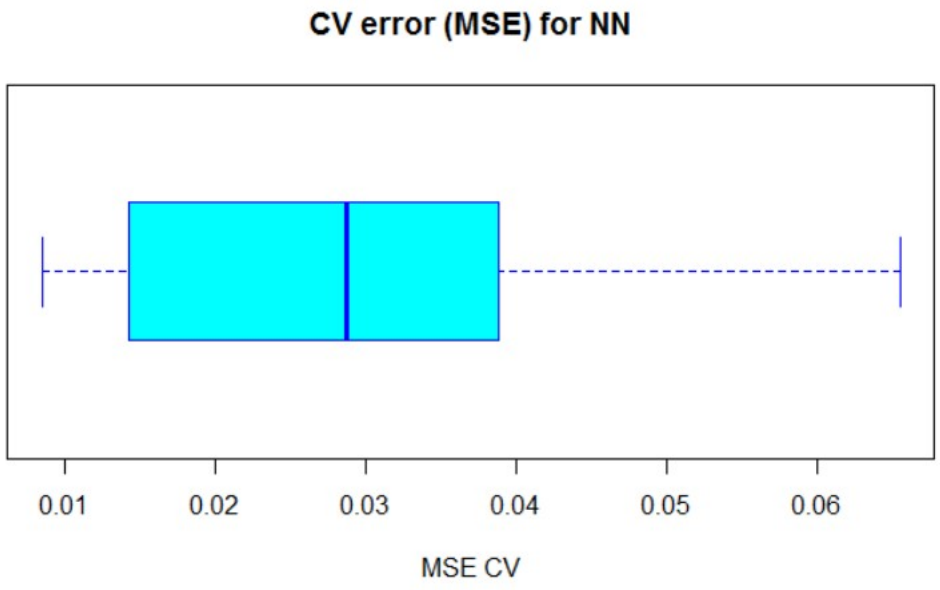
For the neural network, we are splitting the data in this way: 90% train set and 10% test set in a random way for 10 times. We are also initializing a progress bar using the plyr library because we want to keep an eye on the status of the process since the fitting of the neural network may take a while.

```
set.seed(450)
cv.error <- NULL
k <- 10
library(plyr)
pbar <- create_progress_bar('text')
pbar$init(k)
for(i in 1:k){
index <- sample(1:nrow(macro),round(0.9*nrow(macro)))
train.cv <- scaled[index,]
test.cv <- scaled[-index,]
nn <- neuralnet(f,data=train.cv,hidden=c(8,6),linear.output=F)
pr.nn <- compute(nn,test.cv[,1:21])
pr.nn <- pr.nn$net.result*(max(macro$RECESSION)-min(macro$RECESSION))
+min(macro$RECESSION)
test.cv.r <- (test.cv$RECESSION)*(max(macro$RECESSION)-min(macro$RECESSION))
```

```
+min(macro$RECESSION)
cv.error[i] <- sum((test.cv.r - pr.nn)^2)/nrow(test.cv)
pbar$step()
}
```

Now we calculate the average MSE and plot the results as a boxplot.

```
mean(cv.error)
## [1] 0.02930622944
boxplot(cv.error,xlab='MSE CV',col='cyan',
        border='blue',names='CV error (MSE)',
        main='CV error (MSE) for NN',horizontal=TRUE)
```



As you can see, the average MSE for the neural network (0.029) is lower than the one of the linear model although there seems to be a certain degree of variation in the MSEs of the cross validation. This may depend on the splitting of the data or the random initialization of the weights in the net.

Over-fitting issue

One of the problems that occur during neural network training is called overfitting. The error on the training set is driven to a very small value, but when new data is presented to the network the error is large. In this case, the network has memorized the training examples, but it has not learned to generalize to new situations. Here, the model yield a high precision (98%) with a very small error of 0.011. According to the literature, if the number of parameters in the network is much smaller than the total number of points in the training set, there is little or no chance of overfitting. Above, the NN model with 21 input neurons was applied on a training data set of 522 observations.

The weights of a neural network follow a multivariate normal distribution if the network is identified (White, 1989). A neural network is identified if it does not include irrelevant neurons neither in the input layer nor in the hidden layers. An irrelevant neuron in the input layer can be for instance a

covariate that has no effect or that is a linear combination of other included covariates. To check whether in the input layer there are covariates with no effect on the output, we plot the generalized weights for one specific covariate at time and the response variable using gwplot function in R.[4]

The distribution of the generalized weights suggests whether each one has an effect on response neuron or not. If the weights are nearly zero, they have no or little effect and if the variance of the generalized weights is overall greater than one, it has a non-linear effect. As shown in gw plots, most of the variables have non-linear effect on the output and their weight varies over time. In theory that happens since the state of economy is determined by many variables the importnace of which varies over time.[5] Appearantly, RPI, INDPRO, TB3MS, TB6MS, GS1 and GS5 have most of their generalized weights near zero which implies their little impact on the output for most of the time. In the next section, we exclude these variables from the data set and run the model on the narrow data set to see whether we can overcome the possibility of overfitting issue.

```
nn_narrow <- neuralnet(RECESSION ~ CLAIMSx + UNRATE + HOUST + M2REAL + GS10 +
TB3SMFFM + TB6SMFFM + T1YFFM + T5YFFM + T10YFFM + CPIAUCSL + S.P.500 + S.P..indust
+ S.P.div.yield,data=train.cv,hidden= c(6,3),linear.output=F)
head(nn_narrow$result.matrix, 3)
                                  1
error                 0.019014638727
reached.threshold     0.008942599148
steps              3988.000000000000
```

The error term is again low which indicates that the first model with the larger data set is less likely to be overfitted and the neural network has well recognized the underlying pattern.

Results

This paper involves finding how practically useful and applicable neural network resilient backpropagation algorithm is in the “real world.” This involves empirically validating the sensitivity, recall, precision, scalability of this algorithm with realistic, noisy and non-ideal data set. The use of large data sets in macroeconomic forecasting has been widely adopted in the last few decades. However, there are a few studies that apply neural networks algorithms to predict recessions.

Through this paper, the comparison of the error terms show how well neural networks help predict the economic recession in the US compared to the conventional linear regression models when we use a large set of economic indicators. Because there is little prior knowledge about the true underlying function that relates financial, economic and composite indicators to predict recession, and the relative importance of these economic indicators may change over time, the neural networks are expected to yield appropriate results in this context.

In fact, an advantage of neural networks is their ability to adapt to changing market conditions through periodic retaining. Moreover, it can detect nonlinear relationship between the dependent and independent variables. One can efficiently train large data sets using the parallel architecture and it is a nonparametric model so that one can eliminate errors in the estimation of parameters.

4. All plots are depicted in the Appendix

5. Friedman (1968) argued that monetary policy may have an asymmetric effect on real economic activity. Tightening monetary conditions slows output growth to a greater degree than an equivalent expansion of monetary policy stimulates it. This also the case for stock market indicators which has expanded to a large extent compared to 1950s.

The main disadvantages of neural network are that it often converges to the local minimum rather than the global minimum. Also, it might over-fit when the training process goes on for too long. To see whether the model is overfitted, we set up a narrow data set including fourteen variables based on the interpretation of estimated generalized weights. Since the error term of applying neural network algorithm on the narrow data set is also low, the first model with the large set of indicators is less likely to be overfitted.

Once deployed, a neural network's performance will degrade over time unless retraining takes place. However, even with periodic retraining, there's no guarantee that network performance can be maintained as the independent variables selected may have become less important. Thus, the neural network needs to be retrained every time that new economic indicators are released.

Although the results are interesting, the present paper is still limited in many ways and can be extended in future work. In fact, we need to predict the future values of economic indicators and use the model to forecast future recessions. The problem with economic data is their late release and revisions, so we need to forecast their future values.

Discussion

In spite of their numerous strengths, neural nets suffer from a number of weaknesses that researchers should keep in mind.

Sample size: Neural networks, more than linear models, need larger samples in order to be estimated properly. This is due to the large number of parameters introduced in such models that link the inputs to the hidden neurons, which are then linked to the output variable. As the data available for the training and testing of the model increase, one would then expect the marginal gains in forecast accuracy over linear models to increase. There is no rule of thumb for the "optimal" sample size for which one can expect neural nets to improve noticeably over linear models. This explains why, for example, there have been few macroeconomic applications of neural networks thus far in the literature. Forty years of quarterly data are usually judged insufficient to learn the relationships between the input and output variables properly. As such, neural networks should noticeably outperform linear models in forecasts of higher-frequency variables.

Lack of economic structure: Due to the black box nature of neural networks, users of forecasts may feel some uneasiness if they are unable to give proper economic interpretation to the estimated relationships. At the same time, it is difficult to determine which of the explanatory variables are driving the bulk of the forecasts, as comparative statics are difficult to perform.

References

- Angstenberger, J. 1996. “Prediction of the S&P 500 Index with Neural Networks.” In Neural Networks and Their Applications, edited by J. G. Taylor, 143–152. Chichester: Wiley and Sons.

- Bailey, D. L. and D. M. Thompson. 1990. “Developing Neural Network Applications.” AI Expert (September): 34–41.

- Bishop. Neural networks for pattern recognition. Oxford University Press, New York, 1995.

- Campbell, J. Y., A. W. Lo, and A. C. MacKinlay. 1997. The Econometrics of Financial Markets. Princeton: Princeton University Press.

- Cover, J. P. (1992). Asymmetric effects of positive and negative money-supply shocks. The Quarterly Journal of Economics, 107(4), 1261-1282.

- Fernandez-Rodrıguez, F., Gonzalez-Martel, C., & Sosvilla-Rivero, S. (2000). On the profitability of technical trading rules based on artificial neural networks:: Evidence from the Madrid stock market. Economics letters, 69(1), 89-94.

- Fritsch and F. Günther. neuralnet: Training of Neural Networks. R Foundation for Statistical Computing, 2008. R package version 1.2.

- Hutchinson, J., A. Lo, and T. Poggio. 1994. “A Nonparametric Approach to the Pricing and Hedging of Derivative Securities Via Learning Networks.” Journal of Finance 49: 851–99.

- Kohzadi, N., M. S. Boyd, I. Kaastra, B. S. Kermanshahi, and D. Scuse. 1995. “Neural Networks for Forecasting: An Introduction.” Canadian Journal of Agricultural Economics 43: 463–74.

- Kuan, C.-M. and H. White. 1994. “Artificial Neural Networks: An Econometric Perspective.” Econometric Reviews 13: 1–91.

- Kumar and D. Zhang. Personal recognition using hand shape and texture. IEEE Transactions on Image Processing, 15:2454–2461, 2006.

- McCullagh and J. Nelder. Generalized Linear Models. Chapman and Hall, London, 1983.

- Nyman, R., & Ormerod, P. (2017). Predicting Economic Recessions Using Machine Learning Algorithms. arXiv preprint arXiv:1701.01428.

- Refenes, A. P., & White, H. (1998). Preface. Journal of Forecasting, 17(5-6), 347-347.

- Rhee, W., & Rich, R. W. (1995). Inflation and the asymmetric effects of money on output fluctuations. Journal of Macroeconomics, 17(4), 683-702.

- Riedmiller, M., & Braun, H. (1993). A direct adaptive method for faster backpropagation learning: The RPROP algorithm. In Neural Networks, 1993., IEEE International Conference on (pp. 586-591). IEEE.

- Rocha, P. Cortez, and J. Neves. Evolutionary neural network learning. Lecture Notes in Computer Science, 2902:24–28, 2003.

- Schiffmann, M. Joost, and R. Werner. Optimization of the backpropagation algorithm for training multilayer perceptrons. Technical report, University of Koblenz, Insitute of Physics, 1994.

- White. Learning in artificial neural networks: a statistical perspective. Neural Computation, 1:425–464, 1989.

- Zhang, Yu, Wang and Xie, (2010). International Journal of Information Technology & Decision Making Vol. 6, No. 1 (2007) 113–140

Appendix

Table 1 – FRED-MD Data Description

FRED	Description	FRED	Description
RPI	Real Personal Income	T10YFFM	10-Year Treasury C Minus FEDFUNDS
INDPRO	IP Index	TB3MS	3-Month Treasury Bil
CLAIMSx	Initial Claims	TB6MS	6-Month Treasury Bill
UNRATE	Civilian Unemployment Rate	GS1	1-Year Treasury Rate
HOUST	Housing Starts: Total New Privately Owned	GS5	5-Year Treasury Rate
M2SL	M2 Money Stock	GS10	10-Year Treasury Rate
M2REAL	Real M2 Money Stock	CPIAUCSL	CPI, All Items
TB3SMFFM	3-Month Treasury C Minus FEDFUNDS	S&P 500	S&P's Common Stock Price Index: Composite
TB6SMFFM	6-Month Treasury C Minus FEDFUNDS	S&P: indust	S&P's Common Stock Price Index: Industrial
T1YFFM	1-Year Treasury C Minus FEDFUNDS	S&P div yield	S&P's Composite Common Stock: Dividend Yield
T5YFFM	5-Year Treasury C Minus FEDFUNDS		

Summary Statistics of the Data set

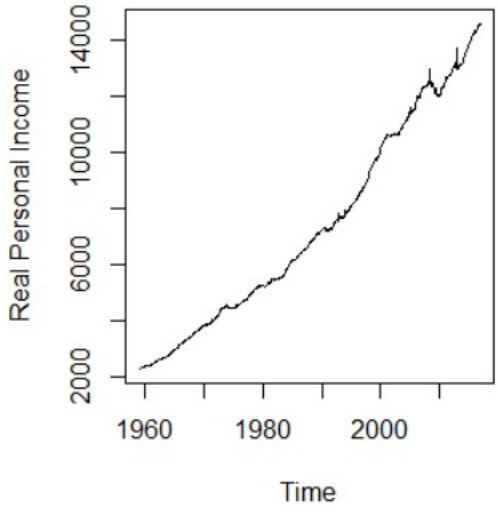
```
   sasdate          RPI                INDPRO            CLAIMSx            UNRATE
1/1/1959:  1   Min.   : 2289.800   Min.   : 22.83310   Min.   :176263.4   Min.   : 3.400000
1/1/1960:  1   1st Qu.: 4441.775   1st Qu.: 43.45970   1st Qu.:293743.7   1st Qu.: 5.000000
1/1/1961:  1   Median : 6799.100   Median : 62.69520   Median :338408.2   Median : 5.700000
1/1/1962:  1   Mean   : 7402.874   Mean   : 65.44817   Mean   :347433.5   Mean   : 6.059339
1/1/1963:  1   3rd Qu.:10653.925   3rd Qu.: 93.80230   3rd Qu.:392556.6   3rd Qu.: 7.100000
1/1/1964:  1   Max.   :14590.300   Max.   :106.68680   Max.   :663892.8   Max.   :10.800000
(Other) :690
  HOUST              M2SL              M2REAL             TB3MS
Min.   : 478.000   Min.   :  286.600   Min.   : 987.900   Min.   : 0.010000
1st Qu.:1185.250   1st Qu.:  835.700   1st Qu.:1799.150   1st Qu.: 2.730000
Median :1471.500   Median : 2836.900   Median :2358.400   Median : 4.725000
Mean   :1438.774   Mean   : 3680.221   Mean   :2488.465   Mean   : 4.664885
3rd Qu.:1676.000   3rd Qu.: 5530.475   3rd Qu.:3077.575   3rd Qu.: 6.162500
Max.   :2494.000   Max.   :13185.100   Max.   :5430.000   Max.   :16.300000

  TB6MS              GS1                GS5                GS10
Min.   : 0.040000   Min.   : 0.100000   Min.   : 0.620000   Min.   : 1.500000
1st Qu.: 2.887500   1st Qu.: 3.067500   1st Qu.: 3.880000   1st Qu.: 4.150000
Median : 4.880000   Median : 5.135000   Median : 5.690000   Median : 5.875000
Mean   : 4.808534   Mean   : 5.191293   Mean   : 5.881552   Mean   : 6.205129
3rd Qu.: 6.437500   3rd Qu.: 6.982500   3rd Qu.: 7.615000   3rd Qu.: 7.782500
Max.   :15.520000   Max.   :16.720000   Max.   :15.930000   Max.   :15.320000

 TB3SMFFM           TB6SMFFM            T1YFFM              T5YFFM
Min.   :-5.370000   Min.   :-5.0100000   Min.   :-5.00000000   Min.   :-6.3100000
1st Qu.:-0.690000   1st Qu.:-0.5500000   1st Qu.:-0.13250000   1st Qu.: 0.1275000
Median :-0.250000   Median :-0.1200000   Median : 0.14000000   Median : 0.9200000
Mean   :-0.472227   Mean   :-0.3285776   Mean   : 0.05418103   Mean   : 0.7444397
3rd Qu.:-0.057500   3rd Qu.: 0.0600000   3rd Qu.: 0.44000000   3rd Qu.: 1.6425000
Max.   : 1.070000   Max.   : 1.2800000   Max.   : 1.75000000   Max.   : 3.1600000

 T10YFFM            CPIAUCSL          S.P.500          S.P..indust
```

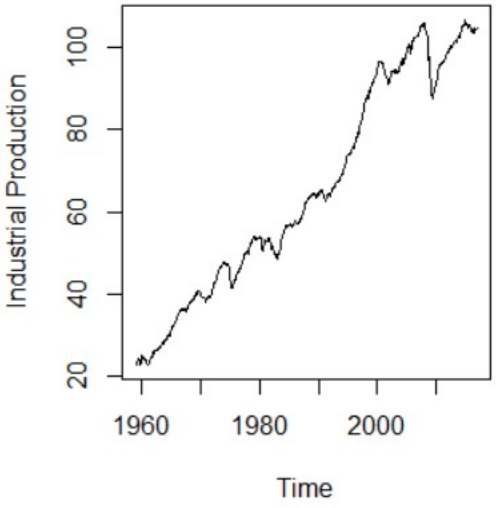
```
Min.   :-6.510000   Min.   : 28.9700   Min.   :  53.7300   Min.   :  56.9000
1st Qu.: 0.190000   1st Qu.: 44.2000   1st Qu.:  97.7275   1st Qu.: 107.9250
Median : 1.240000   Median :115.8000   Median : 270.8500   Median : 312.3300
Mean   : 1.068017   Mean   :120.2483   Mean   : 581.5567   Mean   : 711.2777
3rd Qu.: 2.242500   3rd Qu.:179.7000   3rd Qu.:1111.9975   3rd Qu.:1305.3725
Max.   : 3.850000   Max.   :242.8210   Max.   :2246.6300   Max.   :2962.3800

S.P.div.yield        RECESSION
Min.   :1.108074   Min.   :0.0000000
1st Qu.:2.013717   1st Qu.:0.0000000
Median :2.976297   Median :0.0000000
Mean   :2.985964   Mean   :0.1336207
3rd Qu.:3.557893   3rd Qu.:0.0000000
Max.   :6.237048   Max.   :1.0000000
```

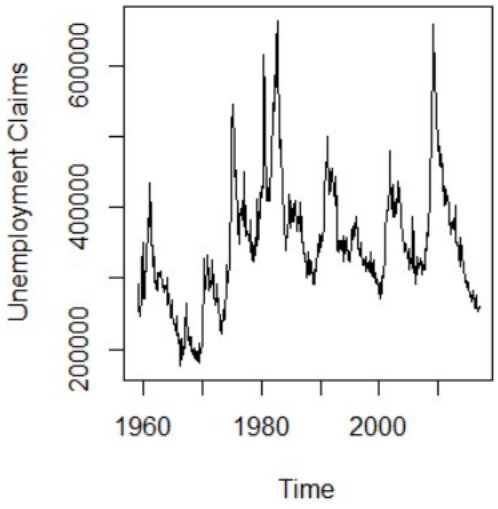
```
# To check if there's any missing value
apply(macro_data,2,function(x) sum(is.na(x)))
##            sasdate                RPI             INDPRO            CLAIMSx             UNRATE
##                  0                  0                  0                  0                  0
##              HOUST               M2SL            M2REAL              TB3MS              TB6MS
##                  0                  0                  0                  0                  0
##                GS1                GS5               GS10           TB3SMFFM           TB6SMFFM
##                  0                  0                  0                  0                  0
##             T1YFFM             T5YFFM            T10YFFM           CPIAUCSL             S.P.500
##                  0                  0                  0                  0                  0
##            S.P..indust                S.P.div.yield                  RECESSION
##                    0                    0                                  0
```

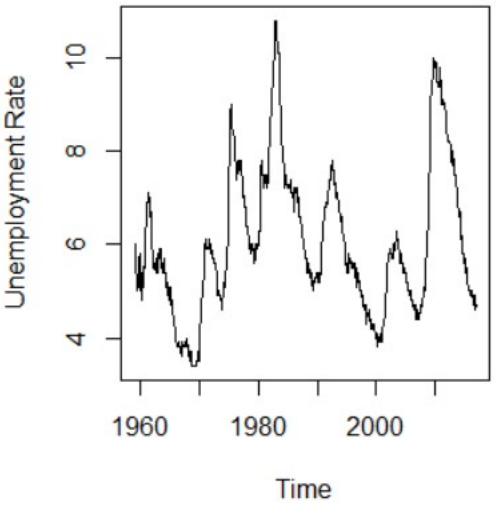
Timeseries Plots

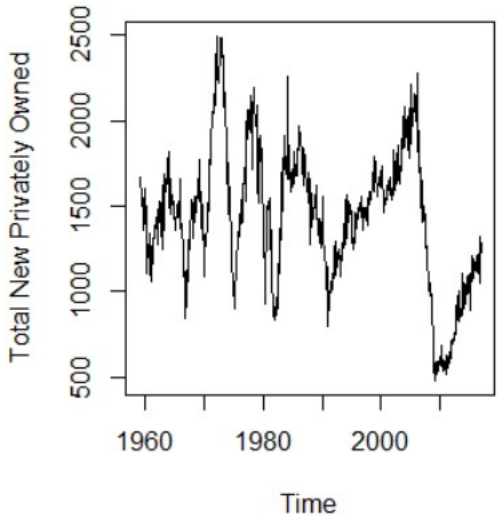
```
# Turn data into timeseries
macro_ts <- ts(macro, start = c(1959,1), end = c(2016, 12), frequency = 12  )
head(macro_ts)

plot.ts(macro_ts[,1], ylab = "Real Personal Income")
plot.ts(macro_ts[,2], ylab = "Industrial Production")
plot.ts(macro_ts[,3], ylab = "Unemployment Claims")
plot.ts(macro_ts[,4], ylab = "Unemployment Rate")
plot.ts(macro_ts[,5], ylab = "Total New Privately Owned")
plot.ts(macro_ts[,6], ylab = "M2 Money Stock")
plot.ts(macro_ts[,7], ylab = "Real M2 Money Stock")
plot.ts(macro_ts[,8], ylab = "3-Month Treasury Bill")
plot.ts(macro_ts[,9], ylab = "6-Month Treasury Bill")
plot.ts(macro_ts[,10], ylab = "1-Year Treasury Rate")
plot.ts(macro_ts[,11], ylab = "5-Year Treasury Rate")
plot.ts(macro_ts[,12], ylab = "10-Year Treasury Rate")
plot.ts(macro_ts[,13], ylab = "3-Month Treasury C Minus FEDFUNDS")
plot.ts(macro_ts[,14], ylab = "6-Month Treasury C Minus FEDFUNDS")
plot.ts(macro_ts[,15], ylab = "1-Year Treasury C Minus FEDFUNDS")
plot.ts(macro_ts[,16], ylab = "5-Year Treasury C Minus FEDFUNDS")
plot.ts(macro_ts[,17], ylab = "10-Year Treasury C Minus FEDFUNDS")
plot.ts(macro_ts[,18], ylab = "CPI: All items")
plot.ts(macro_ts[,19], ylab = "S&Pís Common Stock Price Index: Composite")
plot.ts(macro_ts[,20], ylab = "S&Pís Common Stock Price Index: Industrials")
plot.ts(macro_ts[,21], ylab = "S&Pís Composite Common Stock: Dividend Yield")
```

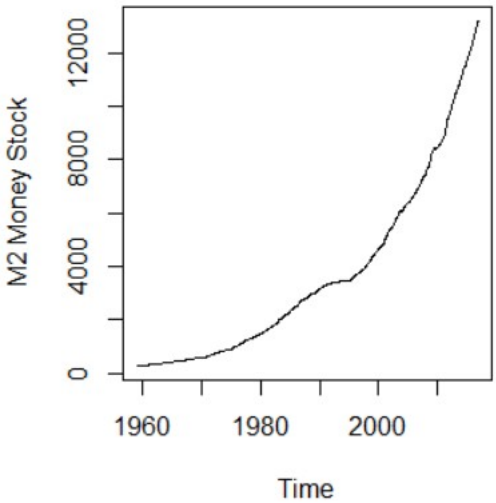

Real Personal Income
2000
6000
10000
14000
1960
1980
2000
Time

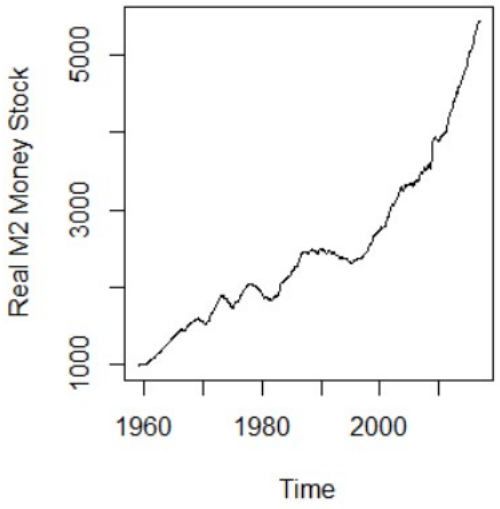

Industrial Production
20
40
60
80
100
1960
1980
2000
Time

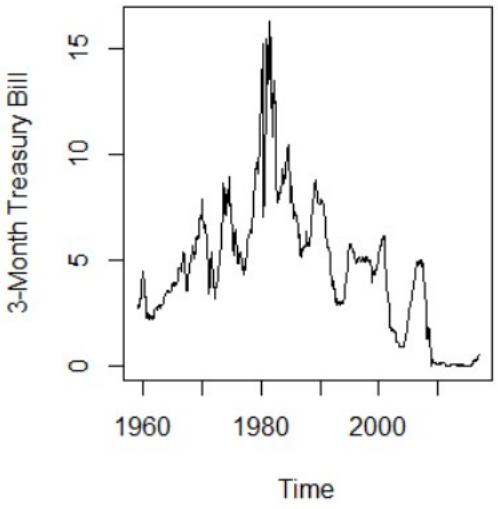

Unemployment Claims
200000
400000
600000
1960
1980
2000
Time

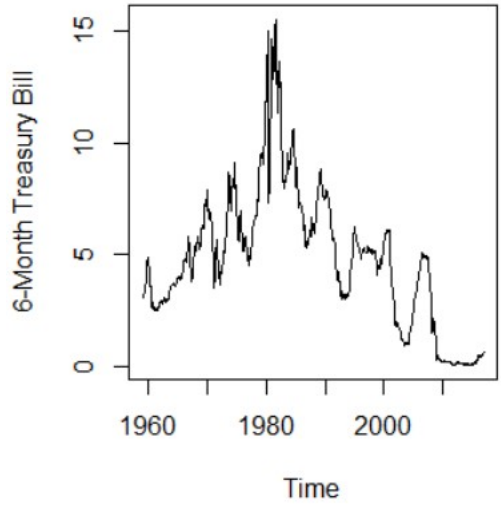

Unemployment Rate
4
6
8
10
1960
1980
2000
Time

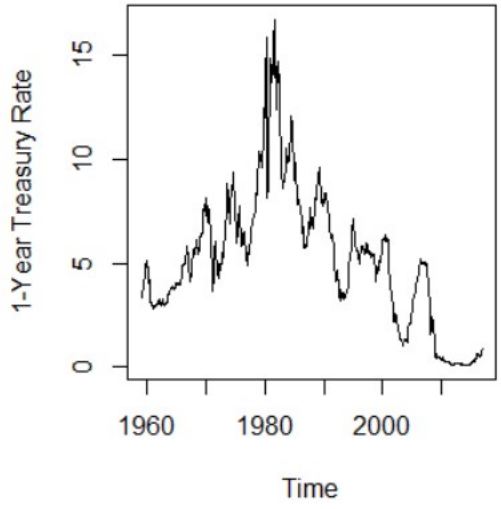

Total New Privately Owned
500
1000
1500
2000
2500
1960
1980
2000
Time

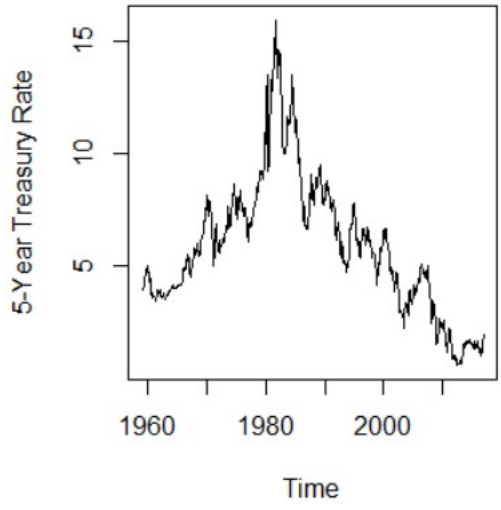

M2 Money Stock
0
4000
8000
12000
1960
1980
2000
Time

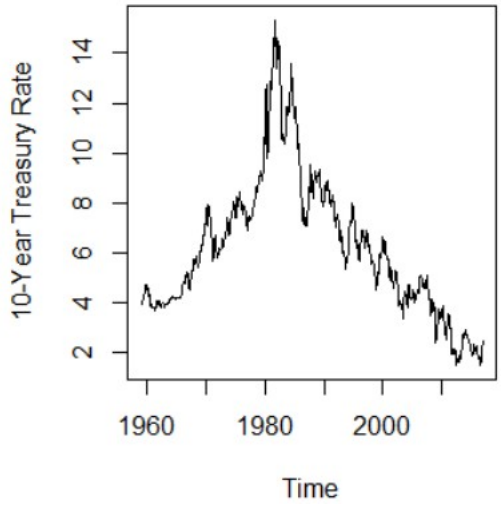

Real M2 Money Stock
1000
3000
5000
1960
1980
2000
Time

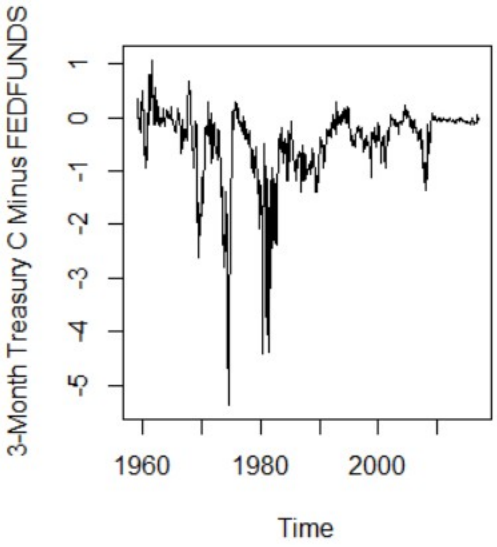

3-Month Treasury Bill
0
5
10
15
1960
1980
2000
Time

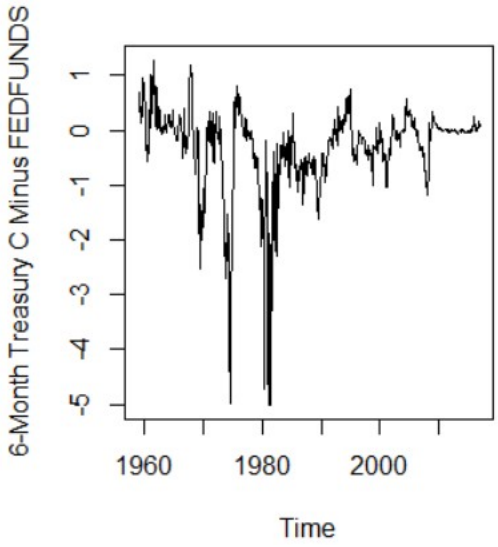

6-Month Treasury Bill
0
5
10
15
1960
1980
2000
Time

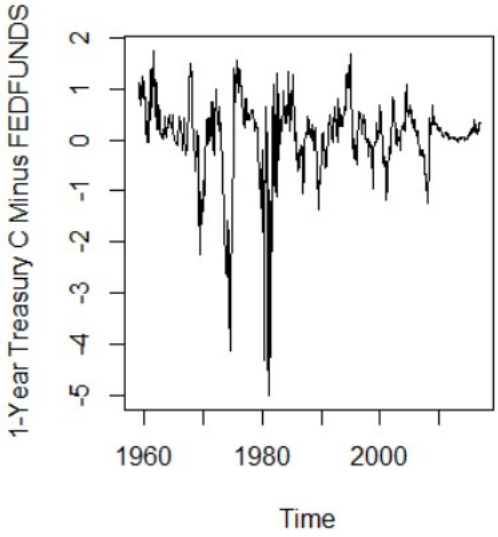

1-Year Treasury Rate
0
5
10
15
1960
1980
2000
Time

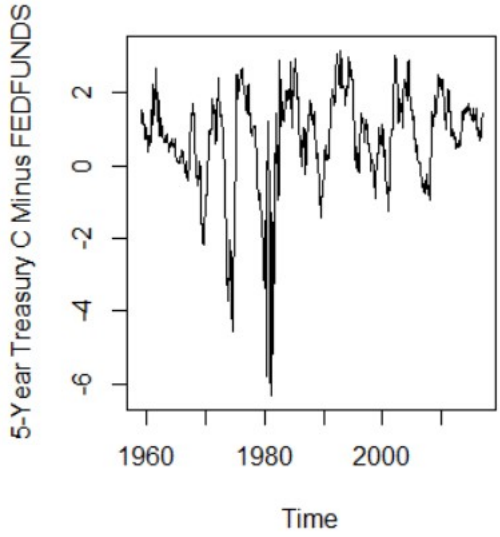

5-Year Treasury Rate
5
10
15
1960
1980
2000
Time

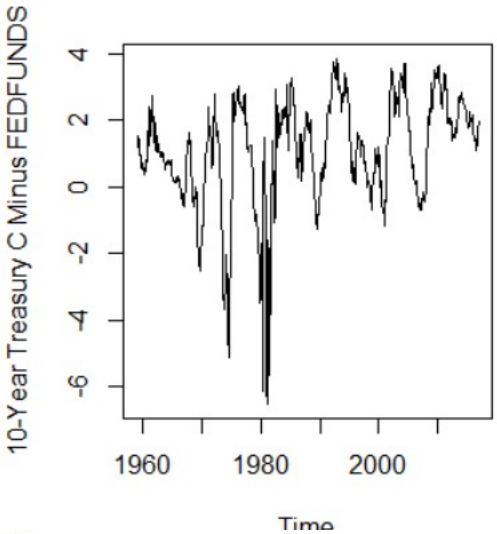

10-Year Treasury Rate
2
4
6
8
10
12
14
1960
1980
2000
Time

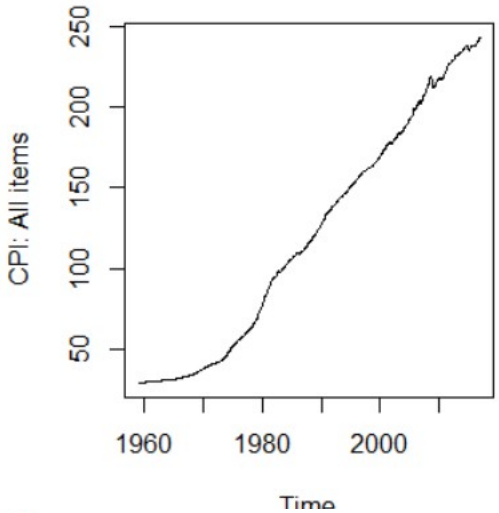

3-Month Treasury C Minus FEDFUNDS
1
0
-1
-2
-3
-4
-5
1960
1980
2000
Time

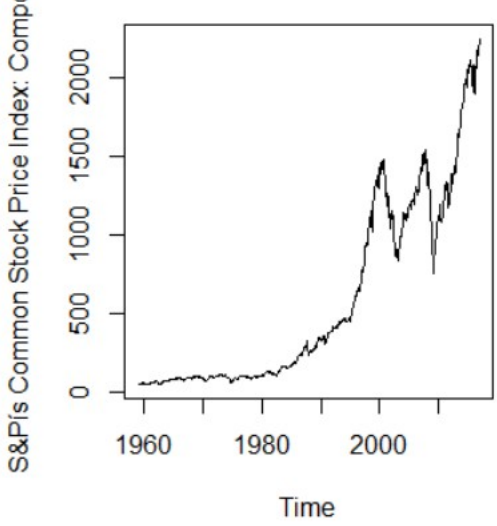

6-Month Treasury C Minus FEDFUNDS
1
0
-1
-2
-3
-4
-5
1960
1980
2000
Time

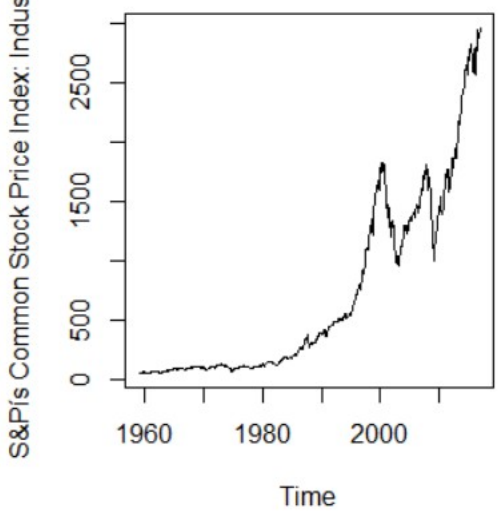

1-Year Treasury C Minus FEDFUNDS
2
1
0
-1
-2
-3
-4
-5
1960
1980
2000
Time

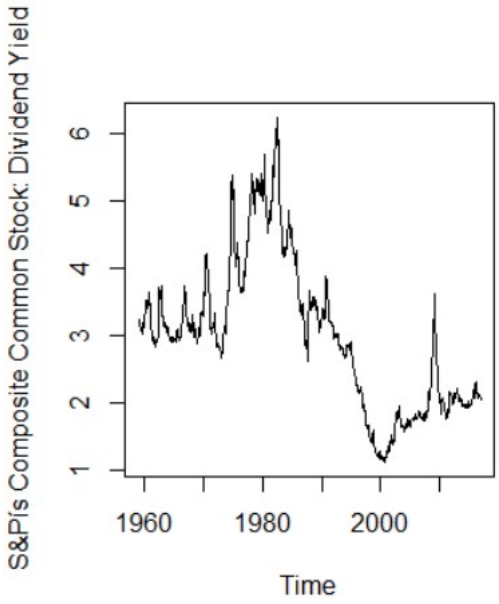

5-Year Treasury C Minus FEDFUNDS
2
0
-2
-4
-6
1960
1980
2000
Time


10-Year Treasury C Minus FEDFUNDS
4
2
0
-2
-4
-6
1960
1980
2000
Time


CPI: All items
250
200
150
100
50
1960
1980
2000
Time


S&Pís Common Stock Price Index: Composite
2000
1500
1000
500
0
1960
1980
2000
Time


S&Pís Common Stock Price Index: Industrials
2500
1500
500
0
1960
1980
2000
Time


S&Pís Composite Common Stock: Dividend Yield
6
5
4
3
2
1
1960
1980
2000
Time

Results of Linear Regression

```
summary(lm.fit)
##
## Call:
## glm(formula = RECESSION ~ ., data = train)
##
## Deviance Residuals:
##      Min        1Q    Median        3Q       Max
## -0.74959  -0.12894  -0.02634   0.09006   0.76777
##
## Coefficients: (4 not defined because of singularities)
##                 Estimate Std. Error t value Pr(>|t|)
## (Intercept)    4.177e-01  1.443e-01   2.896 0.003949 **
## RPI            4.974e-04  9.018e-05   5.515 5.57e-08 ***
## INDPRO        -1.559e-02  8.489e-03  -1.837 0.066866 .
## CLAIMSx        2.960e-06  3.299e-07   8.971  < 2e-16 ***
## UNRATE        -1.633e-01  1.708e-02  -9.562  < 2e-16 ***
## HOUST         -1.698e-04  4.374e-05  -3.883 0.000117 ***
## M2SL           1.521e-04  7.358e-05   2.067 0.039247 *
## M2REAL        -8.701e-04  1.516e-04  -5.738 1.65e-08 ***
## TB3MS         -1.546e-01  7.494e-02  -2.064 0.039558 *
## TB6MS          2.675e-01  1.397e-01   1.915 0.056027 .
## GS1           -2.533e-01  9.863e-02  -2.569 0.010497 *
## GS5            1.506e-01  1.036e-01   1.453 0.146739
## GS10          -9.967e-05  8.498e-02  -0.001 0.999065
## TB3SMFFM      -1.539e-01  2.036e-02  -7.560 1.92e-13 ***
## CPIAUCSL      -1.327e-02  2.024e-03  -6.559 1.34e-10 ***
## S.P.500       -1.172e-03  5.530e-04  -2.119 0.034540 *
## S.P..indust    8.087e-04  3.946e-04   2.050 0.040922 *
## S.P.div.yield  1.208e-01  3.422e-02   3.531 0.000451 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for gaussian family taken to be 0.05299385)
##
##     Null deviance: 63.510  on 521  degrees of freedom
## Residual deviance: 26.709  on 504  degrees of freedom
## AIC: -32.362
##
## Number of Fisher Scoring iterations: 2
```

```
head(out)
```

```
       RPI         INDPRO       CLAIMSx         UNRATE         HOUST             M2SL
[1,] 0.422145441242 0.49496921424 0.5460281264 0.54054054054 0.3829365079 0.2413458929333
[2,] 0.008373643348 0.02466319316 0.1933425996 0.18918918919 0.5084325397 0.0009225878978
[3,] 0.666989146783 0.87585640228 0.3252818427 0.06756756757 0.5322420635 0.3546071248595
[4,] 0.450932888907 0.56348139677 0.3267533421 0.41891891892 0.5386904762 0.2475016474784
[5,] 0.196658672412 0.28010570792 0.4733230210 0.56756756757 0.6160714286 0.0638989029732
[6,] 0.027372871022 0.04299392871 0.2683220248 0.31081081081 0.5153769841 0.0051323797341
     M2REAL         TB3MS         TB6MS          GS1          GS5          GS10          TB3SMFFM
[1,] 0.329776457081 0.2351135666 0.2512919897 0.2521058965 0.3892880470 0.4225759768 0.7996894410
[2,] 0.006100718129 0.2424800491 0.2751937984 0.2743682310 0.2658393207 0.2163531114 0.8322981366
[3,] 0.405731523379 0.3781461019 0.3888888889 0.3604091456 0.3318092750 0.3053545586 0.7810559006
[4,] 0.310033542694 0.2142418662 0.2416020672 0.2539109507 0.3474853037 0.3603473227 0.8586956522
[5,] 0.211746696382 0.3112338858 0.3397932817 0.3453670277 0.4252122796 0.4406657019 0.8074534161
[6,] 0.039891042525 0.1724984653 0.1905684755 0.1865222623 0.2018288700 0.1794500724 0.8167701863
    TB6SMFFM        T1YFFM        T5YFFM       T10YFFM        CPIAUCSL          S.P.500
[1,] 0.7758346582 0.7748148148 0.9324181626 0.9449806950 0.512646655849 0.1636326325870
[2,] 0.8489666137 0.8429629630 0.7423442450 0.6785714286 0.002057507330 0.0009348351498
[3,] 0.7249602544 0.6785185185 0.5807814150 0.5521235521 0.679117703448 0.6039080669433
[4,] 0.8664546900 0.8859259259 0.9408658923 0.9314671815 0.552393956540 0.1870035113320
[5,] 0.8044515103 0.8281481481 0.8648363252 0.8542471042 0.133878261032 0.0236080076611
[6,] 0.8060413355 0.7807407407 0.7486800422 0.7297297297 0.006125760459 0.0021843221305
    S.P..indust S.P.div.yield                    nn-output
[1,] 0.1493694673513 0.36429561678 0.000000000624057346214704 5
[2,] 0.0009292784669 0.44923891995 0.000000001621403452242093 8
[3,] 0.5438860360422 0.01448269374 0.000003448178517281265096 1
[4,] 0.1675489075815 0.31824537011 0.000000000000000424784447 6
[5,] 0.0210980629707 0.49546267873 0.000000000003830968697674 3
[6,] 0.0015109379517 0.47472821385 0.000268114960260810335831 3
```

```
head(nn$generalized.weights[[1]])
```

```
 [,1]           [,2]          [,3]          [,4]           [,5]           [,6]
398  1.718703742  -1.4779012201  75.41701833 -58.61968708  -9.04917525376 -10.43586527071
14   1.226228747  -6.9680850493  64.85156302 -36.39006253  -5.93671634071 -21.48707568175
503  3.824686971  -2.4899680000  52.50609393 -28.92143420  -5.67259044526   2.03050114868
423  2.456864279  -0.8660374601  33.95673181 -18.55795249  -4.89752833888   0.06119154348
213 -3.091266108 -11.4401025314  68.76146916 -47.24975941 -17.33080392294 -79.45715777158
44  20.979016156  -3.8636804642 111.55427082 -11.26012920   0.02747162328 -26.82047691287
      [,7]           [,8]           [,9]          [,10]         [,11]         [,12]
398 -12.631271066 -10.167273034 -6.8431466023  26.490330012 -4.645525231 20.724325967
14  -15.413478439  -3.899375116 -0.3786279827  15.997075910 -2.772091477 14.929267794
503 -10.560318156  -6.519819031 -6.1580063309   8.117455215 -2.389546875  8.957279247
423  -6.538549045  -4.413583838 -1.3061043391   4.933453233 -1.563273880  6.007435850
213 -43.613778642  -7.405180629  0.4637064687  34.364053725 -0.832456536 11.832655653
44   35.962952798 -11.823399846 -6.7607728852 -11.048103616 -2.102636594 43.637627588
     [,13]          [,14]          [,15]         [,16]          [,17]          [,18]
398 -29.130443759 -29.879364188 -25.935553385 5.2660537488  5.4507418698 -18.138283685
14  -16.319612761 -18.443033303 -18.911508658 4.7216105635  2.9876424326 -21.944370417
503  -8.783521183 -13.339012465 -15.415885971 0.6772213472 -1.0542224543   6.203336979
423  -7.698449878  -9.158574218  -9.685998664 0.1915114925  0.1051416683   3.273085434
213 -24.443653507 -20.090166819 -22.497572215 9.3265835928 10.0143909177 -90.775272477
44  -24.678620495 -24.022406151 -30.274452676 2.2364533063  4.4876804818 -17.970084965
     [,19]          [,20]        [,21]
398 26.602865533   2.2694389624 39.28128899
14  12.092035517  -7.4265628277 24.85798785
503 14.266495183   3.4654585800 11.72984141
423 10.020798115  -0.4161912328 13.03965177
213  5.251830351 -21.4968782490 34.42647912
44   2.181618785 -22.9529735893 29.43075714
```

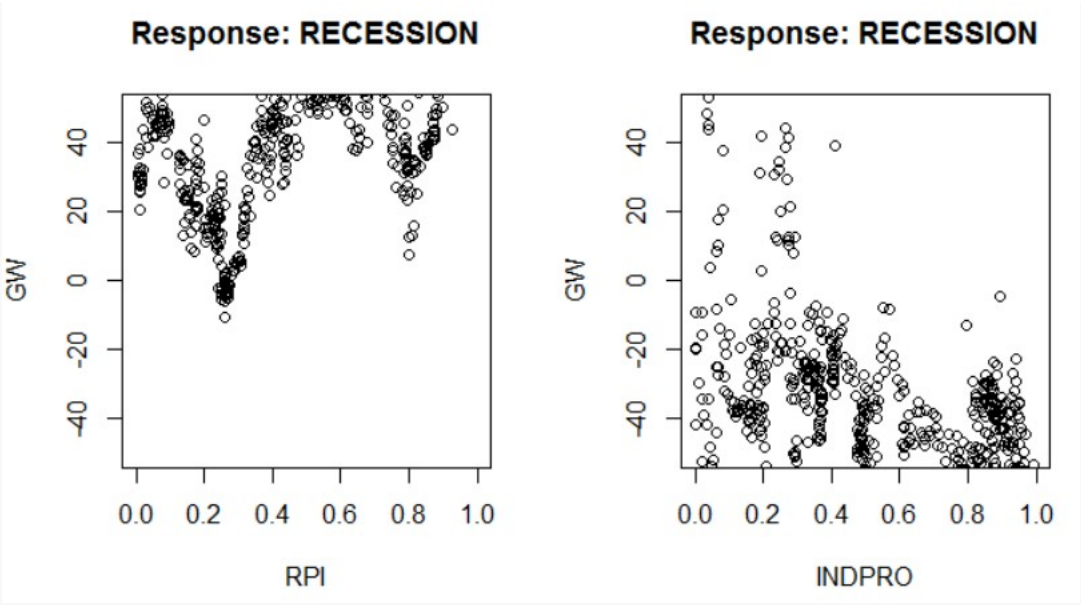
```
results
##     actual prediction
## 3        0          0
## 6        0          0
## 9        0          0
## 20       1          1
## 23       1          1
## 26       1          1
## 28       0          0
## 29       0          0
## 35       0          0
## 37       0          0
## 44       0          0
## 46       0          0
## 47       0          0
## 52       0          0
## 54       0          0
## 56       0          0
## 59       0          0
## 63       0          0
## 65       0          0
## 67       0          0
## 70       0          0
## 73       0          0
## 75       0          0
## 78       0          0
## 83       0          0
## 97       0          0
## 108      0          0
## 115      0          0
## 116      0          0
## 122      0          0
## 131      0          0
## 143      1          1
## 149      0          0
## 150      0          0
## 151      0          0
## 152      0          0
## 160      0          0
## 162      0          0
## 167      0          0
## 169      0          0
## 197      0          0
## 200      0          0
## 205      0          0
## 206      0          0
## 208      0          0
## 209      0          0
## 215      0          0
## 218      0          0
## 220      0          0
## 221      0          0
## 222      0          0
## 228      0          0
## 229      0          0
## 231      0          0
## 232      0          0
## 233      0          0
## 241      0          0
```

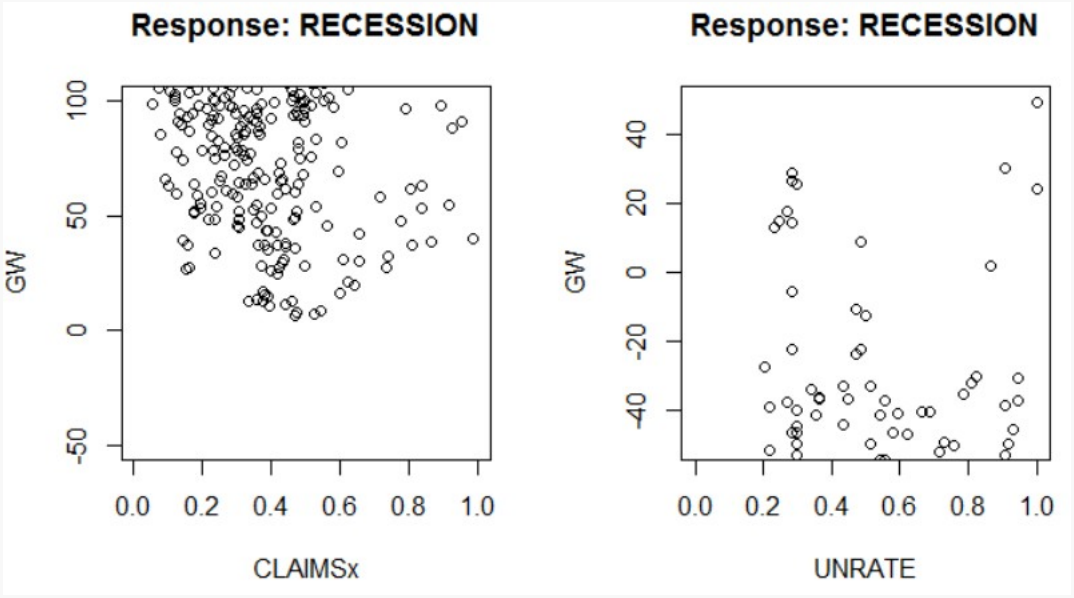
```
## 250      0           0
## 254      1           0
## 258      1           1
## 260      0           1
## 262      0           0
## 264      0           0
## 275      1           1
## 277      1           1
## 282      1           1
## 287      1           0
## 295      0           0
## 296      0           0
## 297      0           0
## 298      0           0
## 299      0           0
## 300      0           0
## 306      0           0
## 310      0           0
## 312      0           0
## 314      0           0
## 317      0           0
## 319      0           0
## 321      0           0
## 323      0           0
## 328      0           0
## 331      0           0
## 332      0           0
## 333      0           0
## 336      0           0
## 339      0           0
## 341      0           0
## 350      0           0
## 355      0           0
## 360      0           0
## 362      0           0
## 365      0           0
## 369      0           0
## 370      0           0
## 372      0           0
## 373      0           0
## 374      0           0
## 375      0           0
## 378      0           0
## 379      0           0
## 384      1           1
## 385      1           1
## 386      1           1
## 387      1           1
## 388      0           0
## 393      0           0
## 395      0           0
## 405      0           0
## 407      0           0
## 411      0           0
## 421      0           0
## 426      0           0
## 430      0           0
## 432      0           0
## 434      0           0
## 439      0           0
## 447      0           0
```

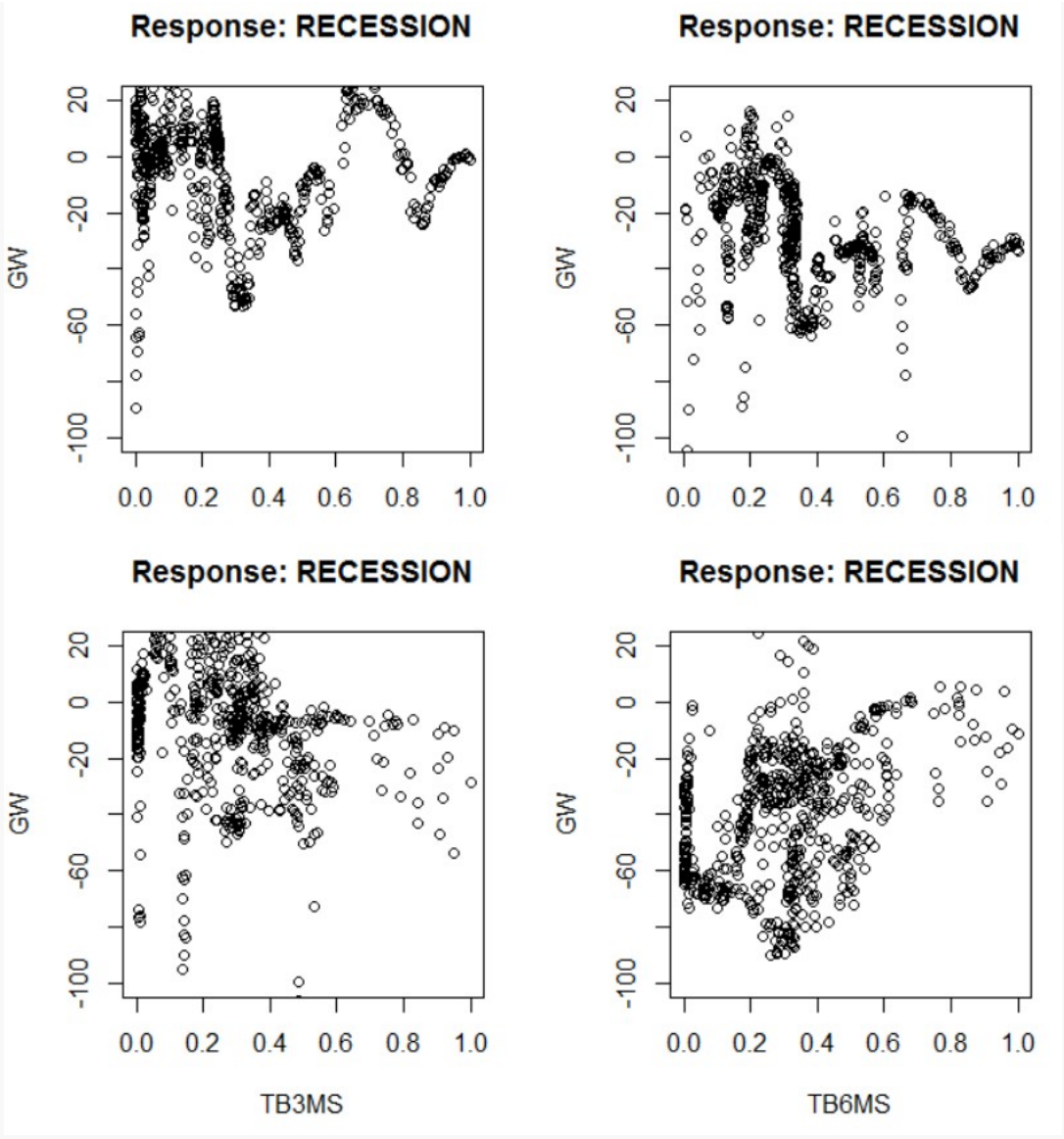
```
## 459      0           0
## 461      0           0
## 464      0           0
## 465      0           0
## 472      0           0
## 473      0           0
## 479      0           0
## 483      0           0
## 487      0           0
## 493      0           0
## 494      0           0
## 500      0           0
## 511      1           1
## 514      1           1
## 527      0           0
## 529      0           0
## 530      0           0
## 534      0           0
## 537      0           0
## 541      0           0
## 545      0           0
## 547      0           0
## 554      0           0
## 574      0           0
## 575      0           0
## 576      0           0
## 577      0           0
## 578      0           0
## 585      0           0
## 586      0           0
## 598      1           1
## 601      1           1
## 602      1           1
## 609      0           0
## 610      0           0
## 612      0           0
## 615      0           0
## 622      0           0
## 623      0           0
## 631      0           0
## 634      0           0
## 640      0           0
## 641      0           0
## 644      0           0
## 650      0           0
## 659      0           0
## 661      0           0
## 668      0           0
## 669      0           0
## 674      0           0
## 676      0           0
## 678      0           0
## 679      0           0
## 683      0           0
## 686      0           0
## 695      0           0
```

Plots of generalized weights

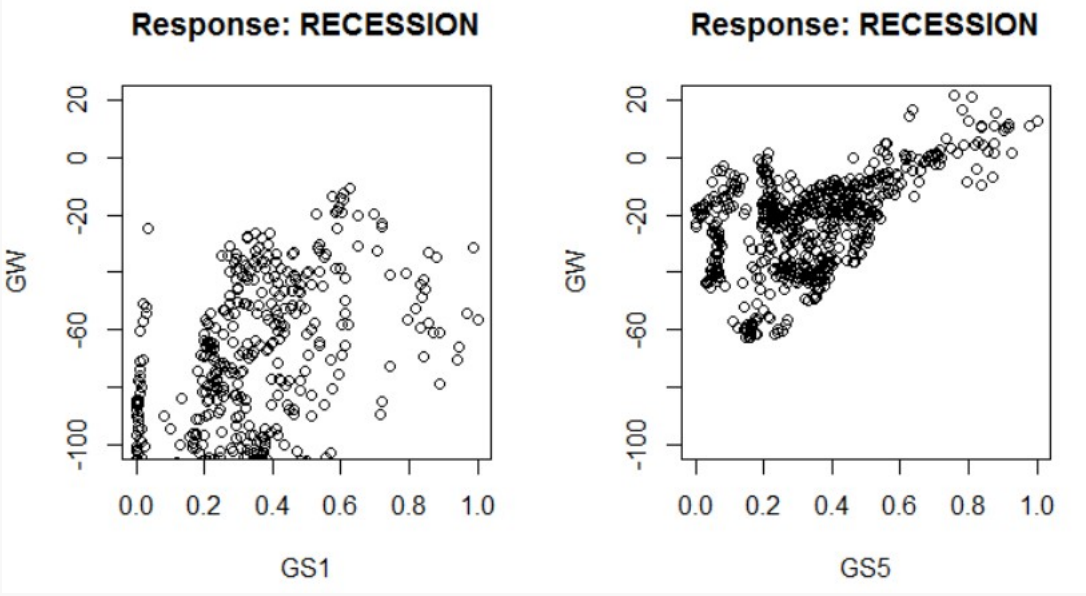
```
gwplot(nn,selected.covariate="RPI", min=-50, max=50)
gwplot(nn,selected.covariate="INDPRO", min=-50, max=50)
gwplot(nn,selected.covariate="INDPRO", min=-50, max=50)
gwplot(nn,selected.covariate="CLAIMSx", min=-50, max=100)
gwplot(nn,selected.covariate="UNRATE", min=-50, max=50)
gwplot(nn,selected.covariate="HOUST", min=-50, max=50)
gwplot(nn,selected.covariate="M2SL", min=-100, max=20)
gwplot(nn,selected.covariate="M2REAL", min=-100, max=20)
gwplot(nn,selected.covariate="TB3MS", min=-100, max=20)
gwplot(nn,selected.covariate="TB6MS", min=-100, max=20)
gwplot(nn,selected.covariate="GS1", min=-100, max=20)
gwplot(nn,selected.covariate="GS5", min=-100, max=20)
gwplot(nn,selected.covariate="GS10", min=-20, max=200)
gwplot(nn,selected.covariate="TB3SMFFM", min=-100, max=20)
gwplot(nn,selected.covariate="TB6SMFFM", min=-100, max=50)
gwplot(nn,selected.covariate="T5YFFM", min=-50, max=50)
gwplot(nn,selected.covariate="T10YFFM", min=-50, max=50)
gwplot(nn,selected.covariate="CPIAUCSL", min=-50, max=50)
gwplot(nn,selected.covariate="S.P.500", min=-100, max=100)
gwplot(nn,selected.covariate="S.P..indust", min=-50, max=50)
gwplot(nn,selected.covariate="S.P.div.yield", min=-50, max=100)
```


Response: RECESSION
GW
RPI
Response: RECESSION
GW
INDPRO

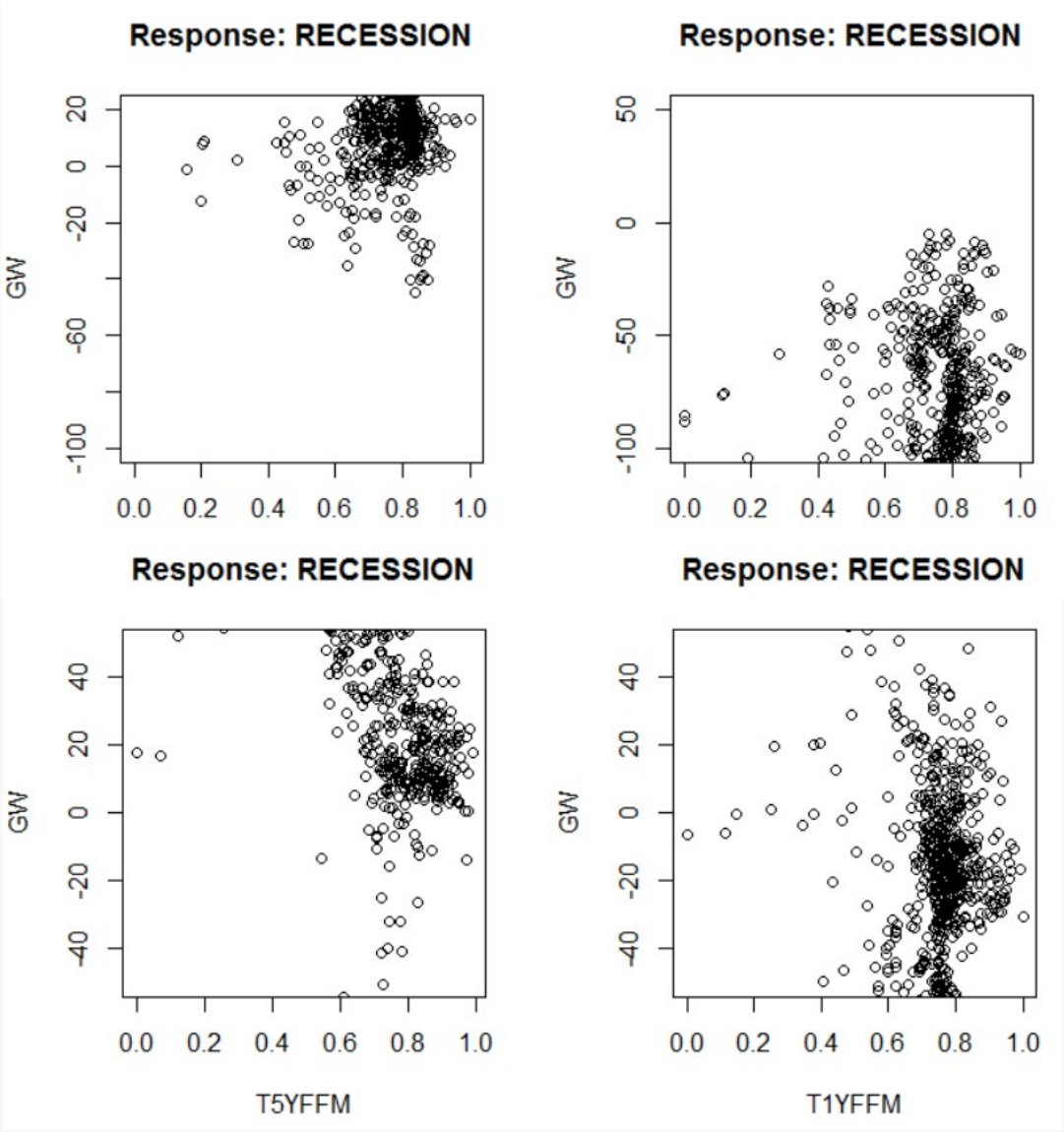

Response: RECESSION
GW
CLAIMSx
Response: RECESSION
GW
UNRATE

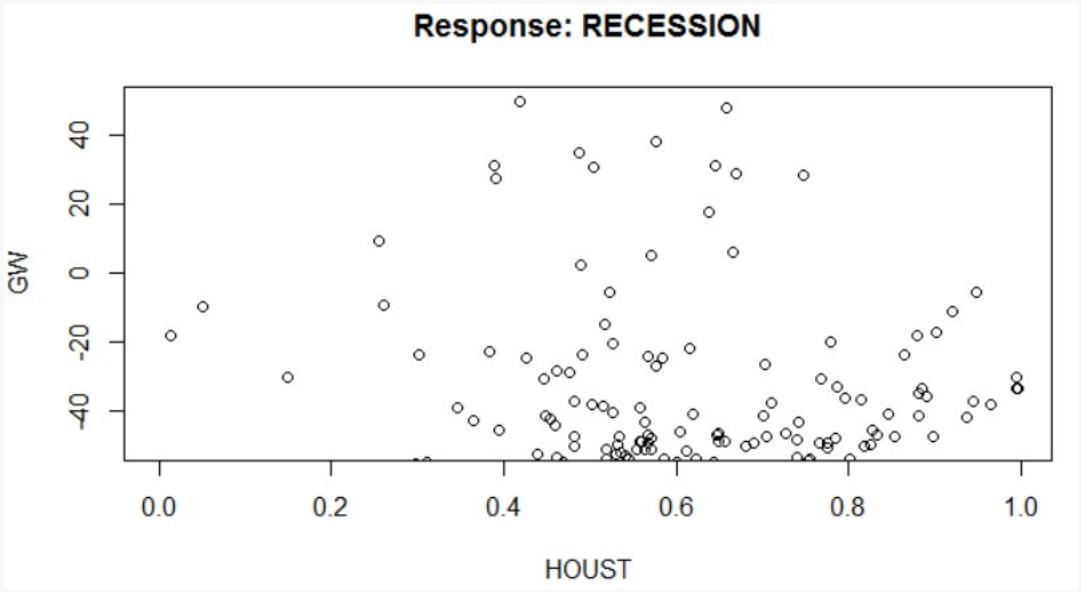

Response: RECESSION
GW
20
0
-20
-60
-100
0.0
0.2
0.4
0.6
0.8
1.0
TB3MS

Response: RECESSION
GW
20
0
-20
-60
-100
0.0
0.2
0.4
0.6
0.8
1.0
TB6MS


Response: RECESSION
GW
20
0
-20
-60
-100
0.0
0.2
0.4
0.6
0.8
1.0
GS1

Response: RECESSION
GW
20
0
-20
-60
-100
0.0
0.2
0.4
0.6
0.8
1.0
GS5


Response: RECESSION
GW
20
0
-20
-60
-100
0.0
0.2
0.4
0.6
0.8
1.0
Response: RECESSION
GW
50
0
-50
-100
0.0
0.2
0.4
0.6
0.8
1.0
Response: RECESSION
GW
40
20
0
-20
-40
0.0
0.2
0.4
0.6
0.8
1.0
T5YFFM
Response: RECESSION
GW
40
20
0
-20
-40
0.0
0.2
0.4
0.6
0.8
1.0
T1YFFM


Response: RECESSION
GW
40
20
0
-20
-40
0.0
0.2
0.4
0.6
0.8
1.0
HOUST