Bachelor Thesis: BSc^2 Econometrics & Economics

Erasmus University Rotterdam

Erasmus School of Economics

Predicting South African GDP Growth Rates Using Factor Models and Machine Learning Techniques

Author:
Lisa-Marie Plag

Student Number:
430040

Supervisor:
Prof. dr. P.H.B.F. Franses

Second Assessor:
dr. A.M. Schnucker

July 7, 2019

Abstract

This paper investigates the usefulness of a combined factor estimation and shrinkage approach in forecasting South African GDP growth rates one-quarter ahead. For this purpose, 62 quarterly macroeconomic variables from Q1 1996 to Q1 2019 are examined in an empirical forecasting experiment in addition to a simulation study of the constructed model. It is found that the hybrid model, combining boosting with principal component analysis, leads to significantly lower forecast errors than standard autoregressive forecasting methods. However, this result is heavily dependent on the boosting parameters. Simulations of data with varying dimensions reveal that the gain in forecasting accuracy achieved by the combination method is larger when more latent factors exist and that datasets with higher dimensions increase performance. As an extension, the effectiveness of using a recurrent neural network with long-short term memory to produce the forecasts is evaluated, giving rise to similar findings. Hence, it is concluded that machine learning methods are valuable tools to predict quarterly South African GDP growth rates if the parameters are chosen properly.

Contents

1 Introduction

Due to advances in technology, today's economists have access to a vast amount of data across variables and time for building predictive models. This is a curse and a blessing at the same time. While it might seem that the increased number of economic figures will improve the quality of related models, the contrary is often the case as an information overload can occur (Bai et al., 2008). Therefore, many researchers recently turned to the use of dimension reduction and shrinkage methods to filter out those variables that add value to their models (Kim and Swanson, 2014, 2018; Stock and Watson, 2012). This way, better predictions can be obtained for many economic variables of social relevance.

One variable that is of major interest to researchers and the public is the economic growth of a country, measured by the change in its gross domestic product (GDP). Economic growth is a key driver of development, which is particularly important for emerging markets such as African economies. Forecasting a country's GDP can help policymakers take appropriate measures to meet their targets and make their economies thrive. However, the task of producing macroeconomic forecasts in the context of the ever growing amount of data has become increasingly difficult. One of the main problems researchers in this field are facing is how to make use of all the potentially valuable data that is available to them. Some variables solely add noise to the model, leading to a negative impact on its performance and should therefore be omitted.

Factor analysis can be used as a way to achieve dimension reduction of a large number of explanatory time series. More specifically, statistical factor models such as principal component analysis (PCA) attempt to capture as much variation of the target variable as possible with the smallest number of estimated factors, where the variable of interest is assumed to have a linear relationship with the latent factors. This relation can be exploited to produce forecasts of macroeconomic variables, which can be used by central banks to decide on an economic policy.

Another way to shrink the number of variables considered for a model is by applying an ensemble learning technique called boosting, which was first proposed by Schapire (1990). Boosting is a machine learning algorithm that is meant to improve the accuracy of predictive models by retraining the same model various times, each time predominantly including those observations with the largest forecast errors. Initially, boosting was only implemented for classification problems (Freund and Schapire, 1997), however it was soon extended to regression analysis (Ridgeway et al., 1999). Moreover, boosting can be used in combination with factor models to choose appropriate variables for estimating the factors as suggested by Kim and Swanson (2018). By pre-screening a large dataset with a boosting method, the predictions of a factor-type model can potentially be improved (Bai and Ng, 2009). This leads to the following main research question:

How can machine learning techniques be used to outperform autoregressive models in predicting quarterly GDP growth rates of South Africa?

To answer this question, forecast errors of a simple autoregressive model are compared to those of a diffusion index forecasting model according to a specification type of Kim and Swanson (2018). The chosen type of model is constructed by first pre-selecting variables with boosting, then estimating latent factors with PCA, and finally estimating the factor coefficients by again applying a boosting method. The same methodology has already been applied by Kim and Swanson (2018) to predict monthly macroeconomic data from the United States. Furthermore, South African economic growth has been subject to similar studies using factor models and shrinkage methods (Cepni et al., 2018, 2019). However, the effectiveness of applying boosting in particular to quarterly South African GDP growth rates remains to be evaluated. Moreover, it would be interesting to check for robustness of the proposed methodology across different locations and data frequencies. Therefore, this study analyses if the constructed diffusion index model performs equally well for American and African data as well as for monthly and quarterly frequencies, to answer the first sub-question:

1) Is the diffusion index forecasting model robust to different data frequencies and regional differences between the United States and South Africa?

Additionally, the predictive ability of a recurrent neural network (RNN) is analysed in this context. RNNs are a special type of neural network that is able to learn underlying time dependencies without explicitly specifying them, which can be beneficial for modelling time series data such as economic growth. To the best of my knowledge, research on the use of RNNs for forecasting South African GDP growth does not exist yet. Consequently, another sub-question is:

2) Can forecast accuracy be improved by applying a recurrent neural network for predicting GDP growth?

After applying the above mentioned models to a large set of economic predictor variables from the United States and South Africa from January 1960 to May 2009 and Q1 1996 to Q1 2019, respectively, it was found that the model combining boosting with PCA is effective in predicting different data. The results of Kim and Swanson (2018) were successfully replicated, proving that the method indeed leads to a 13% reduction of the mean squared error relative to a standard autoregressive model when forecasting monthly GDP growth rates from the United States. Furthermore, the model significantly improves forecasts of quarterly South African growth rates by 6.5%. However, these results are highly sensitive to the boosting parameters chosen and the wrong set of parameters can deteriorate forecasting performance of the model, making it worse

than a simple autoregressive model. Furthermore, a simulation study confirmed the usefulness of the model for predicting time series data based on latent factors. The more underlying factors are hidden in the explanatory variables, the larger is the gain in forecast accuracy relative to an autoregressive model. Lastly, a recurrent neural network with ten neurons is able to significantly outperform the autoregressive model, but not the combination of boosting and PCA.

This report adheres to the following structure. Section 2 provides an overview of econometric and economic literature on forecasting macroeconomic time series and the South African economy. Next, Sections 3 and 4 describe the data and methodology used for empirical analysis in this study. The resulting outcomes are presented and interpreted in Section 5, followed by a final discussion and conclusion in Section 6.

2 Literature Review

2.1 Macroeconomic Forecasting with Many Predictors

To make better use of high-dimensional datasets in terms of variables and observations for forecasting an individual time series, Stock and Watson (2002a) developed the so-called diffusion index model, which combines the use of factor estimation with linear regression to construct a more parsimonious forecasting framework. Essentially, diffusion index models consist of two steps. First, a small number of unobserved common factors, also called indices, is extracted from a large set of potential predictors by applying PCA or a similar factor estimation technique. Second, these latent factors are included in a linear regression of the target variable on some baseline explanatory variables that are typically used by researchers to produce forecasts of the given target. The diffusion index method can be used with both, static and dynamic factor models to produce forecasts. Stock and Watson (2002a) derived several asymptotic properties of principal components, such as consistency and efficient forecasts and used Monte Carlo simulations to also validate small sample properties. Furthermore, Stock and Watson (2002a) applied factor models to predict 12-step-ahead growth rates of industrial production based on 149 monthly macroeconomic time series from January 1959 to December 1998 and found that the out-of-sample forecast errors of their model combining factors and autoregressive components are substantially lower than those of the benchmark autoregressive model or a vector autoregression. In another paper, Stock and Watson (2002b) present further empirical evidence in favour of diffusion indices. The diffusion index method for forecasting a time series with many predictors is still applied by many researchers today and represents the basic forecasting framework used throughout this paper.

Nevertheless, it is important to note that Bai and Ng (2009) discovered two major short-

comings of the diffusion index methodology. The first problem they pointed out in their work is that for PCA, factors are ordered in terms of the variance explained by the eigenvectors of the covariance or correlation matrix of the explanatory variables. Information criteria then select the first few factors based on this ordering. However, these factors should be used to forecast the dependent variable, which might actually require a different ordering concerning the importance of factors. While the chosen factors best capture the variation in the explanatory variables, they might not be the most important ones to explain the dependent variable. Moreover, Bai and Ng (2009) argue that another issue with the selection criteria used to construct diffusion indices results from the fact that when a certain lag order p^* of the autoregressive terms is chosen, all previous lags $p = 1, \ldots, p^*$ are incorporated in the model as well. In order to deal with the difficulty of selecting not only the right number but also the appropriate lags and factors in general, Bai and Ng (2009) adapted a popular machine learning algorithm called boosting to pre-select variables used for factor estimation in diffusion index models. This way, they developed the component-wise and block-wise boosting approaches for time series. Boosting was proposed by Schapire (1990) as a means to achieve a higher predictive accuracy by combining the results of many 'weak learners', such as ordinary least squares (OLS) estimators. More specifically, a linear combination of many individually estimated weak learners is constructed to form an opinion (prediction), which is why boosting is also called an ensemble learning technique. This technique initially only worked for classification problems, where the variable to be predicted belongs to a finite set of discrete classes. Ridgeway et al. (1999) modified boosting to work for continuous regression problems, however they did not deal with time dependencies in the data. Only Bai and Ng (2009) applied the algorithm to time series in combination with factor analysis. While factor analysis compresses the information of a large set of observed variables into a smaller set of common factors, boosting focuses on filtering out the most informative variables for predicting one particular variable of interest. Based on simulations of different data generating processes, Bai and Ng (2009) conclude that the best way of applying boosting with factor models is highly dependent on the dataset. For example, when the factor structure in the data is strong, boosting the estimated factors instead of the observed predictors performs better. Moreover, the performance gap between different ways to apply boosting is large, hence researchers need to carefully pick the most suitable procedure for their data. In an empirical evaluation, Bai and Ng (2009) also showed that their boosting methods can improve 12-month-ahead forecasts of inflation, changes in the Federal Funds rate, the growth rate of industrial production, the growth rate of employment and the level of the unemployment rate. Consequently, their component-wise boosting approach is incorporated in my forecasting framework.

Kim and Swanson (2018) compared numerous different combinations of factor estimation and machine learning techniques in terms of their ability to predict 11 monthly U.S. macroeconomic variables, including unemployment rates, inflation and GDP. They applied PCA, independent component analysis (ICA) and sparse principal component analysis (SPCA) to estimate factors and considered multiple machine learning shrinkage methods, such as bagging, boosting and ridge regression, to more accurately select factors or variables for the estimation thereof. In their paper, Kim and Swanson (2018) defined four distinct specification types, which differ in how the factor estimation and shrinkage methods are combined. For models of type 1, factors are estimated first followed by the machine learning techniques, while for type 2 models, a subset of the large set of explanatory variables is first constructed using the machine learning techniques followed by the estimation of factors and their coefficients. In contrast, specification types 3 and 4 solely consist of machine learning methods and do not include any factors in the forecasting equation. As forecasting framework for all combination models, Kim and Swanson (2018) used the diffusion index method introduced by Stock and Watson (2002a). By evaluating the forecast errors of four different types of model specifications relative to several autoregressive benchmark models, they found that the standard time series methods are often outperformed by factor-type forecasting models. Models combining factor-based dimension reduction with machine learning methods performed reasonably well, while pure machine learning or autoregressive models almost never achieved the lowest errors. Moreover, applying PCA to estimate factors gave better results at longer forecast horizons, whereas ICA and SPCA were preferred for one-step ahead forecasts. Similarly, expanding window estimation beat rolling window forecasts at a one-step horizon. Regarding GDP growth, forecast errors could be reduced by up to 13% relative to autoregressive models with lag order selected according to the Schwarz information criterion. Hence, the specification types of Kim and Swanson (2018) are used to construct an appropriate combination model for predicting GDP growth rates in the context of this report.

Only recently have Cepni et al. (2019) applied part of the factor estimation and shrinkage methods combined by Kim and Swanson (2018) to predict GDP growth data from five emerging markets, including Brazil, Indonesia, Mexico, South Africa and Turkey. Using a dynamic factor model (Giannone et al., 2008) as general forecasting framework, they compute monthly nowcasts and forecasts of quarterly GDP growth rates over a period from January 2005 to September 2017. In an empirical analysis, large numbers of monthly predictive variables were considered for each country, including 110 economic indicators for South Africa. However, Cepni et al. (2019) only employed SPCA for estimating factors and the elastic net, the least absolute shrinkage operator and least angle regression to select specific predictors. An autoregressive model and

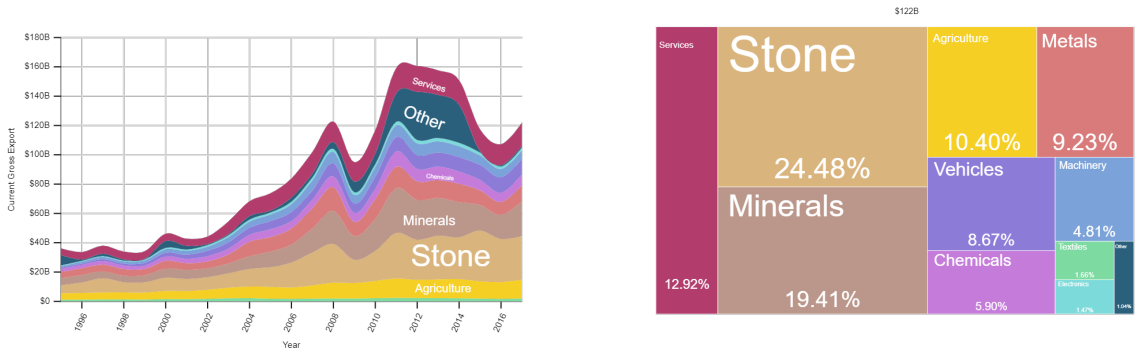
a dynamic factor model without pre-selection of variables were used to compute benchmark forecasts. These benchmarks were outperformed by models making use of dimension reduction and machine learning in most cases, which indicates that models similar to those of Kim and Swanson (2018) also work well for emerging market economies. My research aims to add to the discussion and evaluation of the effectiveness of combined factor estimation and machine learning techniques for high-dimensional macroeconomic data from developing countries. To fill a gap in this literature, boosting combined with PCA is applied to predict quarterly South African GDP growth based on a large set of predictors similar to the one used by Cepni et al. (2019).

A different approach to handle large amounts of macroeconomic time series involves the training of an artificial neural network (ANN), a class of machine learning models that is able to learn any underlying pattern in the data. In a comparison of numerous model selection criteria, Swanson and White (1997) empirically evaluated the forecasting ability of an adaptive ANN with up to five hidden neurons selected by the Schwarz information criterion. Using quarterly macroeconomic data of the U.S. from January 1960 to March 1993, they found that their neural networks were only slightly better than adaptive autoregressive models, if at all. Hence, constructing adaptive models with flexible lag and variable selection seems to be more beneficial than the potential non-linearity of ANNs. Similarly, Tkacz (2001) applied ANNs to predict Canadian GDP growth rates at one- and four-quarter horizons. The results of this paper showed that neural networks significantly improve the accuracy of four-quarter GDP growth forecasts compared to linear models. However, at the one-quarter forecast horizon the neural network was not able to even outperform a random walk. These results are not very promising, therefore I do not consider standard neural network models for forecasting GDP growth but a recurrent neural network (RNN) with long short-term memory (LSTM), which was developed by Hochreiter and Schmidhuber (1997). RNNs are particularly useful in dealing with time series data as they take into account previous sequences of input data. However, research on applying this specific type of neural network to forecast GDP growth remains to be done.

2.2 South African GDP Growth

With a GDP of $349.43 billion U.S. dollars as of 2017 (International Monetary Fund, 2018), South Africa represents the second largest economy of Africa, following Nigeria. Economic activity of South Africa has historically been steadily growing for several decades, with only few exceptions due to local or global crises. However, recently growth has slowed down to less than 2% a year due to political uncertainty within the country (Toyana, 2019). South African growth predominantly derives from financial services, the government, wholesale and retail trade

and manufacturing (Brand South Africa, 2018). The South African economy has shifted from a focus on the primary and secondary sectors to the tertiary sector. Moreover, South Africa sells a large part of its resources to other countries and the value of its exports has increased rapidly since 1995 (see Figure 1a). As can be seen in Figure 1b, the largest share of South African exports consists of stones, a category dominated by gold, platinum, and diamonds (Center for International Development at Harvard University, 2017). In fact, with 14.14% gold was the single most exported good of South Africa in 2017. Stones are followed by minerals, services, agriculture and metals. Hence, exports are driven by the mining industry. Overall, South African GDP growth largely depends on the prices of the resources it has an abundant supply of.



(a) 1995 - 2017 **(b)** 2017

Figure 1: South African exports by sector (Center for International Development at Harvard University, 2017)

Just as economic activity, monetary policy of the South African Reserve Bank (SARB), the central bank of South Africa, has evolved over time (Aron and Muellbauer, 2002). Aron and Muellbauer (2007) provide a detailed discussion of South African monetary policy from 1994 to 2007. They point out that during the 1990s, monetary policy in South Africa was opaque, leading to a decreased credibility of the SARB and negative effects on economic growth. Furthermore, Aron and Muellbauer (2007) argue that the shift to inflation targeting in 2002 has increased effectiveness of the SARB's policy, which also had a positive impact on GDP growth. However, an inflation targeting policy requires good macroeconomic forecasting models to determine appropriate targets (Svensson, 1997).

For this purpose, South Africa's central bank makes use of a range of models complementing each other to produce a good macroeconomic outlook and support policy making. The two main macroeconomic models of the SARB are the so-called 'Core' econometric model and the most recent Quarterly Projection Model (QPM) (Dejager, 2017). The 'Core' model is an error correction model, which is estimated using historical data and assumes a certain trend of the interest and exchange rate over the forecast horizon. The QPM is a gap model based on general

equilibrium theory and was introduced by the SARB in 2007 (Botha et al., 2017). It assumes that economic agents base decisions on their expectations for the future and that the economy moves along an equilibrium path, which it occasionally deviates from due to shocks. The large number of macroeconomic forecasting models used by the SARB highlights the importance of good frameworks for predicting South African GDP growth.

3 Macroeconomic Data

Kim and Swanson (2018) used quarterly U.S. GDP figures, which they transformed into monthly data according to the interpolation method of Chow et al. (1971). Moreover, the GDP data is converted into growth rates by taking logarithmic differences. Additionally, Kim and Swanson (2018) constructed a set of $N = 143$ explanatory variables to predict GDP growth, which is essentially an extended version of the dataset used by Stock and Watson (2012). For the purpose of verifying my methods, I use the same dataset[1] as Kim and Swanson (2018). Non-stationary time series are transformed to achieve stationarity by taking simple or logarithmic differences as indicated by the transformation code of each variable that comes with the dataset. The period they examined is January 1960 to May 2009, leading to a total number of $T = 593$ observations. For a full overview of all variables included, see Kim and Swanson (2018).

As the main interest of this research lies in applying selected methods of Kim and Swanson (2018) to forecast South African GDP growth, a second set of data is required. More specifically, quarterly real GDP growth rates in annual year-over-year percentages from Q1 1996 to Q1 2019 are considered, resulting in a total of $T = 93$ observations. To predict growth rates, quarterly observations of 62 South African economic variables, including year-over-year real GDP growth, were retrieved from Bloomberg. This is a subset of the monthly dataset created by Cepni et al. (2018), who analysed 109 variables from January 2003 to June 2018. However, I investigate Q1 1996 to Q1 2019 to allow for sufficiently large values of both T and N with quarterly frequency. The larger T, and hence the earlier the starting year of the sample, the smaller N since less explanatory variables are available. Consequently, only $N = 61$ South African macroeconomic variables, excluding GDP growth, can be analyzed for the chosen period. The final list of macroeconomic variables and their Bloomberg tickers can be found in Tables 7 and 8 in Appendix A.1. Further discussion of this dataset is provided by Cepni et al. (2018). All time series were tested for non-stationarity by using the augmented Dickey-Fuller test for unit roots. The null hypothesis of non-stationarity could not be rejected for 53 out of 62 time series at the 10% level.

[1]The data analyzed by Kim and Swanson (2018) was already used in a previous study (Kim and Swanson, 2014) and can be found on `https://sites.google.com/site/khdouble2/research`.

Therefore, these series were transformed by taking first differences or logarithmic first differences for non-negative variables.

Figure 17 in Appendix A.1 gives an overview of correlations between the lagged predictors and South African GDP growth. It can be seen that several lagged variables have a strong positive correlation with GDP growth, for instance South Africa Mining SA Constant Prices (SASGMINE Index), South Africa Real GDP Expenditure on GDP (SADXRGSA Index), South Africa Private Credit Extension Mortgage Advances (SACEMORT Index) and the Composite Business Cycle Indicator - Coincident Indicator (SACBCI Index). The strongest negative correlation occurs with South Africa Current Account SA (SACTLVL Index) but is still relatively weak. Moreover, there seem to be some clusters of variables that are strongly correlated and might contain similar information. Therefore, factor estimation techniques are expected to be a useful tool to summarize the informative value in this dataset and build a parsimonious forecasting model.

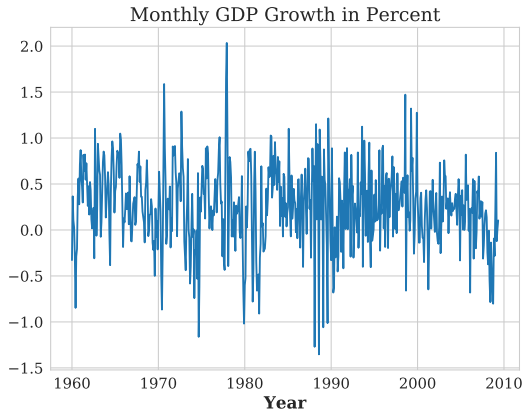


Figure 2: United States ($T = 593$)

Figure 3: South Africa ($T = 92$)

A graphical representation of the dependent variables to be predicted can be found in Figures 2 and 3. It can be seen that South African growth is higher than that of the United States, which is partly a result of the different data frequencies. In contrast to U.S. GDP growth, that of South Africa consists almost solely of positive growth rates. Especially during the first half of the South African sample from 1996 to 2008, economic growth is positive. After 2008, growth rates of South Africa are generally lower, suggesting a structural break in the data. The South African data is subject to more fluctuations, which can be seen by the higher standard deviation of 1.86 compared to a value of only 0.44 for the U.S. as reported in Table 1. From this Table and the graphs it can also be concluded that GDP growth rates of South Africa include more extreme values, ranging from -2.6% in Q2 2009 to 7.1% in Q4 2006, while growth of the U.S. moves between -1.4% and 2.0% only. Especially the decline in growth of -2.6% does not fit the rest of the sample, which consists almost exclusively of positive growth rates. This decline is probably

a result of the aftermath of the 2008 global financial crisis but could be influential for predictive models. Therefore, residuals of the forecasting models should be checked for outliers.

Table 1: Descriptive statistics of GDP growth rates in percent

	T	N	Min	Max	Mean	Std. Dev.	Skewness	Kurtosis
U.S.	593	143	-1.35	2.03	0.26	0.44	-0.22	1.00
S.A.	92	61	-2.60	7.10	2.69	1.86	-0.19	0.03

The data frequency for the U.S. is monthly, while that of South Africa (S.A.) is only quarterly.

Another important feature to be considered when forecasting economic growth is that growth often persists over time. This is highlighted by Figures 4 and 5, which clearly show a positive linear relation between GDP growth and its value during the previous period, both in the United States and South Africa, regardless of the different data frequencies. The relationship is more pronounced in the South African dataset, as Figure 5 displays an almost perfect 45-degree line through the origin. Hence, there seems to be significant autocorrelation.

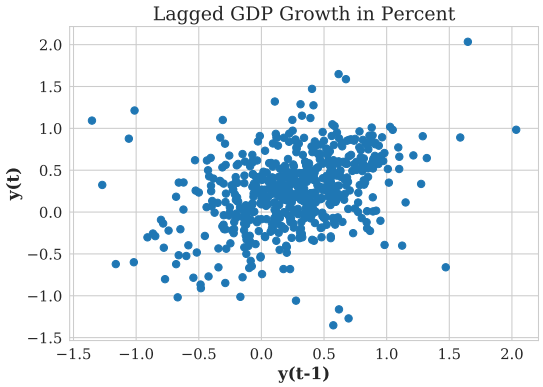


Figure 4: United States ($T = 593$)

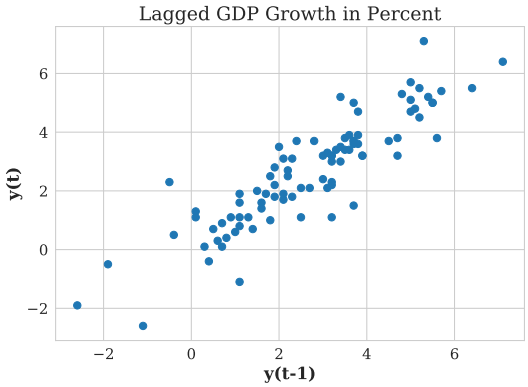


Figure 5: South Africa ($T = 92$)

An investigation of further lags of the dependent variables reveals that there indeed is significant autocorrelation at lags $p = 1, 2$ for the U.S. (see Figure 18 in Appendix A.1) and at lags $p = 1, 2, 3$ for South Africa (see Figure 6 below). Moreover, for U.S. GDP growth also lags 5 and 6 seem to be significant, which might be explained by the fact that these observations are monthly and thus lags 5 and 6 roughly correspond to a second quarterly lag. Moreover, Figure 18 in Appendix A.1 shows that the partial autocorrelation of U.S. GDP growth becomes insignificant after the first lag, indicating that this series might be an AR(1) process. Generally, the autocorrelation of the South African growth is stronger with values of close to 0.9 for the first quarterly lag and 0.7 for the second. Since the autocorrelation function of South African GDP growth declines towards zero and the partial autocorrelations oscillate but become insignificant after the second lag, this series is expected to follow an AR(2) process. Consequently, previ-

ous observations of the dependent variable are expected to provide a good starting point for producing forecasts of future values thereof.

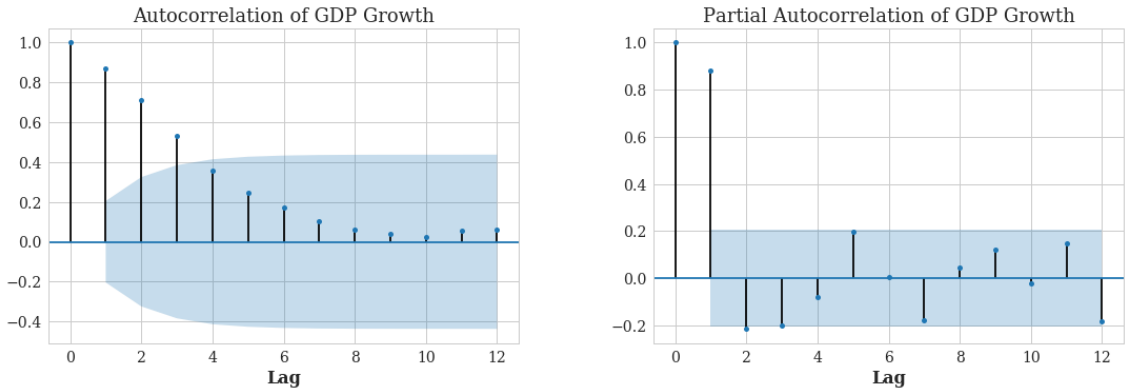


Figure 6: Autocorrelation functions of quarterly South African GDP growth

4 Methodology

4.1 Model Specification

In the following, I construct a forecasting model according to specification type 2 (SP2) of Kim and Swanson (2018) to predict GDP growth rates one step ahead. Out of the four specifications they investigated, this type was found to perform best for predicting economic growth at forecast horizon $h = 1$. Choosing this specification type means that common factors are estimated based on a subset of the high-dimensional dataset of predictors. This subset is extracted from the full set of 61 South African macroeconomic variables by using one of the machine learning techniques suggested by Kim and Swanson (2018). More specifically, a diffusion index forecasting model as described in Section 4.1.1 is constructed by first reducing the number of variables with boosting (Section 4.1.3) and then estimating the latent factors of this smaller dataset with principal component analysis (Section 4.1.2). Finally, boosting is applied again to estimate the factor coefficients in the diffusion index model.

4.1.1 Diffusion Index Forecasting

Estimating vector autoregressive models to produce predictions based on a large set of explanatory variables involves estimating a vast amount of parameters, which might have a negative impact on the predictive accuracy of the model (Stock and Watson, 2002b). Therefore, the following linear equation used in similar research (Bai and Ng, 2006; Kim and Swanson, 2018; Stock and Watson, 2002a,b) will be used to obtain forecasts for GDP growth rates:

$$Y_{t+h} = W_t\beta_W + F_t\beta_F + \epsilon_{t+h}, \tag{1}$$

where W_t is a small number of observed explanatory variables that should be included (here lags of Y_t), F_t are again the unobservable factors and ϵ_{t+h} is the error term. Moreover, β_W and β_F are the coefficients of the observed and unobserved explanatory variables, respectively. Because W_t is chosen to consist of lags of the dependent variable, $W_t\beta_W$ simply represents the right-hand side of an autoregressive model as further described in section 4.2.1. However, since β_W, β_F and F_t are all unknown, estimates for the common factors $\hat{F}_t$ first have to be obtained as described in Section 4.1.2. Only then can their coefficients $\hat{\beta}_F$ be obtained, for example by using least squares estimation. As in Kim and Swanson (2018), estimates of $\hat{\beta}_F$ are produced by applying boosting to the estimated factors. Combining Equation (1) with Equation (2) from the next section leads to the diffusion index model defined by Stock and Watson (2002a).

4.1.2 Factor Estimation

Let X_{ti} be the value of the ith variable at time $t = 1, \ldots, T$ from the large matrix of potential predictors[2] X $(T \times N)$. To predict GDP growth using this high-dimensional dataset, common factors will be estimated to reduce the dimension of the data. Hence, for each variable $i = 1, \ldots, N$, the factor representation at time t is given by

$$X_{ti} = F_t\lambda_i' + \eta_{ti}, \tag{2}$$

where F_t is a $1 \times k$ vector of latent factors ($k \ll N$), λ_i is a $1 \times k$ vector of factor loadings (corresponding to the ith row of the $N \times k$ loading matrix Λ) and η_{it} is the idiosyncratic error. The unobserved factors F_t, also called principal components, can be estimated using a factor model, such as PCA. PCA finds linear combinations of the variables in X_t that are orthogonal, meaning they are uncorrelated, and have maximum variance. This helps to decrease the dimensions of the predictor dataset if it contains correlated variables. To do so, eigenvectors of the variance covariance matrix of X_t, $\hat{V}$ are computed to solve the following equation:

$$\hat{V}e_i = \lambda_i e_i,$$

where λ_i is the eigenvalue corresponding to eigenvector e_i. Usually, all eigenvectors are normalized to have unit length ($e_i'e_i = 1$) and are arranged in decreasing order of their corresponding eigenvalues. The eigenvectors represent the axes of a reduced, k-dimensional subspace of the original N-dimensional space of explanatory variables. Moreover, since they are orthogonal, it holds that $e_i'e_j = 0$ for all $i \neq j$. After obtaining ordered eigenvectors, the ith principal component P_{ti}

[2]All explanatory variables are standardized to have zero mean and unit variance before estimating factors, as is commonly done for PCA.

can be computed as

$$P_{ti} = e_i' X_t,$$

where $i = 1, \ldots, N$. Since the aim is to describe X_{ti} with a limited number of factors, only the first $\hat{k}$ principal components are chosen and used as factors in the diffusion index model of Equation (1). Therefore, the number of factors $\hat{k}$ to be used for further analysis is chosen based on the following class of selection criteria proposed by Bai and Ng (2002):

$$\mathrm{IC}(\hat{k}) = \log\left(V(\hat{k}, \hat{F}^{\hat{k}})\right) + \hat{k}h(N,T), \tag{3}$$

where $\hat{F}^{\hat{k}}$ are the first $\hat{k}$ factors, $h(\cdot)$ is a penalty term and $V(\cdot)$ is a function minimizing the distance between the matrix of original variables X and their factor representation $\hat{F}^{\hat{k}}\Lambda^{\hat{k}\prime}$. More specifically, information criterion IC_{p2} by Bai and Ng (2002) is used, where the penalty is given by

$$h(N,T) = \frac{T+N}{T \cdot N} \log\left(\min\{N,T\}\right),$$

and $V(\hat{k}, \hat{F}^{\hat{k}})$ is the mean residual variance, which can be computed as

$$V(\hat{k}, \hat{F}^{\hat{k}}) = \frac{1}{N}\sum_{i=1}^{N}\left(\frac{1}{T}\sum_{t=1}^{T}\left(X_{ti} - \hat{F}_t^{\hat{k}}\lambda_i^{\hat{k}\prime}\right)^2\right).$$

Kim and Swanson (2018) limit the number of factors to a maximum of $k_{max} = 20$ in their analysis of 144 U.S. macroeconomic variables. Since the South African dataset with only 62 variables is considerably smaller, I set the maximum number of factors that can be selected to $k_{max} = 10$.

4.1.3 Boosting

As suggested by Kim and Swanson (2018), I use the component-wise L_2 boosting algorithm for time series, which was introduced by Bai and Ng (2009) for pre-filtering explanatory variables before estimating factors together with an information criterion to select the optimal stopping iteration m^* out of a total of M iterations. This way, only variables that are potentially useful for predicting the dependent variable are considered in PCA, which can speed up computations and improve forecast accuracy of the resulting model. Algorithm 1 shows the adjusted version of Kim and Swanson (2018), who applied a configuration with a maximum of $M = 50$ iterations and step size $\nu = 0.5$. For their specific implementation of boosting, the errors $Z_t = Y_t - \hat{Y}_t^{\mathrm{AR(p)}}$ are first computed by estimating an autoregressive model of order p as described in Section 4.2.1. During each iteration $m = 1, \ldots, M$, N linear regression models are fit to the residual Z_t and one variable from the $T \times N$ matrix X is chosen based on the sum of squared residuals (SSR).

The regressions are estimated using OLS and one of the explanatory variables at a time. To apply boosting to the factors obtained with PCA, X is simply replaced by $\hat{F}$ and N reduces to k in Algorithm 1. This way the factor coefficients of the diffusion index model of Equation (1) can be estimated.

Algorithm 1: Component-wise L_2 Boosting

Initialize $\hat{\Phi}_{t,0} = \bar{Z}$ for every t = 1, ..., T and $\hat{\beta}_{i,0} = 0$ for every i = 1, ..., N.
for *m = 1, ..., M* **do**
  for *t = 1, ..., T* **do**
    Compute the current residual $u_t = Z_t - \hat{\Phi}_{t,m-1}$.
  end
  for *i = 1, ..., N* **do**
    Regress u $(T \times 1)$ on the ith predictor X_i to obtain $\hat{b}_i$. Compute $\hat{e}_i = u - X_i\hat{b}_i$ and $\text{SSR}_i = \hat{e}_i'\hat{e}_i$.
  end
  Choose i_m^* to minimize SSR: $\text{SSR}_{i_m^*} = \min_{i \in [1,\ldots,N]} \text{SSR}_i$.
  for *t = 1, ..., T* **do**
    Update $\hat{\Phi}_{t,m} = \hat{\Phi}_{t,m-1} + \nu X_{i_m^*}\hat{b}_{i_m^*}$, where $0 < \nu \leq 1$ is the step size.
  end
  for *i = 1, ..., N* **do**
    Update $\hat{\beta}_{i,m} = \hat{\beta}_{i,m-1} + \nu\hat{b}_{i_m^*}\mathbb{1}_{i_m^*}$, where $\mathbb{1}_{i_m^*}$ is an $N \times 1$ standard unit vector with the i_m^*th element equal to 1.
  end
end

Stopping the algorithm at iteration m^* prevents over-fitting. This is done by choosing m^* to minimize the following criterion proposed by Bai and Ng (2009):

$$\text{IC}(m) = \log(\hat{\sigma}_m^2) + \frac{\log(T) \cdot df_m}{T},$$

where $\hat{\sigma}_m^2 = \sum_{t=1}^{T}(Z_t - \hat{\Phi}_{t,m})^2$ and $df_m = \text{trace}(B_m)$ are the degrees of freedom of the algorithm stopping in round m. Moreover, the matrix B_m is defined recursively as

$$B_m = B_{m-1} + \nu P_m(I_T - B_{m-1}),$$

with the projection matrix $P_m = X_{i_m^*}(X_{i_m^*}'X_{i_m^*})^{-1}X_{i_m^*}'$ constructed using the i_m^*th regressor and I_T being the $T \times T$ identity matrix. Furthermore, the initialization of B is given by $B_0 = \frac{1}{\nu}P_0 = \frac{\iota_T'\iota_T}{T}$, where ι_T is a $T \times 1$ vector of ones. Initially, a boosting step size of $\nu = 0.5$ is used with a maximum of $M = 50$ iterations. To prevent these parameters from influencing the results, a suitable combination of M and ν is chosen as outlined in Section 4.4, with $0 < \nu \leq 1$.

4.2 Model Evaluation

4.2.1 Benchmark Model

To evaluate the model constructed in this research, benchmark forecasts will be produced using a simple univariate autoregressive (AR) model with the lag order p being selected according to the Schwarz Information Criterion (SIC) derived by Schwarz et al. (1978). However, for the sake of efficiency the maximum lag order is limited to $p_{max} = 4$. The AR(p) model is defined as follows (Franses et al., 2014):

$$Y_t = \alpha + \phi_1 Y_{t-1} + \cdots + \phi_p Y_{t-p} + \epsilon_t,$$

where α is a constant, $\phi_1 \dots \phi_p$ are the coefficients of the lagged dependent variables and ϵ_t is a white noise disturbance term. From this specification, forecasts can be obtained as $\hat{Y}_{t+h}^{\text{AR(p)}} = \hat{\alpha} + \hat{\phi}_1 Y_{t+h-1} + \cdots + \hat{\phi}_p Y_{t+h-p}$, with $\hat{\alpha}$ and $\hat{\phi}_1 \dots \hat{\phi}_p$ being the model parameters estimated using OLS.

4.2.2 Forecast Comparison

Based on the diffusion index and benchmark models, one-period-ahead forecasts of GDP growth are constructed starting with 13 ($R = 156$) and almost 17 ($R = 67$) years of in-sample data for the U.S. and South Africa, respectively[3]. For instance, the first diffusion index forecast for U.S. GDP growth according to Equation (1) is computed as $\hat{Y}_{157} = W_{156}\hat{\beta}_W + \hat{F}_{156}\hat{\beta}_F$, while the first quarterly forecast for South Africa is given by $\hat{Y}_{68} = W_{67}\hat{\beta}_W + \hat{F}_{67}\hat{\beta}_F$.

Both, the model parameters and the optimal number of lags and factors are re-estimated before each new point forecast by using an expanding window because this estimation method produced the best GDP growth predictions in the analysis of Kim and Swanson (2018) for a forecast horizon of $h = 1$. The forecasts of the combination model described in Section 4.1 are then compared to the benchmark forecasts by using the Mean Squared Forecast Error (MSFE), which is computed as

$$\text{MSFE}_h = \frac{1}{P}\sum_{j=0}^{P-h}\left(Y_{R+h+j} - \hat{Y}_{R+h+j|R+j}\right)^2 = \frac{1}{P}\sum_{j=0}^{P-h}\left(\widehat{\epsilon_{R+h+j|R+j}}\right)^2,$$

where R is the number of in-sample observations, P is the number of out-of-sample observations, Y_t is the actual GDP growth at time t and $\hat{Y}_{t|t-h}$ is the corresponding forecast of GDP growth at time t using information available at time $t-h$ for $t = R+h, \dots, R+h+P-1$. To summarize the predictive ability of the diffusion index model of SP2 relative to the AR benchmark, MSFE ratios

[3]The different conversions from in-sample years to the number of in-sample data points R results from the different frequencies of the U.S. and South African datasets.

are computed as $\text{MSFE}_h^{\text{SP2}}/\text{MSFE}_h^{\text{AR}}$. Thus, a lower MSFE ratio indicates a better performance of the SP2 forecasts.

To test for the significance of differences in forecast accuracy, the Diebold-Mariano (DM) test statistic can be used (Diebold and Mariano, 1995):

$$\text{DM} = \frac{1}{P}\sum_{t=1}^{P}\frac{d_t}{\hat{\sigma}_{d_t}} = \frac{\bar{d}}{\hat{\sigma}_{d_t}/\sqrt{P}} \xrightarrow{d} N(0,1),$$

where $d_t = \left(\widehat{\epsilon^{\text{AR}}_{t+h|t}}\right)^2 - \left(\widehat{\epsilon^{\text{SP2}}_{t+h|t}}\right)^2$ is the loss differential of two predictions to be compared and $\bar{d}$ is its sample mean. Moreover, $\hat{\sigma}_{d_t}$ is the estimated sample standard deviation of the differential and $\widehat{\epsilon^{\text{AR}}_{t+h|t}}$ and $\widehat{\epsilon^{\text{SP2}}_{t+h|t}}$ are the estimated forecast errors of the AR model and the model of SP2. Hence, if the value of DM is positive, the forecasts of the diffusion index model beat the AR forecasts and vice versa. The number of predictions P is 422 for the U.S. and 25 for South Africa, leading to sufficiently large samples for testing statistical significance of any differences in predictive accuracy found. As the interest of this research is in improving upon the standard AR model forecasts, the null hypothesis is defined as

H_0: The forecast errors of the AR model are smaller than those of the alternative model.

As Kim and Swanson (2018) argue, the DM statistic asymptotically follows a standard normal distribution for non-nested models. Hence, the null hypothesis of more accurate AR forecasts is rejected for values that are more extreme than one-sided critical values from the N(0,1) distribution, which are 2.326, 1.645, and 1.282 for the 1%, 5%, and 10% significance levels, respectively.

4.2.3 Simulation

To further investigate the finite sample behaviour of the factor estimation and diffusion index forecasting methods, data with different dimensions N and T as well as varying numbers of underlying factors k is simulated. First, k factors are generated to follow an autoregressive process of order one. From these latent factors, the observed X variables are constructed. Hence, the general form of the data generating process (DGP) for $t = 1, \ldots, T$ looks as follows:

$$\begin{aligned} X_{ti} &= F_t\lambda_i' + \sqrt{k}\eta_{ti}, & \eta_i &\sim N(0,1), & i &= 1,\ldots,N, \\ F_{tj} &= \alpha_j F_{t-1j} + u_{tj}, & u_j &\sim N(0,1), & j &= 1,\ldots,k, \end{aligned}$$

where $\alpha = (0.8, 0.7, 0.6, 0.5, 0.4)'$ is the same for each of 100 iterations and the first k elements of α are used depending on the true number of factors $k = 1, 3, 5$ simulated. In contrast, λ_i $(1 \times k)$ the ith row of the $N \times k$ matrix of factor loadings Λ with all elements $\lambda_{ij} \sim N(0,1)$ is re-drawn

for each iteration. Success rates of estimating the true number of factors in X are obtained by applying PCA with the IC_{p2} information criterion of Equation (3) to the simulated data. Furthermore, the performance of the SP2 diffusion index method as a whole can be evaluated by also creating a target series y from the generated factors as

$$y_{t+h} = F_t\beta_F + v_{t+h}, \qquad v \sim N(0,1),$$

where $\beta_F = (0.8, 0.5, 0.3, 0, -0.3)'$ is fixed during the simulation. This is similar to the simulation setup Bai and Ng (2009) used to evaluate their boosting algorithms. To evaluate forecast accuracy, the last 25 observations of the generated data are taken as out-of-sample data, irrespective of dimension T. The out-of-sample values of y are then predicted using the AR and SP2 models, and the forecasts are compared as described in Section 4.2.2. For each dimensionality (T, N) of the DGP, the average MSFE ratio over all iterations and the number of rejected DM tests is computed to evaluate the effectiveness of the constructed method on simulated data.

4.3 Recurrent Neural Networks

As extension deviating from the models specified by Kim and Swanson (2018), I implement a recurrent neural network (RNN) with long short-term memory (LSTM) (Hochreiter and Schmidhuber, 1997) to forecast GDP growth. RNNs are a particularly suitable type of neural network to forecast time series data as they are able to learn underlying time dependencies. Neural networks provide several advantages over standard time series methods, such as the ability to learn non-linear relations and to deal with noise in the data without manually identifying outliers. A more detailed explanation of neural networks and the LSTM architecture is provided below.

4.3.1 Introduction to Neural Networks

An artificial neural network (ANN) is a popular supervised learning algorithm that is able to approximate almost arbitrary complex functions. In general, a standard feed-forward neural network consists of an input layer, one or more hidden layers and an output layer, each containing a variable number of computational units called neurons. Each neuron performs a computation

$$z = \sum_{i=1}^{N} w_i x_i + b = w'x + b, \tag{4}$$

where x is the $N \times 1$ input vector, w is an $N \times 1$ vector of parameters, also referred to as weights, and b is called the bias term. Every unit has its own set of weights w applied to the input in a linear transformation, therefore the output z a neuron produces is simply a weighted

sum of its inputs. This sum is then fed into a non-linear activation function $g(\cdot)$. The very first neuron simply employed a step function as activation (McCulloch and Pitts, 1943). Today, popular activation functions include the sigmoid and tanh functions, among others. Several computational units can then be combined to form a layer. Hence, a neural network can be regarded as a mapping $A : X \to Y$ consisting of a total of L layers, where each individual layer l contains K_l neurons. Each of these layers l is fed a vector of output data from the previous layer $(l-1)$ as input, with the size of this vector being the number of neurons in its preceding layer. Within a given layer l, every neuron k produces a weighted sum of its input as shown in Equation (4) according to weight vector $w_k^{(l)}$ and bias term $b_k^{(l)}$. Next, the result is converted to the input size required by the subsequent layer by applying activation function $g^{(l)}$, hence the output of layer l is

$$Z^{(l)} = g^{(l)}(W^{(l)}Z^{(l-1)} + B^{(l)}),$$

where uppercase letters denote the layer analogue of the individual neuron variables $z_k^{(l)}$, $w_k^{(l)}$ and $b_k^{(l)}$. The input layer works similarly, except that it takes the original data X as input, hence $Z^{(0)} = X$. Lastly, the output layer L produces a prediction of a given dependent variable Y, which is the final result of the network:

$$\hat{Y}(X) = W^{(L)}Z^{(L)} + B^{(L)}.$$

The layers between original input X and output Y are hidden layers. Neural networks can contain one or more hidden layers and their ability to fit complex functions increases with the number of layers. However, the number of hidden layers depends on the specific data at hand and adding too many layers can create undesired noise. In general, a parsimonious network size in terms of hidden layers and neurons per layer should be preferred. This is justified by the approximation theorem of Hornik et al. (1989), which states that neural networks containing only one hidden layer can approximate arbitrary continuous functions if their hidden layer includes enough neurons. In order to obtain parameters W and B that lead to a good fit with the training data, an algorithm called backpropagation is usually used. Backpropagation pushes the errors of later layers back through the network, thereby minimizing a chosen loss function such as the mean squared error (Werbos et al., 1990).

4.3.2 Long Short-Term Memory Networks

Long short-term memory (LSTM) is a special type of RNN architecture, therefore the general theory of RNNs is explained briefly. RNNs are the standard tool to incorporate lags of the input

data in a neural network. Consequently, RNNs are often used for sequential data, such as texts, or time series data, such as GDP growth. In general, RNNs work similar to the standard ANNs discussed above. However, in contrast to feed-forward ANNs, RNNs contain loops enabling them to pass on previous values of the input data to later stages of the network. Figure 7 shows a loop of an RNN and its unrolled version, where $A : X \to H$ is a part of the network mapping input x to some output, which is denoted by h instead of z in this representation.

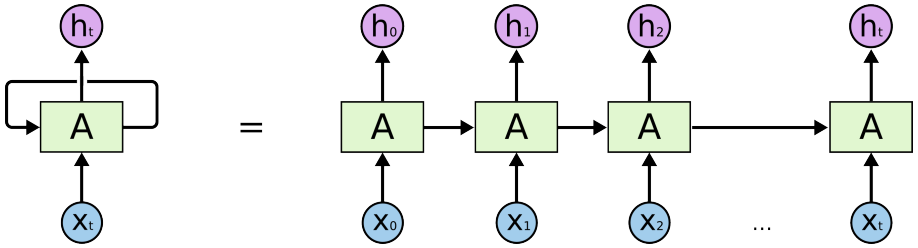
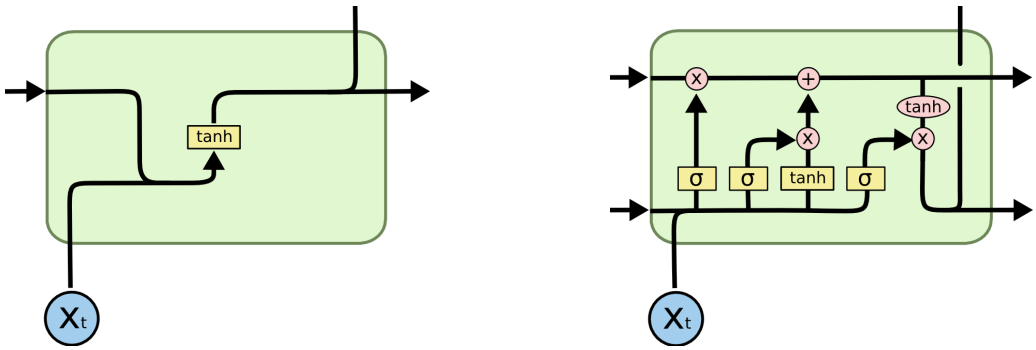


Figure 7: RNN representation (Olah, 2015)

A more detailed representation of the mappings performed by a standard RNN can be found in Figure 8a. The tanh activation function is often used in RNNs as its second derivative approaches zero only slowly, and is defined as

$$g(z) = \tanh(z) = \frac{\exp(z) - \exp(-z)}{\exp(z) + \exp(-z)} = \frac{\exp(2z) - 1}{\exp(2z) + 1}.$$

However, RNNs can suffer from the so-called vanishing gradient problem (Hochreiter, 1998). This means that during backpropagation, gradients of a simple RNN configuration can vanish or even explode. The LSTM architecture for RNNs proposed by Hochreiter and Schmidhuber (1997) solves this issue and is therefore used in this research. Figure 8b displays the more elaborate structure of the mappings performed by an LSTM model, which includes, among others, so-called gates for input and output data. For a more detailed explanation of the individual components of the LSTM architecture, I refer to Hochreiter and Schmidhuber (1997).



(a) RNN **(b)** LSTM

Figure 8: Network layer representations (Olah, 2015)

A recurrent neural network with only lagged dependent variables as input is essentially a non-linear version of univariate AR models (Kuan and Liu, 1995). Multivariate LSTM networks also exist but are more time-consuming to estimate. Due to time constraints, I restrict my analysis of neural networks to univariate LSTM models[4], taking $p_{max} = 4$ lags of GDP growth as input.

4.4 Parameter Optimization

Machine learning methods are highly dependent on their so-called hyper-parameters. Choosing the most suitable parameters for a given dataset is therefore crucial for obtaining good results. This applies to both, boosting and neural networks. Hence, different combinations of the model parameters are compared using a linear grid search. For boosting, the parameters to be fine-tuned are the maximum number of iterations M and the step size ν. For neural networks, the number of hidden neurons n and the epochs E, which represent the number of iterations the network is trained, are adjusted. Moreover, since each time a neural network is trained the weights are randomly initialized, the network with the best configuration is trained 10 times to prevent this randomness from having an influence on the conclusion.

5 Results

5.1 Simulation Results

In the following, the results of applying PCA and the boosted diffusion index forecasting model to simulated data of varying dimensions are reported. Table 2 verifies that the implemented PCA algorithm together with the IC_{p2} selection criterion enables me to recover the true number of factors in the generated data. For this purpose, the average number of factors estimated during 100 replications of the simulation and the corresponding hit rate, the percentage of replications in which the number of factors was estimated correctly, are displayed for each combination of dimensions T and N. It can be seen that when T or N are too small, namely 10, PCA is not able to estimate the true number of factors at all. Hence, the hit rates are zero in the first rows and columns for all underlying factors $k = 1, 3, 5$. Starting from a sample size of $T = N = 20$, the correct number of factors is obtained in 96 out of 100 replications. However, this is only the case for Panel A of Table 2, which corresponds to one underlying factor in the data. For three factors (Panel B), the hit rate decreases to 0.26 and for five factors (Panel C) even to zero. When $N = 20$, the hit rate decreases as T increases for $k = 1$ and the number of factors is overestimated. The contrary is the case when $k = 3$ or $k = 5$. For these underlying numbers of

[4]All LSTM-type models were implemented in Python using the Keras module and its predefined LSTM layers. The code for these and all other models of this research can be found in Appendix A.3.

factors, the hit rate increases as T increases for $N = 20$, even though this only happens starting from $T = 50$ when $k = 5$. For $T > 10$ and $N > 20$ in Panel A of Table 2, the number of factors is always correctly estimated. In Panel B, hit rates are one for larger values $T > 20$ and $N > 20$, hence more observations are required when there are more factors in the data. Furthermore, for $T > 50$ and $N > 50$ hit rates become equal to one in Panel C, even though there are two other cases when the hit rate is one for $N = 50$, namely for very large T. In conclusion, PCA with the chosen selection criterion works well for $T > 10$, $N > 20$ when there is one factor, $T > 20$, $N > 20$ when there are three factors and $T > 50$, $N > 50$ when there are five factors. Therefore, the more latent factors are present in the observed data, the more difficult it becomes to accurately estimate the true number of factors and the more observations and variables are needed.

Table 2: Estimated number of factors for different dimensions of the data with hit rates in parentheses

T \ N	10		20		50		100		200	
PANEL A: $k = 1$										
10	9.43	(0.00)	9.02	(0.00)	9.00	(0.00)	9.00	(0.00)	9.00	(0.00)
20	10.00	(0.00)	1.36	(0.96)	1.00	(1.00)	1.00	(1.00)	1.00	(1.00)
50	10.00	(0.00)	1.80	(0.88)	1.00	(1.00)	1.00	(1.00)	1.00	(1.00)
100	10.00	(0.00)	2.56	(0.81)	1.00	(1.00)	1.00	(1.00)	1.00	(1.00)
200	10.00	(0.00)	3.84	(0.68)	1.00	(1.00)	1.00	(1.00)	1.00	(1.00)
500	10.00	(0.00)	3.92	(0.67)	1.00	(1.00)	1.00	(1.00)	1.00	(1.00)
1000	10.00	(0.00)	4.33	(0.62)	1.00	(1.00)	1.00	(1.00)	1.00	(1.00)
PANEL B: $k = 3$										
10	9.29	(0.00)	9.00	(0.00)	9.00	(0.00)	9.00	(0.00)	9.00	(0.00)
20	10.00	(0.00)	2.80	(0.26)	2.62	(0.66)	2.85	(0.86)	2.89	(0.89)
50	10.00	(0.00)	2.96	(0.88)	3.00	(1.00)	3.00	(1.00)	3.00	(1.00)
100	10.00	(0.00)	3.19	(0.94)	3.00	(1.00)	3.00	(1.00)	3.00	(1.00)
200	10.00	(0.00)	3.16	(0.96)	3.00	(1.00)	3.00	(1.00)	3.00	(1.00)
500	10.00	(0.00)	3.22	(0.96)	3.00	(1.00)	3.00	(1.00)	3.00	(1.00)
1000	10.00	(0.00)	3.11	(0.96)	3.00	(1.00)	3.00	(1.00)	3.00	(1.00)
PANEL C: $k = 5$										
10	9.32	(0.00)	9.00	(0.00)	9.00	(0.00)	9.00	(0.00)	9.00	(0.00)
20	10.00	(0.00)	2.37	(0.00)	2.77	(0.05)	3.62	(0.25)	3.78	(0.24)
50	10.00	(0.00)	3.67	(0.21)	4.21	(0.40)	4.89	(0.89)	4.97	(0.97)
100	10.00	(0.00)	4.35	(0.54)	4.95	(0.95)	5.00	(1.00)	5.00	(1.00)
200	10.00	(0.00)	4.77	(0.79)	4.99	(0.99)	5.00	(1.00)	5.00	(1.00)
500	10.00	(0.00)	4.88	(0.81)	5.00	(1.00)	5.00	(1.00)	5.00	(1.00)
1000	10.00	(0.00)	4.83	(0.79)	5.00	(1.00)	5.00	(1.00)	5.00	(1.00)

The values in this Table are averages over 100 simulation replications. Each panel represents a different number of k underlying factors in the DGP. The estimated number of factors $\hat{k}$ is selected based on the IC_{p2} criterion with $k_{max} = 10$ as discussed in Section 4.1.2.

The above simulation findings for PCA are consistent with those of Bai and Ng (2002) and show that the information criterion IC_{p2} correctly specifies the number of factors for large enough

dimensions of the dataset. Additionally, Table 9 in Appendix A.2 provides simulation results with 1000 iterations. As these PCA estimation outcomes are highly similar, it can be concluded that the discussed hit rates are a good representation of the performance on DGPs of this type.

Table 3 contains the results of applying the SP2 method to simulated data with different dimensions. Since 25 observations are required to form the hold-out sample and evaluate the predictive ability of the model, T only starts at 50 for this experiment. To compare the forecasting performance of the SP2 model with that of the AR model based on simulated data, average MSFE ratios and the corresponding number of iterations for which the DM test of better AR forecasts was rejected are shown for each dimensionality. On average, all MSFE ratios are smaller than 1, indicating lower forecast errors of the SP2 model. However, the null hypothesis of smaller AR forecast errors could never be rejected for all 100 replications of the simulation, as the rejection numbers are smaller than 100. Naturally, less rejections usually also come with higher MSFE ratios as the relatively higher forecast errors of the SP2 model make it harder to reject superiority of AR forecasts. In general, it can be seen that the more factors are present in the data, the easier it becomes to reject the DM test. This can be explained by the inability of the AR model to capture the information contained in the factors, which are incorporated in the SP2 model leading to relatively better forecasts when more factors are underlying the data.

In Panel A of Table 3, no clear pattern is visible at first sight. When $k = 1$, the lowest MSFE ratio of 0.671 occurs for data with $T = 50$ and $N = 100$. Moreover, the DM test rejected higher accuracy of AR forecasts 75 out of 100 times at the 10% significance level for these dimensions. However, the same number of rejections was also found for $T = 1000$ and $N = 10$ but with a larger MSFE ratio of 0.734. The highest relative forecast errors and correspondingly lowest amount of rejections was found for dimensions $T = 50$ and $N = 20$. When both T and N are too low, factor estimation cannot properly capture underlying patterns in the data as demonstrated by the PCA simulation results. For this number of factors, it is striking that the SP2 method performs best relative to the AR model for datasets with either large T and small N or for small T and large N. This might be due to the AR model performing poorly when only a small number of observations is available, thus resulting in high MSFEs. These can then easily be outperformed by the SP2 model because of the additional information contained in the factors, which can be estimated accurately with a large number of variables N.

For $k = 3$ factors, a pattern seems to emerge with good forecasts located towards the lower right half and relatively bad forecasts in the upper left of Panel B. Dimensions $T = 1000$ and $N = 50$ result in the smallest MSFE ratio of 0.690. Datasets with these dimensions also exhibit a large number of rejections equal to 72, although this is not the largest number reported for this

value of k, which is 76 for $T = 1000$ and $N = 20$. The worst relative forecasting performance of the SP2 model is achieved when $T = 50$ and $N = 10$, leading to the highest MSFE ratio of 0.838 and the lowest number of DM test rejections for this number of factors.

In Panel C of Table 3 corresponding to five factors, a clear pattern is visible. The larger both dimensions T and N, the better the predictions of the SP2 model relative to the AR model as indicated by lower MSFE ratios and more rejections of the DM test. Similarly, the smaller the dimensions of the dataset, the worse is the relative performance of the SP2 model. Both, the lowest MSFE ratio of 0.675 and the largest number of rejected DM tests is achieved for the highest dimensions of $T = 1000$ and $N = 200$, while the largest MSFE ratio of 0.905 with the least rejections is obtained for the smallest dimensions $T = 50$ and $N = 10$.

Table 3: MSFE ratios for different dimensions of the data with number of rejected DM tests in parentheses

T \ N	10		20		50		100		200	
PANEL A: $k = 1$										
50	0.809	(48)	0.828	(41)	0.720	(57)	0.671	(75)	0.688	(71)
100	0.756	(56)	0.747	(61)	0.795	(47)	0.760	(53)	0.702	(71)
200	0.756	(64)	0.761	(55)	0.769	(58)	0.748	(60)	0.768	(55)
500	0.757	(59)	0.748	(69)	0.732	(73)	0.734	(64)	0.750	(67)
1000	0.734	(75)	0.751	(63)	0.748	(58)	0.742	(61)	0.759	(62)
PANEL B: $k = 3$										
50	0.838	(41)	0.806	(40)	0.755	(61)	0.716	(64)	0.705	(63)
100	0.774	(62)	0.750	(61)	0.752	(54)	0.709	(71)	0.702	(74)
200	0.756	(59)	0.748	(65)	0.727	(62)	0.737	(60)	0.725	(65)
500	0.760	(61)	0.713	(73)	0.720	(71)	0.727	(58)	0.724	(67)
1000	0.774	(58)	0.718	(76)	0.690	(72)	0.702	(71)	0.700	(75)
PANEL C: $k = 5$										
50	0.905	(31)	0.859	(31)	0.791	(52)	0.790	(54)	0.803	(51)
100	0.819	(51)	0.783	(54)	0.738	(62)	0.716	(58)	0.703	(66)
200	0.790	(58)	0.731	(63)	0.734	(60)	0.717	(68)	0.702	(68)
500	0.767	(63)	0.726	(71)	0.710	(70)	0.707	(70)	0.693	(73)
1000	0.771	(65)	0.732	(63)	0.703	(73)	0.672	(74)	0.675	(74)

The values in this Table are averages over 100 simulation replications. Each panel represents a different number of k underlying factors in the DGP. The estimated number of factors $\hat{k}$ is selected based on the IC_{p2} criterion with $k_{max} = 10$ as discussed in Section 4.1.2. The MSFE ratios are based on forecast errors of the SP2 model relative to the AR model. For the SP2 model, the boosting parameters are $M = 50$ and $\nu = 0.5$. The number of rejected Diebold-Mariano (DM) tests is based on the 10% one-sided standard normal critical value of 1.282. Results at the 5% level can be found in Table 10 in Appendix A.2.

5.2 Replication of Kim and Swanson (2018)'s Results

After verifying the models with simulated data, the results of Kim and Swanson (2018) were replicated. The same U.S. dataset was used, leading to a MSFE ratio of 0.869 compared to the ratio of 0.871 reported in Table 3 of their paper. As the difference between these ratios is

negligible, I conclude that I successfully implemented their SP2 model. With a test statistic of $DM = 2.628$, the null hypothesis of better AR forecasts is rejected at the 1% level, indicating that the combination model produces better forecasts of U.S. GDP growth. This is in accordance with the findings of Kim and Swanson (2018). In Figures 9 and 10, the actual values of GDP growth are plotted against AR and SP2 forecasts, respectively. It can be seen that the AR forecasts are often more extreme, especially after very large or small previous growth rates, as they are based solely on lags of the dependent variable. The SP2 forecasts seem less noisy, but are still not able to capture particularly large or small changes in GDP. When analysing the fitted models, it could be seen that most of the time, the AR model was estimated with $\hat{p} = 1$ lags. This is in line with the discussion of the data in Section 3. However, it seems that the factors in the SP2 model add some predictive value and reduce the noise of previous growth rates.

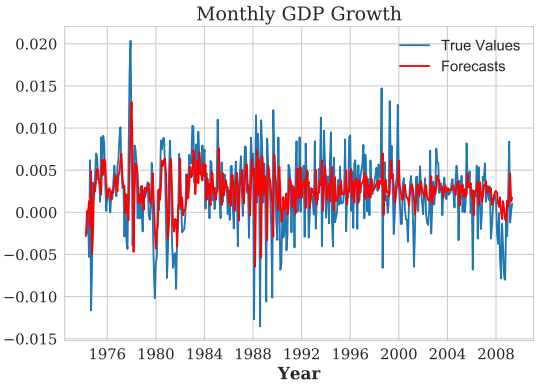


Figure 9: AR(p) Model (MSFE=1.940e-05)

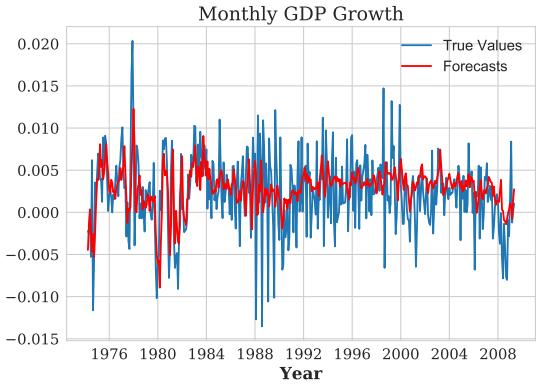


Figure 10: SP2 Model (MSFE=1.685e-05)

5.3 Application to South African Data

Applying the same method with the same parameters ($M = 50$, $\nu = 0.5$) to the South African dataset results in an MSFE ratio of 0.964, implying that the SP2 forecasts perform slightly better than those of the AR model. However, this difference is not significant as shown by the Diebold-Mariano test statistic of $DM = 0.706$. While the forecast errors are on average lower for the SP2 model, the large variation in the loss differential renders this result insignificant. When looking at graphs of the two forecast series in Figures 11 and 12, it becomes apparent that the AR and SP2 forecasts do not differ that much.

After further investigation of the estimated models, it was found that the number of lags chosen for the AR model was $\hat{p} = 2$ for all point forecasts made. This corresponds to the expected number of lags based on the analysis of the autocorrelation and partial autocorrelation functions in Section 3. Therefore, it can be concluded that the AR model is specified correctly and its forecasts are as accurate as possible for this type of model. Moreover, the fact that higher

AR forecast accuracy could not be rejected for South African GDP growth might arise from the different characteristics of the dataset. As there are less observations across time and variables for South Africa, the way boosting is best applied to this data most likely differs from the method used on the larger American dataset. Hence, the outcomes of optimizing the boosting parameters are discussed before interpreting any results for South Africa.

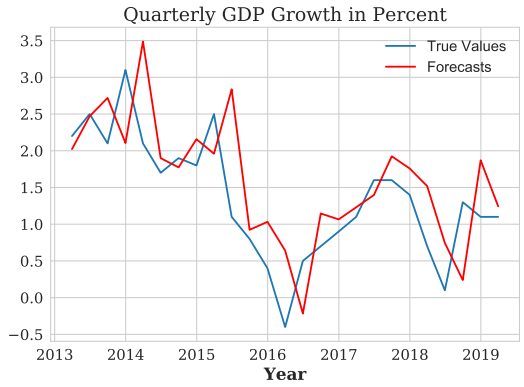


Figure 11: AR(p) Model (MSFE=0.487)

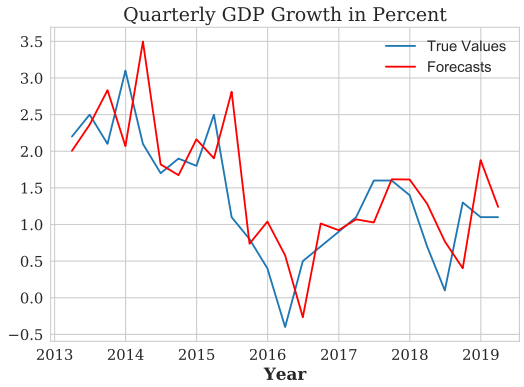


Figure 12: SP2 Model (MSFE=0.470)

5.3.1 Parameter Optimization

Table 4 contains MSFE ratios of the SP2 method relative to AR forecasts for different combinations of boosting iterations M and step size ν. Most remarkably, the forecast accuracy of the SP2 method varies largely depending on the boosting parameters. If the wrong set of parameters is chosen the diffusion index forecasts are even outperformed by autoregressive predictions, which is shown by MSFE ratios above 1. For instance, when $M = 50$ and $\nu = 0.2$, the maximum MSFE ratio of 1.24 is reached indicating that the SP2 forecast errors are notably higher than the AR ones. In general, it seems that larger values of M and ν are better as the MSFE ratios in the lower right half of Table 4 are mainly below 1. However, all of the three best results, which were also the only ones for which the DM test could be rejected, were obtained with $\nu = 0.6$.

Table 4: MSFE ratios for different combinations of boosting parameters M and ν

M \ ν	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1
30	1.071	1.156	1.202	1.178	1.085	1.017	1.087	0.988	0.962	1.081
40	1.074	1.082	1.145	1.104	1.032	0.950	1.079	0.979	1.038	0.989
50	1.045	1.240	1.148	0.965	0.964	0.942**	1.015	0.973	1.019	0.980
60	1.108	1.085	1.090	0.990	0.954	0.935**	1.002	0.990	1.029	0.969
70	1.181	1.181	1.045	1.034	0.959	0.935**	1.002	0.990	1.026	0.962

The ratios in this Table represent MSFEs of the SP2 model relative to the AR(p) model with $p_{\max} = 4$. For entries marked with *, ** and *** the DM test rejected higher accuracy of the AR(p) forecasts at the 10%, 5% and 1% significance level, respectively.

For simplicity, I just pick the parameter configuration with the lowest MSFE ratio and highest DM test statistic. For $\nu = 0.6$, two values of M yield similar MSFE ratios of about 0.935. But the DM value for boosting with $M = 70$, $\nu = 0.6$ is slightly higher than for the parameters $M = 60$, $\nu = 0.6$, therefore the forecasts of this configuration are investigated further.

After choosing appropriate boosting parameters for the South African dataset, the forecast performance measures of the SP2 model became $MSFE = 0.935$ and $DM = 1.955$. Figures 13 and 14 again compare the forecasts of the AR model with the SP2 model, but now using boosting parameters $M = 70$ and $\nu = 0.6$. The results are similar in that the difference between the two forecasts is not immediately visible. While some forecasts are almost exactly the same, others show an improved fit with the SP2 model. Overall, this leads to a significantly lower MSFE of the optimized SP2 forecasts relative to the AR model.

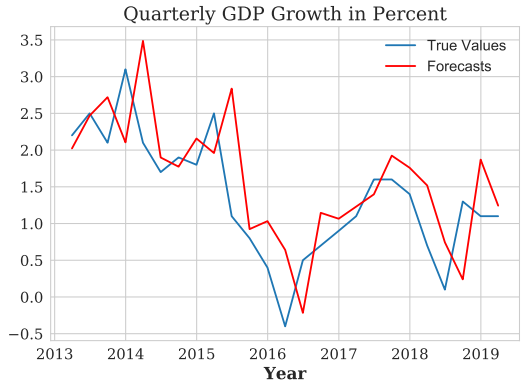


Figure 13: AR(p) Model (MSFE=0.4871)

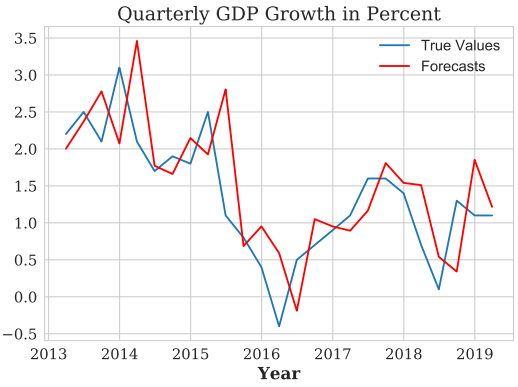


Figure 14: SP2 Model (MSFE=0.4554)

A graphical representation of the in-sample residuals for the last point forecasts and the in-sample residual skewness and kurtosis for each forecast is given by Figures 19 and 20 in Appendix A.2. Inspecting these graphs can help to detect outliers that could not be captured by the model and could potentially deteriorate its forecasting performance. However, the residuals are all roughly in the same range and there is no observation that stands out. Moreover, skewness and kurtosis of the residuals have reasonable values, although not normal, over the entire forecasting period. Therefore, there is no reason to deal with any observations separately.

5.3.2 Selected Variables and Factors

Figures 15 and 16 show the number of variables and factors selected by boosting with $M = 70$ and $\nu = 0.6$ and PCA, respectively. It can be seen that the number of explanatory variables is reduced to less than half, as less than 30 predictors are chosen during each estimation of the model. Sometimes even less than a third of all variables is selected. This shows that boosting

successfully reduced the dimension of the large set of predictors. However, when the number of predictors after boosting is too small, PCA fails to estimate the number of factors correctly. The spikes in Figure 16 correspond to drops in the number of selected variables in Figure 15. At these spikes, the number of factors estimated with PCA jumps to 10, which is the maximum number of factors that can be selected. All other forecasts are made with only one factor, which seems a reasonable result given the consistency of this number when disregarding the spikes in the graph. This is in line with the simulation results, as it was found that when $N = 20$ the hit rate of the IC_{p2} selection criterion falls below one and when $N = 10$ the estimated number of factors is often equal to $k_{max} = 10$ leading to a hit rate that is always 0.

Despite the likely failure of PCA to specify the underlying number of factors correctly when ten factors were estimated, this does not represent a problem for the SP2 forecasting method because boosting is applied a second time to obtain the coefficients of the factors in the diffusion index forecasting equation. As can be seen in Figure 16, boosting reduces the number of factors to at most four. However, in the cases when PCA estimated ten factors, boosting still estimates more than one factor with non-zero coefficients. Hence, there indeed seems to be more than one factor that was relevant for explaining GDP growth rates during these quarters. Therefore, the SP2 model can produce good forecasts even when the number of factors was overestimated.

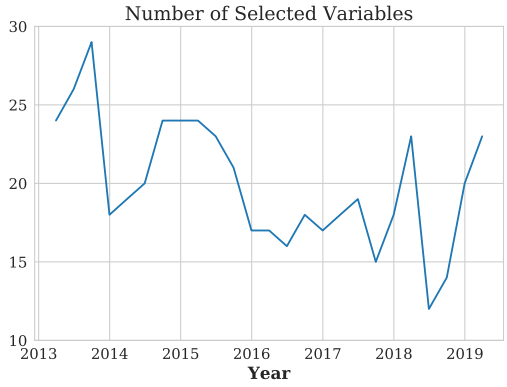


Figure 15: Boosting the predictors

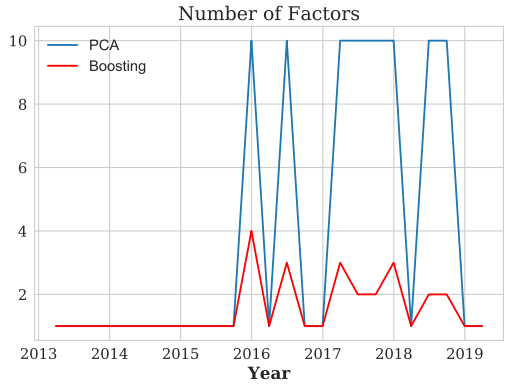


Figure 16: Boosting the factors

Table 5 provides a summary of the most frequently selected variables of the last forecast made by the SP2 model, excluding GDP growth itself. During the numerous boosting iterations, one specific variable can be selected more than once depending on its ability to explain GDP growth. All predictors that were chosen in more than one iteration of boosting are displayed below. When making the last point forecast for Q1 2019, 23 variables were pre-selected by boosting. Out of these variables, South Africa Mining Prices were selected most frequently by the boosting algorithm, namely a total of seven times. This supports the fact that South Africa's economy is still highly dependent on its mining industry, which contributes the largest share of

its exports. Mining prices are followed by the Lagging Composite Business Cycle Indicator, suggesting that there is some time dependency between South African GDP growth and its business cycle. South Africa Wholesale Retail Hotels Prices and South Africa Money Supply M1 were both selected three times. This is consistent with wholesale and retail trade and tourism being two of South Africa's most important economic sectors. As money supply is in the short run expected to positively correlate with GDP according to macroeconomic theory, it could also be a good predictor thereof. Overall, the variables selected multiple times by boosting all have a clear connection to South African GDP growth. Additionally, Table 11 in Appendix A.2 displays the loadings of the single factor estimated for obtaining the last point forecast.

Table 5: Variables that were most frequently selected by boosting

Frequency	Bloomberg Ticker	Description
7	SASGMINE Index	South Africa Mining SA Constant Prices
5	SACBLG Index	Composite Business Cycle Indicator - Lagging Indicator
3	SASGWRH Index	South Africa Wholesale Retail Hotels SA Constant Prices
3	SAMYM1 Index	South Africa Money Supply M1
2	SACTLMI Index	South Africa Current Account SA - Less Merchandise Imports
2	SASGCON Index	South Africa Construction SA Constant 2000 Prices
2	SACBCI Index	Composite Business Cycle Indicator - Coincident Indicator

These are explanatory variables selected in more than one boosting iteration with $M = 70$ and $\nu = 0.6$ when producing the last forecast. For a full list of all 62 variables, see Tables 7 and 8 in Appendix A.1.

5.4 Recurrent Neural Network

Lastly, the results of predicting South African GDP growth using a univariate RNN with LSTM architecture are provided in Table 6. The LSTM model was trained for varying numbers of epochs E with different numbers of neurons n in each configuration. When comparing MSFE ratios and DM test statistics of the 15 different combinations of network parameters, it becomes visible in Panel A that a model with $n = 100$ neurons trained for $E = 50$ epochs achieves the lowest MSFE ratio. However, for this configuration with a ratio of 0.750 the DM test fails to reject the null hypothesis of more accurate AR forecasts as the test statistic is only 0.953. There are several parameter configurations that lead to a DM value above 1, namely $n = 10$ and $E = 200$, $n = 20$ and $E = 200$, or $n = 50$ and $E = 100$. The LSTM network with $n = 10$ neurons trained $E = 200$ iterations performed particularly well, as it corresponds to the second lowest MSFE ratio of 0.802 and the highest test statistic $DM = 1.246$, which is only slightly below the 10% critical value of 1.282. Consequently, none of the investigated LSTM parameters lead to significantly better forecasts relative to the AR model. The LSTM results could not significantly improve upon the SP2 forecasts either, which can be seen in Panel B of Table 6. Table 12 in Appendix A.2 shows results for more parsimonious models in terms of the number of neurons.

Table 6: MSFE ratios for different LSTM parameters with DM test statistics in parentheses

E \ n	10		20		50		100		200	
PANEL A: LSTM network vs AR model										
50	1.319	(-0.725)	1.247	(-0.648)	0.826	(0.613)	0.750	(0.953)	0.868	(0.551)
100	0.968	(0.104)	0.898	(0.388)	0.831	(1.006)	0.845	(0.906)	0.901	(0.583)
200	0.802	(1.246)	0.856	(1.073)	0.893	(0.781)	0.893	(0.708)	0.865	(0.966)
PANEL B: LSTM network vs SP2 model										
50	1.411	(-0.802)	1.334	(-0.735)	0.430	(0.340)	0.391	(0.583)	0.928	(0.206)
100	1.036	(-0.088)	0.960	(0.106)	0.433	(0.315)	0.440	(0.268)	0.964	(0.101)
200	0.858	(0.406)	0.915	(0.233)	0.465	(0.122)	0.465	(0.122)	0.925	(0.205)

The ratios reported above are MSFE ratios of univariate RNNs with LSTM architecture relative to the AR and SP2 model. Each ratio corresponds to an LSTM model with $p = 4$ input lags and different numbers of neurons n trained for E epochs. Corresponding DM test statistics for the null hypothesis of superior benchmark model forecasts are shown in parentheses. Each LSTM configuration was used to predict the out-of-sample GDP growth rates only once. However, it should be noted that training a neural network is subject to randomness. Therefore, these results are not fully representative of the predictive ability of the LSTM models.

However, as discussed in Section 4.4, neural networks are very sensitive to the random initialization of their weight vectors. Therefore, the inability to reject the DM test for any of the LSTM configurations above could simply be due to the initial weights. For this reason, the best configuration is trained and evaluated on its predictions 10 times, leading to an average MSFE ratio of 0.783. Furthermore, the DM test rejected superior AR forecasts in 60% of the trials at the 10% significance level while higher accuracy of the SP2 predictions could only be rejected twice. The individual results for each trial are reported in Table 13 in Appendix A.2. It can be concluded that there is additional predictive value in non-linear LSTM networks relative to a linear AR model. Hence, there might be a non-linear relation between South African GDP growth and its previous values. However, on average the linear diffusion index model still produces better forecasts than the best univariate LSTM considered in this research. This can be explained by the additional predictors contained in the diffusion index combination model.

6 Discussion and Conclusion

In this paper, the SP2 combination model by Kim and Swanson (2018) was analysed in detail and compared to other machine learning and AR models in terms of its predictive power. It was found that this model, combining factor estimation and boosting, significantly improves forecast accuracy relative to an AR model not only for monthly U.S. data consistent with Kim and Swanson (2018), but also for the smaller quarterly South African dataset. A simulation study of datasets with different dimensions across time and variables supported this finding and showed

that the investigated combination model produces on average significantly better forecasts for data of different sizes. The gain in accuracy and the significance thereof increase with the number of factors present in the data. Therefore, the first sub-question was answered by concluding that the constructed forecasting model works properly for different regions and data frequencies.

Additionally, the predictive ability of univariate LSTM networks with different parameter configurations was compared to the AR benchmark and the SP2 model combining boosting and PCA. An empirical evaluation based on the South African macroeconomic dataset revealed that the LSTM model with ten neurons trained for 200 epochs on average leads to significantly better forecasts of South African GDP growth than the AR model, suggesting a non-linear relation of GDP growth with its previous values. However, the SP2 model, which includes multiple predictor series in addition to lagged GDP growth, could not be outperformed significantly. This answers the second sub-question, since forecast accuracy could be improved only compared to AR models.

However, the parameters of boosting and neural networks strongly influenced the above findings, as without finding a good configuration neither the SP2 model nor LSTM networks were able to yield significantly better forecasts than the AR model. Consequently, the general research question how machine learning techniques can be used to beat autoregressive models in predicting South African GDP growth could be answered. More specifically, applying boosting together with PCA and searching for appropriate boosting parameters represents the most suitable way to employ machine learning techniques to predict GDP growth.

However, one of the limitations of this research is the restricted number of South African macroeconomic variables available over a long enough time span to fully discover the potential of the applied methods, especially at quarterly data frequency. Furthermore, appropriate parameters of the machine learning models were found by using a simple linear grid search, which is far from optimal. In order to evaluate the best versions of each model, a more strategic parameter optimization routine such as cross-validation is required. Without knowing whether the chosen parameters lead to the best configuration of a certain method, no final conclusions can be drawn concerning the question which model provides the most accurate forecasts for a given dataset.

Considering that South Africa depends heavily on its exports, it might be beneficial for further research to include more global macroeconomic variables in the set of predictors to help improve forecast accuracy. However, first a larger number of observations over time is needed to enable adding more variables to the model. Another opportunity for future research comes with the application of neural networks for time series prediction. In addition to previous values of GDP growth rates, the LSTM network could take as input either the full set of predictors X or the factors F, leading to a multivariate network with potentially more predictive power.

References

Aron, J. and Muellbauer, J. (2002). Interest rate effects on output: evidence from a gdp forecasting model for south africa. *IMF Staff papers*, 49(1):185–213.

Aron, J. and Muellbauer, J. (2007). Review of monetary policy in south africa since 1994. *Journal of African Economies*, 16(5):705–744.

Bai, J. and Ng, S. (2002). Determining the number of factors in approximate factor models. *Econometrica*, 70(1):191–221.

Bai, J. and Ng, S. (2006). Confidence intervals for diffusion index forecasts and inference for factor-augmented regressions. *Econometrica*, 74(4):1133–1150.

Bai, J. and Ng, S. (2009). Boosting diffusion indices. *Journal of Applied Econometrics*, 24(4):607–629.

Bai, J., Ng, S., et al. (2008). Large dimensional factor analysis. *Foundations and Trends® in Econometrics*, 3(2):89–163.

Botha, B., de Jager, S., Ruch, F., and Steinbach, R. (2017). The quarterly projection model of the sarb. *South African Reserve Bank Working Paper WP/17/01*.

Brand South Africa (2018). SA's key economic sectors. Retrieved June 17, 2019, from `https://www.brandsouthafrica.com/investments-immigration/business/investing/economic-sectors-agricultural`.

Center for International Development at Harvard University (2017). The atlas of economic complexity. Retrieved June 5, 2019, from `http://atlas.cid.harvard.edu/explore/?country=246&partner=undefined&product=undefined&productClass=HS&startYear=undefined&target=Product&year=2017`.

Cepni, O., Guney, I., and Swanson, N. R. (2018). Forecasting and Nowcasting Emerging Market GDP Growth Rates: The Role of Latent Global Economic Policy Uncertainty and Macroeconomic Data Surprise Factors. *Available at SSRN 3298924*.

Cepni, O., Güney, I. E., and Swanson, N. R. (2019). Nowcasting and forecasting GDP in emerging markets using global financial and macroeconomic diffusion indexes. *International Journal of Forecasting*, 35(2):555–572.

Chow, G. C., Lin, A.-l., et al. (1971). *Best linear unbiased interpolation, distribution, and extrapolation of time series by related series*. Princeton University.

Dejager, S. (2017). Comparing the SARB's Quarterly Projection Model to the "Core" macroeconometric model. Occasional Bulletin of Economic Notes 8903, South African Reserve Bank. Retrieved June 17, 2019, from `https://econpapers-repec-org.eur.idm.oclc.org/RePEc:rbz:oboens:8903`.

Diebold, F. X. and Mariano, R. S. (1995). Comparing predictive accuracy. *Journal of Business & Economic Statistics*, 13:253–263.

Franses, P. H., Dijk, D. v., and Opschoor, A. (2014). *Time Series Models for Business and Economic Forecasting*. Cambridge University Press, 2 edition. doi:`10.1017/CBO9781139049894`.

Freund, Y. and Schapire, R. E. (1997). A decision-theoretic generalization of on-line learning and an application to boosting. *Journal of Computer and System Sciences*, 55(1):119–139.

Giannone, D., Reichlin, L., and Small, D. (2008). Nowcasting: The real-time informational content of macroeconomic data. *Journal of Monetary Economics*, 55(4):665–676.

Hochreiter, S. (1998). The vanishing gradient problem during learning recurrent neural nets and problem solutions. *International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems*, 6(02):107–116.

Hochreiter, S. and Schmidhuber, J. (1997). Long short-term memory. *Neural Computation*, 9(8):1735–1780.

Hornik, K., Stinchcombe, M., and White, H. (1989). Multilayer feedforward networks are universal approximators. *Neural Networks*, 2(5):359–366.

International Monetary Fund (2018). *World Economic Outlook: Growth Slowdown, Precarious Recovery*. International Monetary Fund.

Kim, H. H. and Swanson, N. R. (2014). Forecasting financial and macroeconomic variables using data reduction methods: New empirical evidence. *Journal of Econometrics*, 178:352–367.

Kim, H. H. and Swanson, N. R. (2018). Mining big data using parsimonious factor, machine learning, variable selection and shrinkage methods. *International Journal of Forecasting*, 34(2):339–354.

Kuan, C.-M. and Liu, T. (1995). Forecasting exchange rates using feedforward and recurrent neural networks. *Journal of Applied Econometrics*, 10(4):347–364.

McCulloch, W. S. and Pitts, W. (1943). A logical calculus of the ideas immanent in nervous activity. *The Bulletin of Mathematical Biophysics*, 5(4):115–133.

Olah, C. (2015). Understanding lstm networks. Retrieved June 20, 2019, from http://colah.github.io/posts/2015-08-Understanding-LSTMs/.

Ridgeway, G., Madigan, D., and Richardson, T. (1999). Boosting methodology for regression problems. In *AISTATS*.

Schapire, R. E. (1990). The strength of weak learnability. *Machine Learning*, 5(2):197–227.

Schwarz, G. et al. (1978). Estimating the dimension of a model. *The Annals of Statistics*, 6(2):461–464.

Stock, J. H. and Watson, M. W. (2002a). Forecasting using principal components from a large number of predictors. *Journal of the American Statistical Association*, 97(460):1167–1179.

Stock, J. H. and Watson, M. W. (2002b). Macroeconomic forecasting using diffusion indexes. *Journal of Business & Economic Statistics*, 20(2):147–162.

Stock, J. H. and Watson, M. W. (2012). Generalized shrinkage methods for forecasting using many predictors. *Journal of Business & Economic Statistics*, 30(4):481–493.

Svensson, L. E. (1997). Inflation forecast targeting: Implementing and monitoring inflation targets. *European Economic Review*, 41(6):1111–1146.

Swanson, N. R. and White, H. (1997). A model selection approach to real-time macroeconomic forecasting using linear models and artificial neural networks. *Review of Economics and Statistics*, 79(4):540–550.

Tkacz, G. (2001). Neural network forecasting of Canadian GDP growth. *International Journal of Forecasting*, 17(1):57–69.

Toyana, M. (2019). IMF trims South Africa's economic growth forecast to 1.2 percent. *Reuters*. Retrieved June 17, 2019, from https://www.reuters.com/article/safrica-economy-imf/imf-trims-south-africas-economic-growth-forecast-to-1-2-percent-idUSL8N21R3TV.

Werbos, P. J. et al. (1990). Backpropagation through time: what it does and how to do it. *Proceedings of the IEEE*, 78(10):1550–1560.

A Appendix

A.1 Data

Table 7: Subset of South African predictors according to Cepni et al. (2019)

	Bloomberg Ticker	Description
1	SACWC Index	South Africa Consumer Confidence
2	SACWE Index	South Africa Consumer Confidence Economic Position in Next 12m
3	SACWF Index	South Africa Consumer Confidence Financial Position During Next 12m
4	SACTLVL Index	South Africa Current Account SA
5	SACTMEX Index	South Africa Current Account SA - Merchandise Exports Free on Board
6	SACTGEX Index	South Africa Current Account SA - Net Gold Exports
7	SACTLMI Index	South Africa Current Account SA - Less Merchandise Imports
8	SACTCTR Index	South Africa Current Account SA - Current Transfers Net Receipts
9	SACUI Index	South Africa Utilization of Production Capacity
10	SABTHDIQ Index	South Africa Household Debt to Disposable Income of Households
11	SAGNDISA Index	South Africa Nominal Household Disposable Income SA
12	SADXFCFR Index	South Africa Real GDP Gross Fixed Capital Formation SA
13	SASGAGR Index	South Africa Agriculture SA Constant Prices
14	SASGMINE Index	South Africa Mining SA Constant Prices
15	SASGMANU Index	South Africa Manufacturing SA Constant Prices
16	SASGELEC Index	South Africa Electricity SA Constant Prices
17	SASGCON Index	South Africa Construction SA Constant 2000 prices
18	SASGWRH Index	South Africa Wholesale Retail Hotels SA Constant Prices
19	SADXRGSA Index	South Africa Real GDP Expenditure on GDP
20	SACSPSTO Index	SA Recorded Building Plans Total SA
21	SACSPSRB Index	SA Recorded Building Plans Residentual Buildings SA
22	SACSPSNR Index	SA Recorded Building Plans Non-Residentual Buildings SA
23	SACSPSAA Index	SA Recorded Building Plans Additions and Alterations SA
24	SACSCSTO Index	SA Completed Buildings Recorded Total SA
25	SACSCSRB Index	SA Completed Buildings Recorded Residentual Buildings SA
26	SACSCSNR Index	SA Completed Buildings Recorded Non-Residentual Buildings SA
27	SACSCSAA Index	SA Completed Buildings Recorded Additions and Alterations SA
28	ZAR Curncy	USDZAR Spot Exchange Rate - Price of 1 USD in ZAR
29	GBPZAR Curncy	GBPZAR Spot Exchange Rate - Price of 1 GBP in ZAR
30	JPYZAR Curncy	JPYZAR Spot Exchange Rate - Price of 1 JPY in ZAR
31	TRYZAR Curncy	TRYZAR Spot Exchange Rate - Price of 1 TRY in ZAR
32	BISBZAR Index	South Africa Real Effective Exchange Rate Broad
33	TOP40 Index	FTSE/JSE Africa Top40 Tradeable Index
34	JFINX Index	FTSE/JSE Africa Financials Index
35	JBIND Index	FTSE/JSE Africa Basic Materials Index
36	JGIND Index	FTSE/JSE Africa Industrials Index
37	JGOLD Index	FTSE/JSE Africa Gold Mining Index
38	JALSH Index	FTSE/JSE Africa All Share Index
39	SACEI Index	South Africa Private Credit Extension
40	SACEINV Index	South Africa Private Credit Extension Investments

Table 8: Subset of South African predictors according to Cepni et al. (2019) (continued)

	Bloomberg Ticker	Description
41	SACEMORT Index	South Africa Private Credit Extension Mortgage Advances
42	SACELEAS Index	South Africa Private Credit Extension Leasing Finance
43	SACELOAN Index	South Africa Private Credit Extension Total Loans and Advances
44	SACESALE Index	South Africa Private Credit Extension Installment Sales Credit
45	SACEHOUS Index	South Africa Private Credit Extension Of Which To Households
46	SAMYSAM3 Index	South Africa Money Supply M3 Seasonally Adjusted
47	SAMYM1 Index	South Africa Money Supply M1
48	SAMYM2 Index	South Africa Money Supply M2
49	SAMYM0 Index	South Africa Money Supply M0
50	199.055 Index	IMF South Africa Foreign Exchange Reserves in Millions of USD
51	SACPI Index	South Africa CPI 2012=100
52	SACBLI Index	Composite Business Cycle Indicator - Leading Indicator
53	SACBLG Index	Composite Business Cycle Indicator - Lagging Indicator
54	SACBCI Index	Composite Business Cycle Indicator - Coincident Indicator
55	SANOFP$ Index	South Africa Net Open Foreign Currency Position
56	SATBAL Index	South Africa Trade Balance Incl Oil Arms & Bullion
57	SATBEX Index	South Africa Trade Balance Exports Incl Oil Arms & Bullion
58	SATBIM Index	South Africa Trade Balance Imports Incl Oil Arms & Bullion
59	SAMSTGSA Index	South Africa Mining Sales Total Including Gold SA
60	SAMPGDSY Index	South Africa Mining Production Volume Gold SA YoY
61	SAMPTTSY Index	South Africa Mining Production Volume Total Inc Gold SA YoY
62	EHGDZA Index	South Africa Real GDP (Annual YoY %)

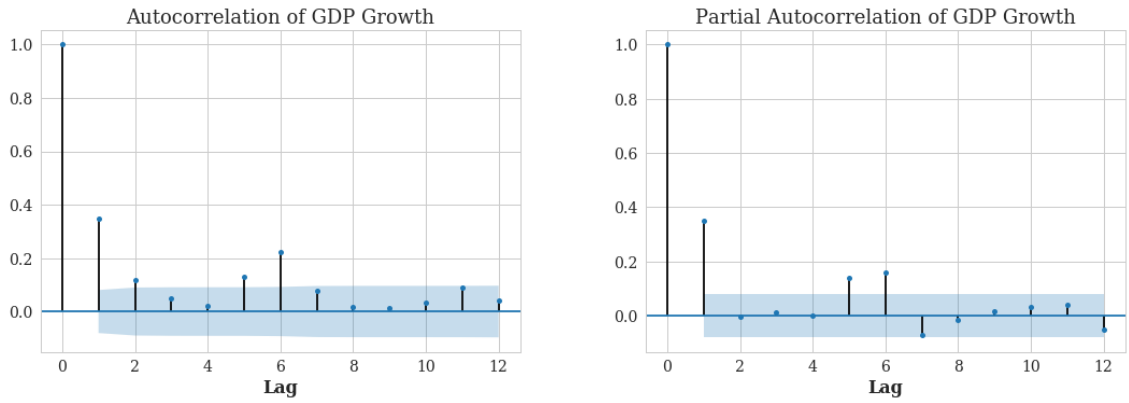


Figure 18: Autocorrelation functions of monthly U.S. GDP growth

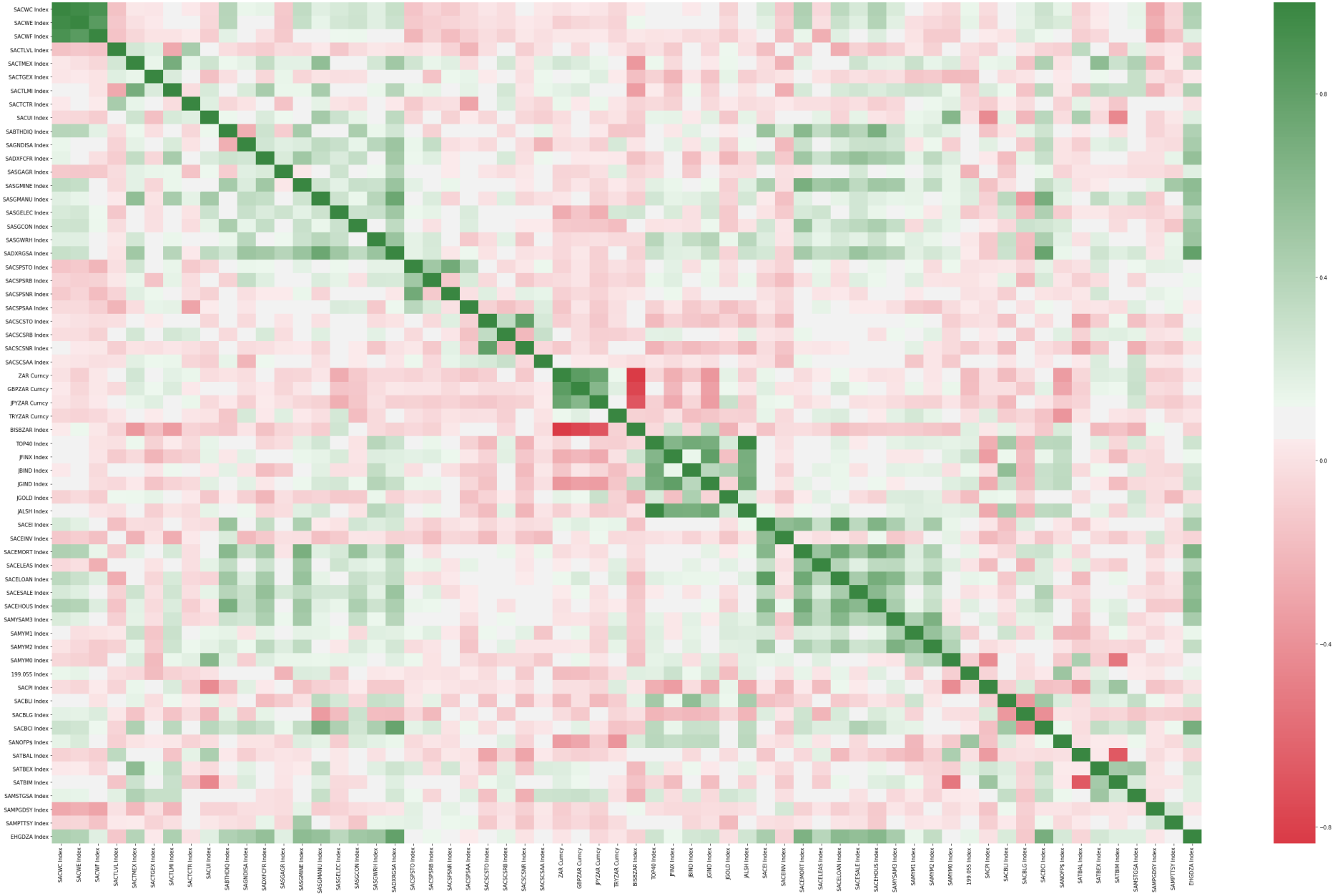


Figure 17: Heatmap depicting the correlations of lagged predictors with South African GDP growth

A.2 Supplementary Results

Table 9: Estimated number of factors for different dimensions of the data with hit rates in parentheses

T \ N	10		20		50		100		200	
PANEL A: $k = 1$										
10	9.349	(0.000)	9.011	(0.000)	9.002	(0.000)	9.000	(0.000)	9.001	(0.000)
20	10.000	(0.000)	1.198	(0.975)	1.009	(0.999)	1.000	(1.000)	1.000	(1.000)
50	10.000	(0.000)	1.075	(0.990)	1.000	(1.000)	1.000	(1.000)	1.000	(1.000)
100	10.000	(0.000)	1.194	(0.975)	1.000	(1.000)	1.000	(1.000)	1.000	(1.000)
200	10.000	(0.000)	1.263	(0.967)	1.000	(1.000)	1.000	(1.000)	1.000	(1.000)
500	10.000	(0.000)	1.392	(0.954)	1.000	(1.000)	1.000	(1.000)	1.000	(1.000)
1000	10.000	(0.000)	1.373	(0.957)	1.000	(1.000)	1.000	(1.000)	1.000	(1.000)
PANEL B: $k = 3$										
10	9.283	(0.000)	9.004	(0.000)	9.001	(0.000)	9.000	(0.000)	9.000	(0.000)
20	10.000	(0.000)	2.237	(0.246)	2.643	(0.689)	2.833	(0.845)	2.913	(0.914)
50	10.000	(0.000)	2.770	(0.775)	2.976	(0.977)	3.000	(1.000)	3.000	(1.000)
100	10.000	(0.000)	2.952	(0.930)	3.000	(1.000)	3.000	(1.000)	3.000	(1.000)
200	10.000	(0.000)	2.997	(0.971)	3.000	(1.000)	3.000	(1.000)	3.000	(1.000)
500	10.000	(0.000)	3.001	(0.989)	3.000	(1.000)	3.000	(1.000)	3.000	(1.000)
1000	10.000	(0.000)	2.996	(0.988)	3.000	(1.000)	3.000	(1.000)	3.000	(1.000)
PANEL C: $k = 5$										
10	9.264	(0.000)	9.001	(0.000)	9.000	(0.000)	9.000	(0.000)	9.000	(0.000)
20	10.000	(0.000)	2.246	(0.011)	2.732	(0.069)	3.460	(0.176)	3.818	(0.233)
50	10.000	(0.000)	3.179	(0.146)	3.935	(0.308)	4.850	(0.854)	4.981	(0.981)
100	10.000	(0.000)	4.159	(0.444)	4.907	(0.912)	5.000	(1.000)	5.000	(1.000)
200	10.000	(0.000)	4.489	(0.589)	4.997	(0.997)	5.000	(1.000)	5.000	(1.000)
500	10.000	(0.000)	4.651	(0.685)	5.000	(1.000)	5.000	(1.000)	5.000	(1.000)
1000	10.000	(0.000)	4.744	(0.751)	5.000	(1.000)	5.000	(1.000)	5.000	(1.000)

The values in this Table are averages over 1000 simulation replications. Each panel represents a different number of k underlying factors in the DGP. The estimated number of factors $\hat{k}$ is selected based on the IC_{p2} criterion with $k_{max} = 10$ as discussed in Section 4.1.2.

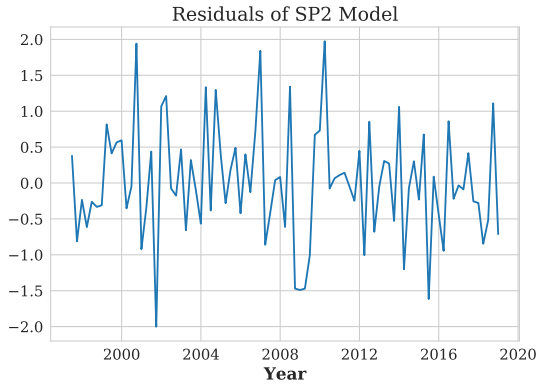


Figure 19: In-sample residuals for last point forecast

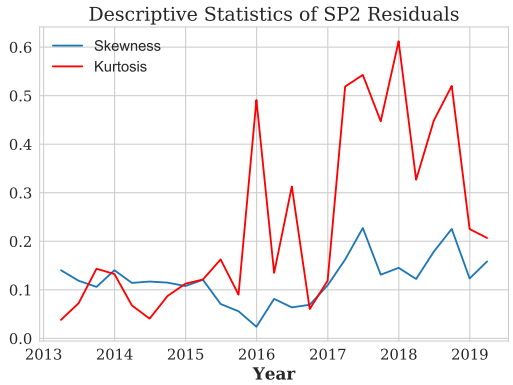


Figure 20: In-sample descriptive statistics of residuals for each point forecast

Table 10: MSFE ratios for different dimensions of the data with numbers of rejected DM tests in parentheses

T \ N	10		20		50		100		200	
PANEL A: $k = 1$										
50	0.809	(37)	0.828	(26)	0.720	(38)	0.671	(63)	0.688	(52)
100	0.756	(42)	0.747	(41)	0.795	(28)	0.760	(38)	0.702	(49)
200	0.756	(51)	0.761	(40)	0.769	(34)	0.748	(41)	0.768	(39)
500	0.757	(42)	0.748	(41)	0.732	(49)	0.734	(40)	0.750	(38)
1000	0.734	(52)	0.751	(46)	0.748	(41)	0.742	(41)	0.759	(37)
PANEL B: $k = 3$										
50	0.838	(29)	0.806	(24)	0.755	(39)	0.716	(42)	0.705	(47)
100	0.774	(42)	0.750	(44)	0.752	(38)	0.709	(54)	0.702	(49)
200	0.756	(45)	0.748	(47)	0.727	(40)	0.737	(39)	0.725	(41)
500	0.760	(37)	0.713	(48)	0.720	(47)	0.727	(40)	0.724	(44)
1000	0.774	(36)	0.718	(52)	0.690	(51)	0.702	(49)	0.700	(48)
PANEL C: $k = 5$										
50	0.905	(19)	0.859	(21)	0.791	(35)	0.790	(40)	0.803	(29)
100	0.819	(31)	0.783	(36)	0.738	(41)	0.716	(41)	0.703	(50)
200	0.790	(35)	0.731	(50)	0.734	(43)	0.717	(43)	0.702	(52)
500	0.767	(43)	0.726	(52)	0.710	(45)	0.707	(46)	0.693	(49)
1000	0.771	(42)	0.732	(45)	0.703	(54)	0.672	(51)	0.675	(56)

The values in this Table are averages over 100 simulation replications. Each panel represents a different number of k underlying factors in the DGP. The estimated number of factors $\hat{k}$ is selected based on the IC_{p2} criterion with $k_{max} = 10$ as discussed in Section 4.1.2. The MSFE ratios are based on forecast errors of the SP2 model relative to the AR model. For the SP2 model, the boosting parameters are $M = 50$ and $\nu = 0.5$. The number of rejected Diebold-Mariano (DM) tests is based on the 5% one-sided standard normal critical value of 1.645.

Table 11: Factor loadings of the chosen factor during the last forecast window

Loading	Variable Description
1.874	South Africa Real GDP (Annual YoY %)
1.710	Composite Business Cycle Indicator - Coincident Indicator
1.556	South Africa Mining SA Constant Prices
1.361	South Africa Real GDP Gross Fixed Capital Formation SA
1.306	South Africa Electricity SA Constant Prices
1.258	South Africa Wholesale Retail Hotels SA Constant Prices
1.158	South Africa Construction SA Constant 2000 prices
1.111	South Africa Private Credit Extension Leasing Finance
1.053	South Africa Current Account SA - Less Merchandise Imports
0.964	South Africa Consumer Confidence
0.939	South Africa Private Credit Extension
0.835	South Africa Utilization of Production Capacity
0.744	SA Recorded Building Plans Residentual Buildings SA
0.736	South Africa Money Supply M1
0.718	Composite Business Cycle Indicator - Leading Indicator
0.671	FTSE/JSE Africa Basic Materials Index
0.405	SA Recorded Building Plans Additions and Alterations SA
0.391	South Africa Agriculture SA Constant Prices
0.370	South Africa Consumer Confidence Financial Position During Next 12m
-0.064	South Africa Mining Production Volume Gold SA YoY
-0.246	JPYZAR Spot Exchange Rate - Price of 1 JPY in ZAR
-0.327	Composite Business Cycle Indicator - Lagging Indicator
-0.351	South Africa CPI 2012=100

Table 12: MSFE ratios for different LSTM parameters with DM test statistics in parentheses

E \ n	1		3		5	
PANEL A: LSTM network vs AR model						
50	3.678	(-3.170)	3.261	(-3.215)	1.106	(-0.234)
100	2.831	(-2.572)	1.313	(-0.579)	1.103	(-0.304)
200	1.129	(-0.353)	1.159	(-0.436)	0.891	(0.463)
PANEL B: LSTM network vs SP2 model						
50	3.934	(-3.150)	3.488	(-3.329)	1.183	(-0.390)
100	3.028	(-2.689)	1.405	(-0.759)	1.180	(-0.396)
200	1.208	(-0.488)	1.239	(-0.535)	0.953	(0.127)

The ratios reported above are MSFE ratios of univariate RNNs with LSTM architecture relative to the AR and SP2 model. Each ratio corresponds to an LSTM model with $p = 4$ input lags and different numbers of neurons n trained for E epochs. Corresponding DM test statistics for the null hypothesis of superior benchmark model forecasts are shown in parentheses. Each LSTM configuration was used to predict the out-of-sample GDP growth rates only once. However, it should be noted that training a neural network is subject to randomness. Therefore, these results are not fully representative of the predictive ability of the LSTM models.

Table 13: MSFE ratios of LSTM model with DM test statistics in parentheses

Trial	1	2	3	4	5	6	7	8	9	10
LSTM/AR	0.745*	0.856	0.704*	0.790*	0.725*	0.856	0.768**	0.755	0.767**	0.863
	(1.420)	(0.902)	(1.633)	(1.284)	(1.654)	(0.797)	(1.775)	(1.254)	(1.726)	(0.818)
LSTM/SP2	0.797	0.916	0.753*	0.845	0.775	0.915	0.821*	0.808	0.821	0.923
	(1.082)	(0.491)	(1.319)	(0.906)	(1.264)	(0.442)	(1.302)	(0.943)	(1.258)	(0.428)

This Table shows MSFE ratios of a univariate RNN with LSTM architecture relative to the AR and SP2 model. All ratios correspond to the LSTM model with $p = 4$ input lags and $n = 10$ neurons trained for $E = 200$ epochs. This LSTM configuration was used to predict the out-of-sample GDP growth rates ten times with different random weight initialization. Corresponding DM test statistics for the null hypothesis of superior benchmark model forecasts are shown in parentheses. For entries marked with *, ** and *** the DM test rejected higher accuracy of the AR or SP2 forecasts at the 10%, 5% and 1% significance level, respectively.

A.3 Python Code

```
# -*- coding: utf-8 -*-
"""
Created on Tue May 14 12:34:41 2019

@author: plagl
"""

import os
import numpy as np
import pandas as pd
from pandas.plotting import lag_plot
from statsmodels.graphics.tsaplots import plot_acf, plot_pacf
from statsmodels.tsa.stattools import adfuller
from scipy import stats
import matplotlib.pyplot as plt
import seaborn as sns
myfont = {'fontname':'Serif'}



def read_data(filename):
    # Get the current location of the file
    script_path = os.getcwd()
    os.chdir( script_path )
    location = './data/raw_data/' + filename

    # Load spreadsheet into dataframe
    df = pd.read_excel(location, header=0, index_col=0)

    if 'Transformation' in df.index: # U.S. dataset
        tcode = df.iloc[0,:]
        df = df.drop('Transformation', axis=0)

    df.index.names = ['Date']
    df.index = pd.to_datetime(df.index)

    if filename == 'south_africa_quarterly.xlsx':
        # Reduce sample to limit NA values for South African data
        df = df.loc['1996-03-31':, :] #(T, N) = (93, 62)
        df = df.dropna(axis = 1)
        df = make_stationary(df)
        df = df.loc['1996-06-30':, :]
        #df.to_csv('./data/raw_data/south_africa_quarterly_1998.csv')
    else:
        # Apply transformation codes for U.S. data
        df = transform(df, tcode)
        # Adjust sample to cover the same range as Kim & Swanson
        df = df.loc['1960-01-01':'2009-05-01', :]
    return df


def transform(data, tcodes):
    # Transforms the data to stationarity by differencing or log differencing
    # Using the transformation codes coming with the U.S. dataset
    transformed_data = data.copy()
    for i in range(0, len(data.columns)):
        if tcodes[i] == 2:
            transformed_data.iloc[:, i] = data.iloc[:, i].diff(periods=1)
        elif tcodes[i] == 3:
            transformed_data.iloc[:, i] = data.iloc[:, i].diff(periods=2)
        elif tcodes[i] == 4:
```

```
            transformed_data.iloc[:, i] = np.log(data.iloc[:, i])
        elif tcodes[i] == 5:
            transformed_data.iloc[:, i] = (np.log(data.iloc[:, i])).diff(periods
    =1)
        elif tcodes[i] == 6:
            transformed_data.iloc[:, i] = (np.log(data.iloc[:, i])).diff(periods
    =1)
    return transformed_data


def make_stationary(data):
    # Tests if time series is stationary using augmented Dickey-Fuller test
    # If time series is not stationary it is transformed to be stationary
    X = data.copy()
    T, N = X.shape
    df_test = [1]*N
    p_values = np.ones((N,1))
    tcode = np.ones((N,1))

    for i in range(0, N):
        x = X.iloc[:, i]
        df_test[i] = adfuller(x, maxlag=12, autolag='BIC')
        p_values[i] = df_test[i][1]
        if p_values[i]>0.1 and (x>=0).all():
            tcode[i] = 5
        elif p_values[i]>0.1:
            tcode[i] = 2

    X = transform(X, tcode)
    return X


def compute_descriptives(df):
    data = df
    minimum = data.min()
    maximum = data.max()
    mean = data.mean()
    std = data.std()
    variance = data.var()
    skewness = data.skew()
    kurtosis = data.kurtosis()
    jb = stats.jarque_bera(data)

    # Save desctiptives in dataframe
    rows = ['Min', 'Max', 'Mean', 'Std', 'Var', 'Skew', 'Kurt', 'JB']
    descriptives = pd.DataFrame(index=rows)
    descriptives.loc['Min'] = minimum
    descriptives.loc['Max'] = maximum
    descriptives.loc['Mean'] = mean
    descriptives.loc['Std'] = std
    descriptives.loc['Var'] = variance
    descriptives.loc['Skew'] = skewness
    descriptives.loc['Kurt'] = kurtosis
    descriptives.loc['JB'] = jb

    print('Min:', minimum)
    print('Max:', maximum)
    print('Mean:', mean)
    print('Std:', std)
    print('Var:', variance)
    print('Skew:', skewness)
    print('Kurt:', kurtosis)
    print('JB:', jb)
```

```
123     return descriptives
124
125
126 def plot(data):
127     # plot GDP/EHGDZA Index
128     if 'EHGDZA Index' in data.columns:
129         values = data["EHGDZA Index"]
130     else:
131         values = data["'GDP'"]*100
132
133     # Simple line graph
134     plt.figure(figsize=(8,6))
135     plt.style.use(['seaborn-whitegrid', 'seaborn-notebook'])
136     plt.xlabel('Year', **myfont, fontsize=16, fontweight='bold')
137     plt.title('Quarterly GDP Growth in Percent', **myfont, fontsize=18)
138     plt.xticks(**myfont, fontsize=14)
139     plt.yticks(**myfont, fontsize=14)
140     plt.plot(values)
141     plt.savefig('./figures/sa_graph.eps', format='eps', dpi=1000)
142     plt.show()
143
144     # Scatter plot of the lags
145     lag_plot(values)
146     plt.title('Lagged GDP Growth in Percent', **myfont, fontsize=18)
147     plt.xlabel('y(t-1)', **myfont, fontsize=16, fontweight='bold')
148     plt.ylabel('y(t)', **myfont, fontsize=16, fontweight='bold')
149     plt.xticks(**myfont,fontsize=14)
150     plt.yticks(**myfont,fontsize=14)
151     plt.savefig('./figures/sa_scatter.eps', format='eps', dpi=1000)
152     plt.show()
153
154     # Autocorrelation plot
155     plot_acf(values, lags=12)
156     plt.title('Autocorrelation of GDP Growth', **myfont, fontsize=18)
157     plt.xlabel('Lag', **myfont, fontsize=16, fontweight='bold')
158     plt.xticks(**myfont,fontsize=14)
159     plt.yticks(**myfont,fontsize=14)
160     plt.show()
161
162     # Partial autocorrelation plot
163     plot_pacf(values, lags=12)
164     plt.title('Partial Autocorrelation of GDP Growth', **myfont, fontsize=18)
165     plt.xlabel('Lag', **myfont, fontsize=16, fontweight='bold')
166     plt.xticks(**myfont,fontsize=14)
167     plt.yticks(**myfont,fontsize=14)
168     plt.show()
169
170     return values
171
172
173 def heatmap(data):
174     # Replace all variables except GDP growth by lagged series
175     lags = np.roll(data.values[:, :-1], 1, axis=0)
176     lag_data = data.copy()
177     lag_data.iloc[:, :-1] = lags
178
179     # Create correlation heatmap
180     corr = lag_data.iloc[1:, :].corr()
181     plt.figure(figsize=(50,30))
182     sns.heatmap(corr, xticklabels=corr.columns, yticklabels=corr.columns, \
183                 cmap=sns.diverging_palette(10, 130, as_cmap=True))
184     plt.show()
185     return lag_data
```

```

# Testing the functions
file1 = 'united_states.xlsx'
file2 = 'south_africa_quarterly.xlsx'
data = read_data(file2)
array = data.values
X = array[:, :-1]
y = array[:, -1]

percentage_growth = plot(data)
compute_descriptives(percentage_growth)
lag_data = heatmap(data)
```

Listing 1: data.py

```
# -*- coding: utf-8 -*-
"""
Created on Tue May 14 12:34:55 2019

@author: plagl
"""

import numpy as np
import statsmodels.api as sm
from math import log, sqrt
from statsmodels.tsa.tsatools import lagmat
from scipy.stats import kurtosis, skew, jarque_bera



def standardize(data):
    mu = np.mean(data, axis=0)
    sigma = np.std(data, axis=0)
    return (data-mu)/sigma


def DM_test(e1, e2):
    T = len(e1)
    differential = e1**2 - e2**2
    mean = differential.mean()
    std = differential.std()
    statistic = mean / (std / sqrt(T))
    return statistic


def check_residuals(e):
    # Check for outliers and skewness of residuals
    mean = np.mean(e)
    variance = np.var(e)
    skewness = skew(e)
    kurtos = kurtosis(e)
    jb = jarque_bera(e)
    return mean, variance, skewness, kurtos, jb[1]


def ar(Y, max_p):
    y = Y.copy()
    SIC = np.zeros((max_p+1,1))
    y_lags, y_target = lagmat(y, maxlag = max_p, trim='forward', original='sep')
    y_lags = sm.add_constant(y_lags) # adding a constant to explanatory vars

    # Cutting off the first max_p observations
    y_lags = y_lags[max_p:,:]
```

```
    y_target = y_target[max_p:]

    for p in range(0, max_p+1):
        res = sm.OLS(y_target, y_lags[:, :p+1]).fit() # fit OLS model
        SIC[p] = res.bic

    # Fit AR model with optimal number of lags p_star, which minimizes the SIC
    p_star = np.argmin(SIC)
    y_lags = y_lags[:, :p_star+1]
    result = sm.OLS(y_target, y_lags).fit()
    beta = (result.params).reshape(p_star+1, 1)
    return result, y_target, y_lags, beta, p_star


def boost(X_matrix, Z, v, M):
    # Dimensions of X matrix
    X = X_matrix.copy()
    T, N = X.shape

    # Setting up some variables
    sel_x = np.zeros((N,M+1)) # selects a subset of X
    beta = np.zeros((N,M+1)) # regression coefficient of selected X
    B = np.zeros((T,T)) # should be ones acc to paper!
    IC = np.zeros((M,1))

    # Initializing phi for iteration 0
    z = Z.copy()
    z = z.reshape(T,1)
    phi = np.ones((T,1)) * z.mean()

    for m in range(0, M):
        u = z - phi.reshape(T,1) # compute the current residual
        b = np.zeros((N,1))
        SSR = np.zeros((N,1))

        for i in range(0, N):
            x = X[:,i].reshape(T,1) # get ith predictor
            results = sm.OLS(u, x).fit() # fit OLS model
            b[i] = results.params
            SSR[i] = results.ssr

        # Updating phi
        i_star = np.argmin(SSR)
        x_star = X[:,i_star].reshape(T,1)
        u_hat = (x_star * b[i_star]).reshape(T,1)
        phi = phi + v * u_hat

        # Selecting variable x_star in this iteration
        indicator = np.zeros((N,1))
        indicator[i_star] = 1
        sel_x[:,[m+1]] = sel_x[:,[m]] + indicator # using double [] to retain
    dimensions
        beta[:,[m+1]] = beta[:,[m]] + v * b[i_star] * indicator.reshape(N,1)

        # Computing information criterion for optimal stopping iteration
        P = x_star @ np.linalg.inv((x_star.T @ x_star)) @ x_star.T
        B = B + v * (P @ (np.identity(T) - B))
        dof = np.trace(B)
        sigma = (z - phi).T @ (z - phi)
        IC[m] = log(sigma) + ((log(T) * dof) / T)

    # Choosing stopping iteration m_star that minimizes IC
    m_star = np.argmin(IC)
```

```
    beta_star = beta[:,m_star+1]
    sel_x_star = sel_x[:,m_star+1]
    return sel_x_star, beta_star, IC, m_star+1


def estimate_factors(X_matrix, max_k):
    # Scale data to have mean 0 and variance 1
    X = X_matrix.copy()
    X = standardize(X)

    # Dimensions of X matrix
    T, N = X.shape
    PC = np.zeros((max_k,1)) # information criterion to select number of
    components

    # Compute eigenvalues and eigenvectors of the covariance matrix of X
    covariance = np.cov(X, rowvar=False, bias=True)
    eigenvals, eigenvecs = np.linalg.eigh(covariance, UPLO='U')

    # Test if all eigenvectors have unit length
    for e in eigenvecs:
        np.testing.assert_array_almost_equal(1.0, np.linalg.norm(e))

    # Sort eigenvalues in decreasing order
    indices = np.argsort(eigenvals)[::-1]
    eigenvals = eigenvals[indices]
    # Sort eigenvectors accordingly
    eigenvecs = eigenvecs[:,indices]

    for k in range(1, max_k+1):
        # Select the first k eigenvectors as coefficients of the factors
        loadings = eigenvecs[:, :k]  * sqrt(N)
        # Transform X using these eigenvectors
        factors = (X @ loadings) / N
        # Compute selection criterion
        resid = X - factors @ loadings.T
        sigma = np.diagonal(resid.T @ resid) / T
        V = np.sum(sigma) / N
        penalty = ((float(T+N) / (T*N)) * log(min(T,N)))
        PC[k-1] = log(V) + k * penalty

    # Choosing number of factors k_star that minimizes PC
    k_star = np.argmin(PC)+1
    loadings = eigenvecs[:, :k_star] * sqrt(N)
    factors = (X @ loadings) / N
    return factors, loadings.T, k_star, PC
```

Listing 2: model.py

```
# -*- coding: utf-8 -*-
"""
Created on Tue May 14 12:35:10 2019

@author: plagl
"""

import numpy as np
from data import read_data
from model import ar, boost, estimate_factors, DM_test, check_residuals
from sklearn.metrics import mean_squared_error
import matplotlib.pyplot as plt
myfont = {'fontname':'Serif'}


```

```
def recursive_forecasts(data, M=50, v=0.5):
    # Setting up parameters
    max_factors = 20
    max_p = 4
    s = 156
    h = 12

    # Extracting X and y variables
    array = data.values
    X = array[:, :]
    T, N = X.shape
    y = array[:, -1].reshape(T,1) # GDP is last variable

    P = T-s-h-max_p+1
    y_hat_cbst = np.zeros((P,1))
    y_hat_AR = np.zeros((P,1))
    y_true = y[s+h+max_p-1:].reshape(P,1)

    k_hat = np.zeros((P,1))
    p_hat = np.zeros((P,1))
    m_hat = np.zeros((P,2))
    n_hat = np.zeros((P,1))
    resid_descr = np.zeros((P,5))

    for i in range(0, P):
        print(i)
        y_in = y[h-1 : i+s+h+max_p-1].copy()
        x_in = X[max_p+h-2 : i+s+h+max_p-1].copy()

        # Fit benchmark AR(p) model
        res, y_target, y_lags, beta_AR, p = ar(y_in, max_p)
        y_lags_current = np.insert(y_in[:-p-1:-1], 0, 1.0) # add constant
        y_hat_AR[i]  = y_lags_current @ beta_AR
        p_hat[i] = p
        z_in = y_target - res.predict(y_lags).reshape(len(y_target),1)

        # Fit the model of specification type 2
        # 1) pre-select X variables with boosting
        sel_X, beta_X, IC1, m1  = boost(x_in[:-1,:], z_in, v, M) # maybe use z_in
    here
        X_selected = x_in[:, sel_X>0]
        n_hat[i] = np.size(X_selected, 1)
        m_hat[i, 0] = m1

        # 2) Only apply PCA if at least max_factors X variables were selected
        if np.size(X_selected, 1) >= max_factors:
            factors, loadings, k, pc = estimate_factors(X_selected, max_factors)
        elif np.size(X_selected, 1) > 0:
            factors = X_selected
            loadings = 0
            k = 0
        else:
            factors, loadings, k, pc = estimate_factors(x_in, max_factors)
        k_hat[i] = k

        # 3) Use boosting to estimate factor coefficient beta_F
        sel_F, beta_F, IC2, m2 = boost(factors[:-1,:], z_in, v, M) # best IC
    should -6.3482
        m_hat[i, 1] = m2

        # Fitting model to obtain residuals
        y_fit = y_lags @ beta_AR + factors[:-1,:] @ beta_F.reshape(len(beta_F),1)
        resid = y_target - y_fit
```

```
        resid_descr[[i], 0], resid_descr[[i], 1], resid_descr[[i], 2],resid_descr
    [[i], 3], resid_descr[[i], 4] = check_residuals(resid)

        # Compute diffusion index forecast
        y_hat_cbst[i] = y_lags_current @ beta_AR + factors[-1,:] @ beta_F.reshape(
    len(beta_F),1)


    # Compute MSE and MSE ratio
    mse_AR = mean_squared_error(y_true, y_hat_AR)
    mse_cbst = mean_squared_error(y_true, y_hat_cbst)
    mse_ratio = mse_cbst / mse_AR
    error_AR = y_true - y_hat_AR
    error_cbst = y_true - y_hat_cbst
    dm_stat = DM_test(error_AR, error_cbst)
    print(dm_stat)

    # Plotting AR forecasts vs true values
    dates = data.index[s+h+max_p-1:]
    plt.style.use(['seaborn-whitegrid', 'seaborn-notebook'])
    plt.xlabel('Year', **myfont, fontsize=16, fontweight='bold')
    plt.title('Monthly GDP Growth', **myfont, fontsize=18)
    plt.xticks(**myfont, fontsize=14)
    plt.yticks(**myfont, fontsize=14)
    plt.plot(dates, y_true)
    plt.plot(dates, y_hat_AR, 'r')
    plt.legend(['True Values', 'Forecasts'], fontsize=14)
    plt.savefig('./figures/us_forecasts_ar.eps', format='eps', dpi=1000)
    plt.show()
    print(mse_AR)

    # Plotting diffusion index forecasts vs true values
    plt.style.use(['seaborn-whitegrid', 'seaborn-notebook'])
    plt.xlabel('Year', **myfont, fontsize=16, fontweight='bold')
    plt.title('Monthly GDP Growth', **myfont, fontsize=18)
    plt.xticks(**myfont, fontsize=14)
    plt.yticks(**myfont, fontsize=14)
    plt.plot(dates, y_true)
    plt.plot(dates, y_hat_cbst, 'r')
    plt.legend(['True Values', 'Forecasts'], fontsize=14)
    plt.savefig('./figures/us_forecasts_cbst.eps', format='eps', dpi=1000)
    plt.show()
    print(mse_cbst)

    # Plotting descriptives of residuals
    plt.style.use(['seaborn-whitegrid', 'seaborn-notebook'])
    plt.xlabel('Year', **myfont, fontsize=16, fontweight='bold')
    plt.title('Descriptive Statistics of Residuals', **myfont, fontsize=18)
    plt.xticks(**myfont, fontsize=14)
    plt.yticks(**myfont, fontsize=14)
    plt.plot(dates, resid_descr[:, 0])
    plt.plot(dates, resid_descr[:, 1])
    plt.plot(dates, resid_descr[:, 2])
    plt.plot(dates, resid_descr[:, 3])
    plt.legend(['Mean', 'Variance', 'Skew', 'Kurtosis'], fontsize=14)
    plt.show()

    return mse_ratio, dm_stat, k_hat, p_hat, m_hat, n_hat, factors, loadings



file = 'united_states.xlsx'
data = read_data(file)
```

```
mse_ratio, dm_stat, k_hat, p_hat, m_hat, n_hat, factors, loadings =
    recursive_forecasts(data, M=50, v=0.5)
```

Listing 3: forecast_US.py

```
# -*- coding: utf-8 -*-
"""
Created on Tue May 14 12:35:10 2019

@author: plagl
"""

import numpy as np
import pandas as pd
from scipy.stats import kurtosis, skew, jarque_bera
from data import read_data
from math import sqrt
from statsmodels.graphics.tsaplots import plot_acf
from model import ar, boost, estimate_factors, DM_test, check_residuals
from sklearn.metrics import mean_squared_error
import matplotlib.pyplot as plt
from sklearn.model_selection import ParameterGrid
myfont = {'fontname':'Serif'}


def recursive_forecasts(data, M=50, v=0.5):
    # Setting up forecasts
    max_factors = 10
    max_p = 4
    s = 63
    h = 1

    # Extracting X and y variables
    array = data.values
    X = array[:, :]
    T, N = X.shape
    y = array[:, -1].reshape(T,1) # GDP is last variable


    P = T-s-h-max_p+1
    y_hat_cbst = np.zeros((P,1))
    y_hat_AR = np.zeros((P,1))
    y_true = y[1+s+h+max_p-2:].reshape(P,1)

    k_hat = np.zeros((P,1))
    p_hat = np.zeros((P,1))
    m_hat = np.zeros((P,2))
    n_hat = np.zeros((P,1))
    resid_descr = np.zeros((P,5))
    sel_Fs = [0] * P
    beta_Fs = [0] * P

    for i in range(0, P):
        print(i)
        y_in = y[h-1 : i+s+h+max_p-1].copy()
        x_in = X[max_p+h-2 : i+s+h+max_p-1].copy()

        # Fit benchmark AR(p) model
        res, y_target, y_lags, beta_AR, p = ar(y_in, max_p)
        y_lags_current = np.insert(y_in[:-p-1:-1], 0, 1.0) # add constant
        y_hat_AR[i]  = y_lags_current @ beta_AR
        p_hat[i] = p
        z_in = y_target - res.predict(y_lags).reshape(len(y_target),1)

```

```
        # Fit the model of specification type 2
        # 1) pre-select X variables with boosting
        sel_X, beta_X, IC1, m1  = boost(x_in[:-1,:], y_target, v, M) # y_target
    better than z_in for SA
        X_selected = x_in[:, sel_X>0]
        n_hat[i] = np.size(X_selected, 1)
        m_hat[i, 0] = m1

        # 2) Only apply PCA if at least max_factors X variables were selected
        if np.size(X_selected, 1) >= max_factors:
            factors, loadings, k, pc = estimate_factors(X_selected, max_factors)
        elif np.size(X_selected, 1) > 0:
            factors = X_selected
            k = 0
        else:
            factors, loadings, k, pc = estimate_factors(x_in, max_factors)
        k_hat[i] = k

        # 3) Use boosting to estimate factor coefficient beta_F
        sel_F, beta_F, IC2, m2 = boost(factors[:-1,:], z_in, v, M)
        m_hat[i, 1] = m2
        sel_Fs[i] = (sel_F > 0).sum()
        beta_Fs[i] = beta_F

        # Fitting model to obtain residuals
        y_fit_AR = y_lags @ beta_AR
        resid_AR = y_target - y_fit_AR
        y_fit = y_lags @ beta_AR + factors[:-1,:] @ beta_F.reshape(len(beta_F),1)
        resid = y_target - y_fit
        resid_descr[[i], 0], resid_descr[[i], 1], resid_descr[[i], 2],resid_descr
    [[i], 3], resid_descr[[i], 4] = check_residuals(resid)

        # Diffusion index forecast
        y_hat_cbst[i] = y_lags_current @ beta_AR + factors[-1,:] @ beta_F.reshape(
    len(beta_F),1)


    # Compute MSE and MSE ratio
    mse_AR = mean_squared_error(y_true, y_hat_AR)
    mse_cbst = mean_squared_error(y_true, y_hat_cbst)
    mse_ratio = mse_cbst / mse_AR
    error_AR = y_true - y_hat_AR
    error_cbst = y_true - y_hat_cbst
    dm_stat = DM_test(error_AR, error_cbst)
    print(dm_stat)

    # Plotting number of selected variables
    dates = data.index[s+h+max_p-1:]
    plt.style.use(['seaborn-whitegrid', 'seaborn-notebook'])
    plt.xlabel('Year', **myfont, fontsize=16, fontweight='bold')
    plt.title('Number of Selected Variables', **myfont, fontsize=18)
    plt.xticks(**myfont, fontsize=14)
    plt.yticks(**myfont, fontsize=14)
    plt.plot(dates, n_hat)
    plt.yticks(np.arange(min(n_hat)-2, max(n_hat)+2, 5.0))
    plt.savefig('./figures/n_hat.eps', format='eps', dpi=1000)
    plt.show()

    # Plotting number of factors estimated by PCA
    plt.style.use(['seaborn-whitegrid', 'seaborn-notebook'])
    plt.xlabel('Year', **myfont, fontsize=16, fontweight='bold')
    plt.title('Estimated Number of Factors', **myfont, fontsize=18)
    plt.xticks(**myfont, fontsize=14)
```

```
    plt.yticks(**myfont, fontsize=14)
    plt.plot(dates, k_hat)
    plt.show()

    # Plotting number of factors selected by boosting
    plt.style.use(['seaborn-whitegrid', 'seaborn-notebook'])
    plt.xlabel('Year', **myfont, fontsize=16, fontweight='bold')
    plt.title('Number of Selected Factors', **myfont, fontsize=18)
    plt.xticks(**myfont, fontsize=14)
    plt.yticks(**myfont, fontsize=14)
    plt.plot(dates, sel_Fs)
    plt.yticks(np.arange(min(sel_Fs), max(sel_Fs)+1, 1.0))
    plt.show()

    # Plotting number of factors estimated by PCA and boosting
    plt.style.use(['seaborn-whitegrid', 'seaborn-notebook'])
    plt.xlabel('Year', **myfont, fontsize=16, fontweight='bold')
    plt.title('Number of Factors', **myfont, fontsize=18)
    plt.xticks(**myfont, fontsize=14)
    plt.yticks(**myfont, fontsize=14)
    plt.plot(dates, k_hat)
    plt.plot(dates, sel_Fs, 'r')
    plt.legend(['PCA', 'Boosting'], fontsize=14)
    plt.savefig('./figures/k_hat.eps', format='eps', dpi=1000)
    plt.show()

    # Plotting AR forecasts vs true values
    plt.style.use(['seaborn-whitegrid', 'seaborn-notebook'])
    plt.xlabel('Year', **myfont, fontsize=16, fontweight='bold')
    plt.title('Quarterly GDP Growth in Percent', **myfont, fontsize=18)
    plt.xticks(**myfont, fontsize=14)
    plt.yticks(**myfont, fontsize=14)
    plt.plot(dates, y_true)
    plt.plot(dates, y_hat_AR, 'r')
    plt.legend(['True Values', 'Forecasts'], fontsize=14)
    plt.savefig('./figures/sa_forecasts_ar_opt.eps', format='eps', dpi=1000)
    plt.show()
    print(mse_AR)

    # Plotting diffusion index forecasts vs true values
    plt.style.use(['seaborn-whitegrid', 'seaborn-notebook'])
    plt.xlabel('Year', **myfont, fontsize=16, fontweight='bold')
    plt.title('Quarterly GDP Growth in Percent', **myfont, fontsize=18)
    plt.xticks(**myfont, fontsize=14)
    plt.yticks(**myfont, fontsize=14)
    plt.plot(dates, y_true)
    plt.plot(dates, y_hat_cbst, 'r')
    plt.legend(['True Values', 'Forecasts'], fontsize=14)
    plt.savefig('./figures/sa_forecasts_cbst_opt.eps', format='eps', dpi=1000)
    plt.show()
    print(mse_cbst)

    # Plotting AR residuals
    dates1 = data.index[h+max_p-1 : P+s+h+max_p-2]
    plt.style.use(['seaborn-whitegrid', 'seaborn-notebook'])
    plt.xlabel('Year', **myfont, fontsize=16, fontweight='bold')
    plt.title('Residuals of AR Model', **myfont, fontsize=18)
    plt.xticks(**myfont, fontsize=14)
    plt.yticks(**myfont, fontsize=14)
    plt.plot(dates1, resid_AR)
    plt.savefig('./figures/sa_resids_ar.eps', format='eps', dpi=1000)
    plt.show()

```

```
    # Plotting SP2 residuals
    plt.style.use(['seaborn-whitegrid', 'seaborn-notebook'])
    plt.xlabel('Year', **myfont, fontsize=16, fontweight='bold')
    plt.title('Residuals of SP2 Model', **myfont, fontsize=18)
    plt.xticks(**myfont, fontsize=14)
    plt.yticks(**myfont, fontsize=14)
    plt.plot(dates1, resid)
    plt.savefig('./figures/sa_resids_cbst.eps', format='eps', dpi=1000)
    plt.show()

    # Plotting descriptives of residuals
    plt.style.use(['seaborn-whitegrid', 'seaborn-notebook'])
    plt.xlabel('Year', **myfont, fontsize=16, fontweight='bold')
    plt.title('Descriptive Statistics of SP2 Residuals', **myfont, fontsize=18)
    plt.xticks(**myfont, fontsize=14)
    plt.yticks(**myfont, fontsize=14)
    #plt.plot(dates, resid_descr[:, 0])
    #plt.plot(dates, resid_descr[:, 1])
    plt.plot(dates, resid_descr[:, 2])
    plt.plot(dates, resid_descr[:, 3], 'r')
    plt.legend(['Skewness', 'Kurtosis'], fontsize=14)
    plt.savefig('./figures/sa_resids_descr.eps', format='eps', dpi=1000)
    plt.show()

    return mse_ratio, dm_stat, k_hat, p_hat, m_hat, n_hat, X_selected, factors,
    loadings, beta_Fs, sel_Fs



file = 'south_africa_quarterly.xlsx'
data = read_data(file)
mse_ratio, dm_stat, k_hat, p_hat, m_hat, n_hat, X_selected, factors, loadings,
    beta_Fs, sel_Fs = recursive_forecasts(data, M=70, v=0.6)
```

Listing 4: forecast_SA.py

```
# -*- coding: utf-8 -*-
"""
Created on Wed May 29 14:03:07 2019

@author: plagl
"""

import numpy as np
from data import read_data
import pandas as pd
from model import ar, boost, estimate_factors, DM_test, check_residuals
from sklearn.metrics import mean_squared_error
from sklearn.model_selection import ParameterGrid



def recursive_forecasts(array, M=50, v=0.5):
    # Setting up forecasts
    max_factors = 10
    max_p = 4
    s = 63
    h = 1

    # Extracting X and y variables
    array = data.values
    X = array[:, :]
    T, N = X.shape
    y = array[:, -1].reshape(T,1) # GDP is last variable
```

```
29
30      P = T-s-h-max_p+1
31      y_hat_cbst = np.zeros((P,1))
32      y_hat_AR = np.zeros((P,1))
33      y_true = y[1+s+h+max_p-2:].reshape(P,1)
34
35      for i in range(0, P):
36          #print(i)
37          y_in = y[h-1 : i+s+h+max_p-1].copy()
38          x_in = X[max_p+h-2 : i+max_p+h+s-1].copy()
39
40          # Fit benchmark AR(p) model
41          res, y_target, y_lags, beta_AR, p = ar(y_in, max_p)
42          y_lags_current = np.insert(y_in[:-p-1:-1], 0, 1.0) # add constant
43          y_hat_AR[i]  = y_lags_current @ beta_AR
44          z_in = y_target - res.predict(y_lags).reshape(len(y_target),1)
45
46          # Fit the model of specification type 2
47          # 1) pre-select X variables with boosting
48          sel_X, beta_X, IC1, m1  = boost(x_in[:-1,:], y_target, v, M) # y_target
    better than z_in for SA
49          X_selected = x_in[:, sel_X>0]
50
51          # 2) Only apply PCA if at least max_factors X variables were selected
52          if np.size(X_selected, 1) >= max_factors:
53              factors, loadings, k, pc = estimate_factors(X_selected, max_factors)
54          elif np.size(X_selected, 1) > 0:
55              factors = X_selected
56              k = 0
57          else:
58              factors, loadings, k, pc = estimate_factors(x_in, max_factors)
59
60          # 3) Use boosting to estimate factor coefficient beta_F
61          sel_F, beta_F, IC2, m2 = boost(factors[:-1,:], z_in, v, M)
62
63          # Diffusion index forecast
64          y_hat_cbst[i] = y_lags_current @ beta_AR + factors[-1,:] @ beta_F.reshape(
    len(beta_F),1)
65
66
67      # Compute MSE and MSE ratio
68      mse_AR = mean_squared_error(y_true, y_hat_AR)
69      mse_cbst = mean_squared_error(y_true, y_hat_cbst)
70      mse_ratio = mse_cbst / mse_AR
71      error_AR = y_true - y_hat_AR
72      error_cbst = y_true - y_hat_cbst
73      dm_stat = DM_test(error_AR, error_cbst)
74
75      print('MSE ratio:', mse_ratio)
76      print('DM test:', dm_stat)
77
78      return error_AR, error_cbst, mse_ratio, dm_stat
79
80
81
82  # Loading data
83  file = 'south_africa_quarterly.xlsx'
84  data = read_data(file)
85  array = data.values
86
87  # Create grid of parameters to be evaluated
88  param_grid = {'v': [0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1.0], \
89                'M': [30, 40, 50, 60, 70]}
```

```
grid = ParameterGrid(param_grid)
C = len(list(grid))      # Number of parameter combinations in grid

# Create arrays to store results for each parameter combination
errors_AR = np.zeros((25,1,C))
errors_cbst = np.zeros((25,1,C))
mse_ratios = np.zeros((C,1))
dm_stats = np.zeros((C,1))

c = 0
for params in grid:
    print(c, params)
    errors_AR[:,:,c], errors_cbst[:,:,c], mse_ratios[c], dm_stats[c] =
    recursive_forecasts(array, params['M'], params['v'])
    c += 1

# Put MSE ratios and DM statistics into dataframes
results_MSE = pd.DataFrame(index=param_grid['M'], columns=param_grid['v'])
results_DM = pd.DataFrame(index=param_grid['M'], columns=param_grid['v'])

c = 0
for m in param_grid['M']:
    for v in param_grid['v']:
        results_MSE.loc[m, v] = mse_ratios[c][0]
        results_DM.loc[m, v] = dm_stats[c][0]
        c += 1

# Save results to csv files
results_MSE.to_csv('./data/results/grid_search_SA_MSE_y.csv')
results_DM.to_csv('./data/results/grid_search_SA_DM_y.csv')


significant = abs(dm_stats) > 1.65
n_significant = sum(significant)
print(n_significant)
```

Listing 5: parameters_SA.py

```
# -*- coding: utf-8 -*-
"""
Created on Mon Jun 10 17:43:42 2019

@author: plagl
"""

import numpy as np
import pandas as pd
from model import estimate_factors
from sklearn.model_selection import ParameterGrid
from statsmodels.tsa.arima_process import ArmaProcess



def generate_data(T, N, k, alpha, beta):
    # factor loadings for generating X
    loadings = np.random.normal(0, 1, (N, k))

    F = np.zeros((T+1,k))
    X = np.zeros((T+1,N))
    y = np.zeros((T+1,1))

    for i in range(0, k):
        ar1 = np.array([1, -alpha[i]])
        ma1 = np.array([1])
```

```
        AR_object = ArmaProcess(ar1, ma1)
        F[:,i] = AR_object.generate_sample(nsample=T+1, scale=1)
        # scale is standard deviation of noise, by default noise is N(0,1)

    for n in range(0, N):
        X[:, n] = F @ loadings[n, :].T + np.sqrt(k) * np.random.normal(0, 1, T+1)

    y = np.roll(F, 1, axis=0)  @ beta.T + np.random.normal(0, 1, T+1)

    return F[1: , :], X[1: , :], y[1:] # cutting off first observation due to lag
    =1



def simulate(replications, k, alpha, beta):
    # Create grid of DGP dimensions
    dim_grid = {'T': [10, 20, 50, 100, 200, 500, 1000], \
                'N': [10, 20, 50, 100, 200]}
    grid = ParameterGrid(dim_grid)
    C = len(list(grid)) # Total number of dimension combinations in grid

    max_factors=10

    # Create arrays to store results for each dimension combination
    hits = np.zeros((replications,1,C))
    num_factors = np.zeros((replications,1,C))

    # Put hit rates, estimated number of factors, MSE ratios and DM statistics
    into dataframes
    hit_rates = pd.DataFrame(index=dim_grid['T'], columns=dim_grid['N'])
    avg_num_factors = pd.DataFrame(index=dim_grid['T'], columns=dim_grid['N'])

    c = 0 # Combination index
    # Iterate through all combinations of dimensions (T,N)
    for dims in grid:
        print(c, dims)

        # Simulate a given amount of times
        for r in range(0, replications):
            F, X, y = generate_data(dims['T'], dims['N'], k, alpha, beta)
            factors, loadings, k_est, PC = estimate_factors(X, max_factors)
            hits[r,:,c] = (k_est == k)
            num_factors[r,:,c] = k_est

        # Compute average of performance measures
        hit_rates.loc[dims['T'], dims['N']] = hits[:,:,c].mean()
        avg_num_factors.loc[dims['T'], dims['N']] = num_factors[:,:,c].mean()
        c += 1 # Update combination index

    # Save average results to csv files
    hit_rates.to_csv('./data/simulations/1000x_hit_rates_' + str(k) + '_fixed.csv'
    )
    avg_num_factors.to_csv('./data/simulations/1000x_avg_num_factors_' + str(k) +
    '_fixed.csv')
    return [hits, num_factors], [hit_rates, avg_num_factors]



# Start simulation with PCA only
alpha = np.array([0.8, 0.7, 0.6, 0.5, 0.4]) # factor AR(1) coefficients
beta = np.array([0.8, 0.5, 0.3, 0, -0.3]) # factor coefficients for generating y
repl=1000

```

```
# fix random seed for reproducibility
np.random.seed(0)
results1, averages1 = simulate(repl, 1, alpha[:1], beta[:1])
print('1----------')

# fix random seed for reproducibility
np.random.seed(0)
results3, averages3 = simulate(repl, 3, alpha[:3], beta[:3])
print('2----------')

# fix random seed for reproducibility
np.random.seed(0)
results5, averages5 = simulate(repl, 5, alpha[:5], beta[:5])

print('3----------')
print('----------')
print('DONE!')
```

Listing 6: simulation_pca.py

```
# -*- coding: utf-8 -*-
"""
Created on Tue Jun 11 15:04:51 2019

@author: plagl
"""


import numpy as np
import pandas as pd
from model import ar, boost, estimate_factors, DM_test, check_residuals
from sklearn.metrics import mean_squared_error
from sklearn.model_selection import ParameterGrid
from statsmodels.tsa.arima_process import ArmaProcess



def recursive_forecasts(X, y, M=50, v=0.5):
    # Setting up forecasts
    max_factors=10
    max_p = 4
    h = 1

    # Extracting X and y variables
    T, N = X.shape
    y = y.reshape(T,1) # GDP is last variable

    s = T - max_p - 25
    P = T-s-h-max_p+1
    y_hat_cbst = np.zeros((P,1))
    y_hat_AR = np.zeros((P,1))
    y_true = y[s+h+max_p-1:].reshape(P,1)

    k_hat = np.zeros((P,1))
    p_hat = np.zeros((P,1))
    m_hat = np.zeros((P,2))
    resid_descr = np.zeros((P,5))

    for i in range(0, P):
        y_in = y[h-1 : i+s+h+max_p-1].copy()
        x_in = X[max_p+h-2 : i+s+h+max_p-1].copy()

        # Fit benchmark AR(p) model
        res, y_target, y_lags, beta_AR, p = ar(y_in, max_p)
```

```
        y_lags_current = np.insert(y_in[:-p-1:-1], 0, 1.0) # add constant
        y_hat_AR[i]  = y_lags_current @ beta_AR
        p_hat[i] = p
        z_in = y_target - res.predict(y_lags).reshape(len(y_target),1)

        # Fit the model of specification type 2
        # 1) pre-select X variables with boosting
        sel_X, beta_X, IC1, m1  = boost(x_in[:-1,:], y_target, v, M) # y_target
    better than z_in for SA
        X_selected = x_in[:, sel_X>0]
        m_hat[i, 0] = m1

        # 2) Only apply PCA if at least max_factors X variables were selected
        if np.size(X_selected, 1) >= max_factors:
            factors, loadings, k, pc = estimate_factors(X_selected, max_factors)
        elif np.size(X_selected, 1) > 0:
            factors = X_selected
            k = 0
        else:
            factors, loadings, k, pc = estimate_factors(x_in, max_factors)
        k_hat[i] = k

        # 3) Use boosting to estimate factor coefficient beta_F
        sel_F, beta_F, IC2, m2 = boost(factors[:-1,:], z_in, v, M)
        m_hat[i, 1] = m2

        # Fitting model to obtain residuals
        y_fit = y_lags @ beta_AR + factors[:-1,:] @ beta_F.reshape(len(beta_F),1)
        resid = y_target - y_fit
        resid_descr[[i], 0], resid_descr[[i], 1], resid_descr[[i], 2],resid_descr
    [[i], 3], resid_descr[[i], 4] = check_residuals(resid)

        # Diffusion index forecast
        y_hat_cbst[i] = y_lags_current @ beta_AR + factors[-1,:] @ beta_F.reshape(
    len(beta_F),1)

    # Compute MSE and MSE ratio
    mse_AR = mean_squared_error(y_true, y_hat_AR)
    mse_cbst = mean_squared_error(y_true, y_hat_cbst)
    mse_ratio = mse_cbst / mse_AR
    error_AR = y_true - y_hat_AR
    error_cbst = y_true - y_hat_cbst
    dm_stat = DM_test(error_AR, error_cbst)
    p_hat = p_hat.reshape((25))
    return mse_ratio, dm_stat, p_hat



def generate_data(T, N, k, alpha, beta):
    # factor loadings for generating X
    loadings = np.random.normal(0, 1, (N, k))

    F = np.zeros((T+1,k))
    X = np.zeros((T+1,N))
    y = np.zeros((T+1,1))

    for i in range(0, k):
        ar1 = np.array([1, -alpha[i]])
        ma1 = np.array([1])
        AR_object = ArmaProcess(ar1, ma1)
        F[:,i] = AR_object.generate_sample(nsample=T+1, scale=1)
        # scale is standard deviation of noise, by default noise is N(0,1)

```

```
    for n in range(0, N):
        X[:, n] = F @ loadings[n, :].T + np.sqrt(k) * np.random.normal(0, 1, T+1)

    y = np.roll(F, 1, axis=0)  @ beta.T + np.random.normal(0, 1, T+1)

    return F[1: , :], X[1: , :], y[1:] # cutting off first observation due to lag=1



def simulate(replications, k, alpha, beta):
    # Create grid of DGP dimensions
    dim_grid = {'T': [50, 100, 200, 500, 1000], \
                'N': [10, 20, 50, 100, 200]}
    grid = ParameterGrid(dim_grid)
    C = len(list(grid)) # Total number of dimension combinations in grid

    # Create arrays to store results for each dimension combination
    hits = np.zeros((replications,1,C))
    num_factors = np.zeros((replications,1,C))
    mse_ratios = np.zeros((replications,1,C))
    dm_stats = np.zeros((replications,1,C))
    ar_lags = np.zeros((replications,25,C))

    # Put hit rates, estimated number of factors, MSE ratios and DM statistics into dataframes
    hit_rates = pd.DataFrame(index=dim_grid['T'], columns=dim_grid['N'])
    avg_num_factors = pd.DataFrame(index=dim_grid['T'], columns=dim_grid['N'])
    avg_mse_ratio = pd.DataFrame(index=dim_grid['T'], columns=dim_grid['N'])

    avg_dm_stat = pd.DataFrame(index=dim_grid['T'], columns=dim_grid['N'])
    dm_sig1 = pd.DataFrame(index=dim_grid['T'], columns=dim_grid['N'])
    dm_sig5 = pd.DataFrame(index=dim_grid['T'], columns=dim_grid['N'])
    dm_sig10 = pd.DataFrame(index=dim_grid['T'], columns=dim_grid['N'])

    max_factors = 10
    c = 0 # Combination index

    # Iterate through all combinations of dimensions (T,N)
    for dims in grid:
        print(c, dims)

        # Simulate a given amount of times
        for r in range(0, replications):
            F, X, y = generate_data(dims['T'], dims['N'], k, alpha, beta)
            factors, loadings, k_est, PC = estimate_factors(X, max_factors)
            hits[r,:,c] = (k_est == k)
            num_factors[r,:,c] = k_est
            mse_ratios[r,:,c], dm_stats[r,:,c], ar_lags[r,:,c] = recursive_forecasts(X, y, M=50, v=0.5)

        # Compute average of performance measures
        hit_rates.loc[dims['T'], dims['N']] = hits[:,:,c].mean()
        avg_num_factors.loc[dims['T'], dims['N']] = num_factors[:,:,c].mean()
        avg_mse_ratio.loc[dims['T'], dims['N']] = mse_ratios[:,:,c].mean()

        avg_dm_stat.loc[dims['T'], dims['N']] = dm_stats[:,:,c].mean()
        dm_sig1.loc[dims['T'], dims['N']] = (dm_stats[:,:,c] > 2.326).sum()
        dm_sig5.loc[dims['T'], dims['N']] = (dm_stats[:,:,c] > 1.645).sum()
        dm_sig10.loc[dims['T'], dims['N']] = (dm_stats[:,:,c] > 1.282).sum()
        c += 1 # Update combination index

    # Save average results to csv files
```

```
    hit_rates.to_csv('./data/simulations/final_hit_rates_' + str(k) + '_fixed.csv')
    avg_num_factors.to_csv('./data/simulations/final_avg_num_factors_' + str(k) + '_fixed.csv')
    avg_mse_ratio.to_csv('./data/simulations/final_avg_mse_ratio_' + str(k) + '_fixed.csv')

    avg_dm_stat.to_csv('./data/simulations/final_avg_dm_stat_' + str(k) + '_fixed.csv')
    dm_sig1.to_csv('./data/simulations/final_dm_sig1_' + str(k) + '_fixed.csv')
    dm_sig5.to_csv('./data/simulations/final_dm_sig5_' + str(k) + '_fixed.csv')
    dm_sig10.to_csv('./data/simulations/final_dm_sig10_' + str(k) + '_fixed.csv')
    return [hits, num_factors, ar_lags, mse_ratios, dm_stats], [hit_rates, avg_num_factors, avg_mse_ratio, avg_dm_stat], [dm_sig1, dm_sig5, dm_sig10]



# Start simulation with SP2
alpha = np.array([0.8, 0.7, 0.6, 0.5, 0.4]) # factor AR(1) coefficients
beta = np.array([0.8, 0.5, 0.3, 0, -0.3]) # factor coefficients for generating y
repl=100

# fix random seed for reproducibility
np.random.seed(0)
results1, averages1, sig_results1 = simulate(repl, 1, alpha[:1], beta[:1])
print('1----------')

# fix random seed for reproducibility
np.random.seed(0)
results3, averages3, sig_results3 = simulate(repl, 3, alpha[:3], beta[:3])
print('2----------')

# fix random seed for reproducibility
np.random.seed(0)
results5, averages5, sig_results5 = simulate(repl, 5, alpha[:5], beta[:5])

print('3----------')
print('----------')
print('DONE!')
```

Listing 7: simulation_sp2.py

```
# -*- coding: utf-8 -*-
"""
Created on Tue Jun 18 11:59:55 2019

@author: plagl
"""


import numpy as np
import pandas as pd
from data import read_data
from model import ar, boost, estimate_factors, DM_test, check_residuals
from statsmodels.tsa.tsatools import lagmat
import matplotlib.pyplot as plt
from sklearn.model_selection import ParameterGrid
from keras.models import Sequential
from keras.layers import Dense
from keras.layers import LSTM
from sklearn.metrics import mean_squared_error


def plot_loss(history):
```

```
    # summarize history for loss
    plt.plot(history.history['loss'])
    plt.plot(history.history['val_loss'])
    plt.title('Model MSE')
    plt.ylabel('MSE',fontsize=16)
    plt.xlabel('Epoch',fontsize=16)
    plt.legend(['train', 'validation'], loc='upper left')
    plt.show()
    return


def split_series(data, lags):
    y_lags, y_target = lagmat(data, lags, trim='forward', original='sep')
    y_lags = y_lags[lags:,:]
    y_target = y_target[lags:]
    y_lags = np.flip(y_lags, axis=1)
    return y_lags, y_target


def univariate_LSTM(data, lags, n_neurons, n_epochs): # Similar to an AR model
    # Prepare training data
    n_variables = 1
    y_lags, y = split_series(data, lags)
    y_lags = y_lags.reshape((y_lags.shape[0], y_lags.shape[1], n_variables))

    # Set up the network architecture
    model = Sequential()
    model.add(LSTM(n_neurons, activation='tanh', input_shape=(lags, n_variables)))

    # Output layer
    model.add(Dense(1))
    model.compile(optimizer='adam', loss='mean_squared_error')

    # Train the model
    model.fit(y_lags, y, epochs=n_epochs, verbose=0, shuffle=False)

    # Use trained LSTM to predict next value
    x_test = y[-lags:]
    x_test = x_test.reshape((1, lags, 1)) # (1, n_steps, n_features)
    y_hat = model.predict(x_test, verbose=0)
    return y_hat


def recursive_forecasts(data, n_neurons, n_epochs):
    # Setting up forecasts
    s = 63
    max_p = 4
    h = 1

    # Extract X and y variables from array
    array = data.values
    X = array[:, :-1]
    T, N = X.shape
    y = array[:, -1].reshape(T,1) # GDP

    # Setting up parameters
    P = T-s-h-max_p+1

    y_hat_AR = np.zeros((P,1))
    y_hat_LSTM = np.zeros((P,1))
    y_true = y[s+h+max_p-1:].reshape(P,1)

    for i in range(0, P):
```

```
        print('Forecast', i)
        y_in = y[h-1 : i+s+h+max_p-1].copy()
        x_in = X[h-1 : i+s+h+max_p-1].copy()

        # Fit benchmark AR(p) model
        res, y_target, y_lags, beta_AR, p = ar(y_in, max_p)
        y_lags_current = np.insert(y_in[:-p-1:-1], 0, 1.0) # add constant
        y_hat_AR[i]  = y_lags_current @ beta_AR
        print('-')

        # Train LSTM networks
        y_hat_LSTM[i] = univariate_LSTM(y_in.copy(), max_p, n_neurons, n_epochs)
        print('-')


    # Compute MSE and MSE ratio
    mse_AR = mean_squared_error(y_true, y_hat_AR)
    mse_LSTM = mean_squared_error(y_true, y_hat_LSTM)
    mse_ratio = mse_LSTM / mse_AR
    print('MSE ratio LSTM/AR:', mse_ratio)

    error_AR = y_true - y_hat_AR
    error_LSTM = y_true - y_hat_LSTM
    dm_stat = DM_test(error_AR, error_LSTM)
    print('DM stat:', dm_stat)
    error_LSTM = error_LSTM.reshape((25))
    return error_LSTM, [mse_AR, mse_LSTM], mse_ratio, dm_stat



# fix random seed for reproducibility
np.random.seed(0)

file = 'south_africa_quarterly.xlsx'
data = read_data(file)

# Create grid of parameters to be evaluated
param_grid = {'n': [10, 20, 50, 100, 200], \
              'e': [50, 100, 200]}
grid = ParameterGrid(param_grid)
C = len(list(grid))     # Number of parameter combinations in grid

replications = 10

# Create arrays to store results
mse = np.zeros((replications,2,C))
errors = np.zeros((replications,25,C))
mse_ratios = np.zeros((replications,1,C))
dm_stats = np.zeros((replications,1,C))

mse_ratios1 = pd.DataFrame(index=param_grid['e'], columns=param_grid['n'])
dm_stats1 = pd.DataFrame(index=param_grid['e'], columns=param_grid['n'])
avg_dm_stat = pd.DataFrame(index=param_grid['e'], columns=param_grid['n'])


c = 0 # Combination index
# Iterate through all combinations of dimensions (T,N)
for params in grid:
    print(c, params)
    print("--------------------")

    for r in range(0, replications):
        print(r)
```

```
149         errors[r,:,c], mses_list, mse_ratios[r,:,c], dm_stats[r,:,c] =
    recursive_forecasts(data, params['n'], params['e'])
150         mse[r,0,c], mse[r,1,c] = mses_list[0], mses_list[1]
151
152     mse_ratios1.loc[params['e'], params['n']] = mse_ratios[:,:,c].mean()
153     dm_stats1.loc[params['e'], params['n']] = (dm_stats[:,:,c] > 1.645).sum()
154     avg_dm_stat.loc[params['e'], params['n']] = dm_stats[:,:,c].mean()
155     c += 1 # Update combination index
156
157 # Save average results to csv files
158 mse_ratios1.to_csv('./data/networks/mse_ratios_univar.csv')
159 dm_stats1.to_csv('./data/networks/dm_stats_univar.csv')
160 avg_dm_stat.to_csv('./data/networks/avg_dm_stat_univar.csv')
```

Listing 8: network_SA.py