Forecasting the Indian Index of Industrial Production
A Machine Learning Approach

Priyanshu Kumar, Sayam Chakraborty, Kanishkar J, Saptaswa Mukherjee

Department of Humanities & Social Sciences, Indian Institute of Space Science & Technology

HS222 Introduction to Economics

Dr. Shaijumon C S

April 14, 2024

Index

Forecasting the Indian Index of Industrial Production
A Machine Learning Approach

Introduction

This project aims to apply machine learning techniques for forecasting the index of industrial production (IIP), an important macroeconomic indicator for monitoring the industrial output of an economy.

The IIP is a quantum index, the production of items being expressed in physical terms. It details the growth of economic sectors involved in industrial production. A volume index, the IIP is compiled as a simple weighted arithmetic mean of production relatives based on the Laspeyres formula (MoSPI, Govt. of India; Yamada, 2016):

$$I = \frac{\sum W_i R_i}{\sum W_i}$$

where R_i is the production relative and W_i is the weight of an item at the ISIC/NIC Division (or) 2-digit level. A similar approach is applied for the use-based product classification as well.

The all-India IIP is a composite indicator that measures the short-term changes in the volume of production of a basket of industrial products in India during a given period with respect to that in a chosen base period. It is compiled and published monthly by the Central Statistics Office (CSO) six weeks after the reference month ends (OGD Platform India, Govt. of India). The scope of the IIP as recommended by the United Nations Statistics Division (UNSD) – the agency responsible for the International Standard Industrial Classification of All Economic Activities (ISIC) – includes mining, manufacturing, construction, electricity, gas, and water supply. However, due to constraints of data availability, the IIP compiled in India excludes construction, gas, and water supply sectors (MoSPI, Govt. of India).

The macroeconomic indicators for monitoring the manufacturing sector include the gross domestic product (GDP) and the IIP. Since GDP numbers are available only annually, therefore, the IIP, which appears every month, albeit in arrears by two months, has become the most widely used macroeconomic variable to monitor growth in industrial output. As such, the IIP numbers are highly anticipated by policymakers, stakeholders and third parties alike (Sodhi et al, 2013).

However, it is not as if the IIP is free from doubt & scrutiny. Even though better than an annual rate, IIP numbers are, for all forms of consideration, retrospective, given the six-week time lag. There have also been problems concerning “noise” and a high degree of volatility in the index (Sodhi et al, 2013). Additionally, the validity of the methods used by the CSO for compiling the data has been challenged. The growth numbers released monthly by the CSO are not in concordance with the reports of the Annual Survey of Industries, the annual series released by the *Economic Survey of India*. The quality of primary data supplied by the Department for Promotion of Industry & Internal Trade (the erstwhile Department of Industrial Policy & Promotion, DIPP), has also been under scrutiny (Nagaraj, 1999).

Forecasting of macroeconomic variables like the IIP is a challenging yet popular exercise in understanding economic growth, given its policy relevance. As such, we need a way of performing statistical analysis on pre-existing data to reliably predict future values, whilst considering that the data is inherently “noisy” and is subject to error to a non-trivial degree.

Herein we attempt to propose a possible solution to the aforementioned problems: machine learning. By training models on relevant datasets, we can make a computer analyze the data and predict future values based on the patterns & trends observed. The caveat of going for “strict” modeling is that it causes a strong adherence to the supplied data: as the data is noisy, the model, too, becomes subject to the gusts of noise – “overfitting”.

As such, we go for regression with regularization – a “lenient” approach. By imposing less penalty on the model through adjusting the regularization constant, we can identify the trends in the data and thereby produce suitable predictions. However, if the degree of leniency is too high, it may lead to the model ignoring even the desired patterns, thereby rendering it dysfunctional – “underfitting”.

This balancing act between strictness & leniency is key to developing a stable, functional, and robust model. This can be achieved through suitable optimization of hyperparameters – the number of neurons used, the learning rate of the machine, the regularization constant, etc.

Research Statement

This project aims to develop forecasting models for the monthly index of industrial production (IIP) via machine learning models developed using:

1. Support Vector Machine Regression (SVM Regression/SVR), and
2. Recurrent Neural Networks (RNN),

focusing on the sectoral indices at the NIC Division level, and the indices for the use-based classification categories. At the NIC Division level, we consider only the products & activities under Section C – Manufacturing, i.e., Divisions 10-33 as enlisted in the National Industrial Classification 2008 (NIC 2008).

The datasets used for training the models are derived from:

1. The monthly sectoral indices from April 2012 to June 2022 (base year 2011-12 = 100) at the NIC 2-digit level,
2. The monthly use-based indices from April 2005 to March 2017 (base year 2004-05 = 100), and
3. The monthly use-based indices from June 2012 to June 2022 (base year 2011-12 = 100).

This project focuses purely on the development of ML models: integration & deployment of the model with a user interface for feeding in current/real-time data and making predictions is beyond the scope of our work. There is a limit to the time for which predictions can be made far-out from the month for which data has been made available to the models.

We refrain from training the models on the annual IIP figures for two reasons. First, the amount of annual IIP data made available to us was found to be inadequate for our purposes: this may be overcome through more advanced ML techniques currently beyond our scope. Secondly, for a volatile index like the IIP, monthly figures are more suited to capturing market trends than annual figures. There exist traditional statistical methods to deal with such situations – we do not venture into that domain and leave it as an open end for future work in this field.

Methodology

The models have been developed using Python programing in the Google Colaboratory. The relevant data from the spreadsheets was converted into .csv (comma separated values) format, and then uploaded to Colab.

SVM Regression

A Support Vector Machine (SVM) is a powerful and versatile supervised Machine Learning model, capable of performing linear or nonlinear classification, regression, and even outlier detection. It is one of the most popular models in Machine Learning. SVMs are particularly well suited for classification of complex small- or medium-sized datasets (Géron, 2019).

SVM Regression or SVR tries to find a function that best predicts the continuous output value for a given input value. It is designed to predict continuous numeric values, making it suitable for tasks like time series forecasting and stock price prediction (Theodore T B and Ince H, 2000).

For our analysis, we used SVR to construct a hyperplane in three-dimensional space. Subsequently, the distances of points initially residing in two dimensions – x & y – were shifted by a constant along the z-axis. Using the Radial Basis Function (RBF) kernel, we identified the optimal curve fit. The non-linear kernel was optimized by adjusting the γ (gamma) parameter, thereby delineating a curved surface in three dimensions. Finally, all points from this three-dimensional surface were projected back to the regular two-dimensional xy-plane, effectively reducing the dimensionality of the curve to a best-fit non-linear 2D curve.

More information about SVM Regression is available in Appendix – I.

LSTM – Long Short-Term Memory (Recurrent Neural Network – RNN)

An artificial neural network (ANN) is a machine learning model inspired by the neuronal organization found in the biological neural networks in animal brains (Wikipedia).

An ANN is made of connected units or nodes called “artificial neurons”, which loosely model the neurons in a brain. These are connected by edges, which model the synapses in a brain. An artificial neuron receives signals from connected neurons, then processes them and sends a signal to other connected neurons. The "signal" is a real number, and the output of each neuron is computed by some non-linear function of the sum of its inputs, called the activation function.

Neurons and edges typically have a weight that adjusts as learning proceeds. The weight increases or decreases the strength of the signal at a connection.

Typically, neurons are aggregated into layers. Different layers may perform different transformations on their inputs. Signals travel from the first layer (the input layer) to the last layer (the output layer), possibly passing through multiple intermediate layers (hidden layers).

Artificial neural networks find use in predictive modeling, adaptive control, artificial intelligence, and other applications where they can be trained via a dataset.

A recurrent neural network (RNN) is one of the two broad types of artificial neural network, characterized by direction of the flow of information between its layers. In contrast to the unidirectional feedforward neural network, it is a bi-directional artificial neural network, meaning that it allows the output from some nodes to affect subsequent input to the same nodes.

Long short-term memory (LSTM) is a recurrent neural network (RNN) architecture, aimed at dealing with the vanishing gradient problem present in traditional RNNs. Its relative insensitivity to gap length is its advantage over other RNNs, hidden Markov models and other sequence learning methods. It aims to provide a short-term memory for RNN that can last thousands of timesteps, thus "long short-term memory". LSTM finds varied applications in fields demanding classification, processing, and predicting data based on time series (Wikipedia).

A common LSTM unit is composed of a cell, an input gate, an output gate and a forget gate. The cell remembers values over arbitrary time intervals and the three gates regulate the flow of information into and out of the cell. Forget gates decide what information to discard from a previous state by assigning a previous state, compared to a current input, a value between 0 and 1. A rounded value of 1 means to keep the information, and a value of 0 means to discard it. Input gates decide which pieces of new information to store in the current state, using the same system as forget gates. Output gates control which pieces of information in the current state to output by assigning a value from 0 to 1 to the information, considering the previous and current states.

Selectively outputting relevant information from the current state allows the LSTM network to maintain useful, long-term dependencies to make predictions, both in current and future time-steps.

We have employed an LSTM-based RNN to the monthly IIP data – a time series – to help with prediction of future IIP values. More information about LSTM RNNs is available in Appendix – II.

Data Analysis

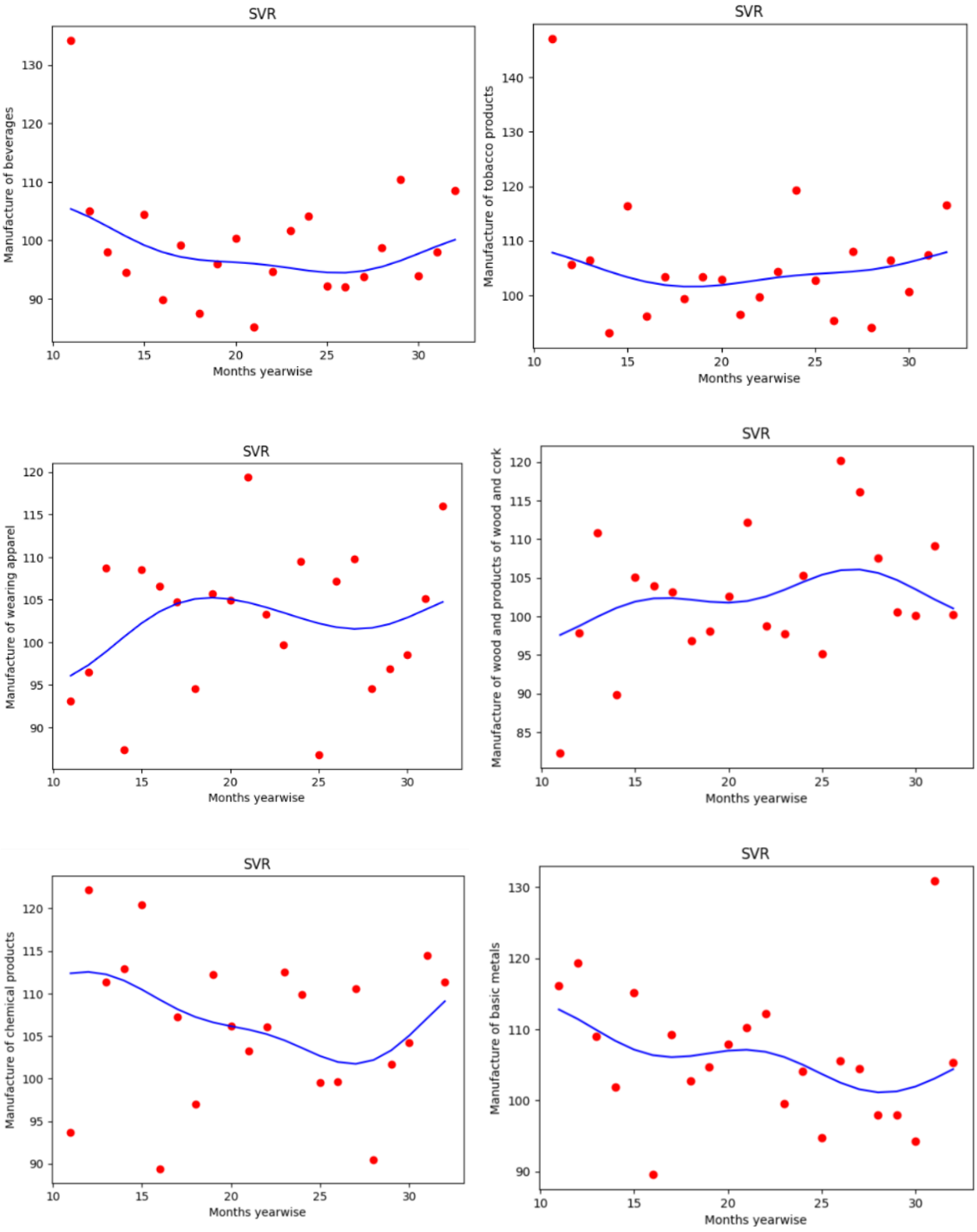
SVM Regression

As explained earlier, SVM Regression involves shifting of the data points from a 2D plane to a fixed height along the z-axis in a 3D space. This is followed by hyperplane fitting and projecting the relevant support vectors back onto a 2D plane to obtain the regression curve and support vector projections.

The Python code for SVM Regression / SVR is as follows:

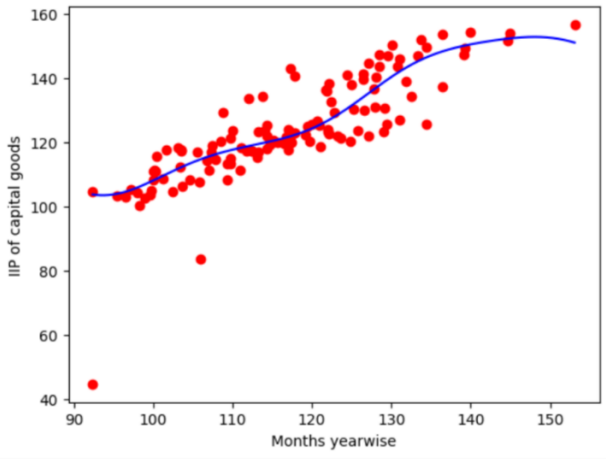
```
[20] X = df.iloc[1:,0].values.reshape(-1,1)
     y = df.iloc[1:,2].values.reshape(-1,1)

     y = y.reshape(len(y),1)

     # Feature Scaling
     from sklearn.preprocessing import StandardScaler
     sc_X = StandardScaler()
     sc_y = StandardScaler()
     X = sc_X.fit_transform(X)
     y = sc_y.fit_transform(y)


     # Training the SVR model on the whole dataset
     from sklearn.svm import SVR
     regressor = SVR(kernel = 'rbf')
     regressor.fit(X, y)

     # Visualising the SVR results (for higher resolution and smoother curve)
     X_grid = np.arange(min(sc_X.inverse_transform(X)), max(sc_X.inverse_transform(X)), 0.1)
     X_grid = X_grid.reshape((len(X_grid), 1))
     plt.scatter(sc_X.inverse_transform(X), sc_y.inverse_transform(y), color = 'red')
     plt.plot(X_grid, sc_y.inverse_transform(regressor.predict(sc_X.transform(X_grid)).reshape(-1,1)), color = 'blue')
     plt.xlabel('Months yearwise')
     plt.ylabel('WPI of basic goods')
     plt.show()



     plt.scatter(sc_X.inverse_transform(X), sc_y.inverse_transform(y), color = 'red')
     plt.plot(sc_X.inverse_transform(X), sc_y.inverse_transform(regressor.predict(X).reshape(-1,1)), color = 'blue')
     plt.title('SVR')
     plt.xlabel('Months yearwise')
     plt.ylabel('WPI of basic goods')
     plt.show()


     # Predicting a new result
     sc_y.inverse_transform(regressor.predict(sc_X.transform([[145]])).reshape(-1,1))
```

The following are the results for the NIC 2008 Division-level indices in the time-frame April 2005 to March 2017 (base year 2004-05) for select Divisions:



The horizontal axis represents time – each month is assigned a “test set number”. An offset of 10 is provided by default: it does not have any real impact on the model.

Applying SVR to the indices for Capital Goods for the time-frame June 2012 to June 2022 yields the following graph with R^2-score 0.70. Once again, the offset of 90 on the time axis has no real impact on the model.



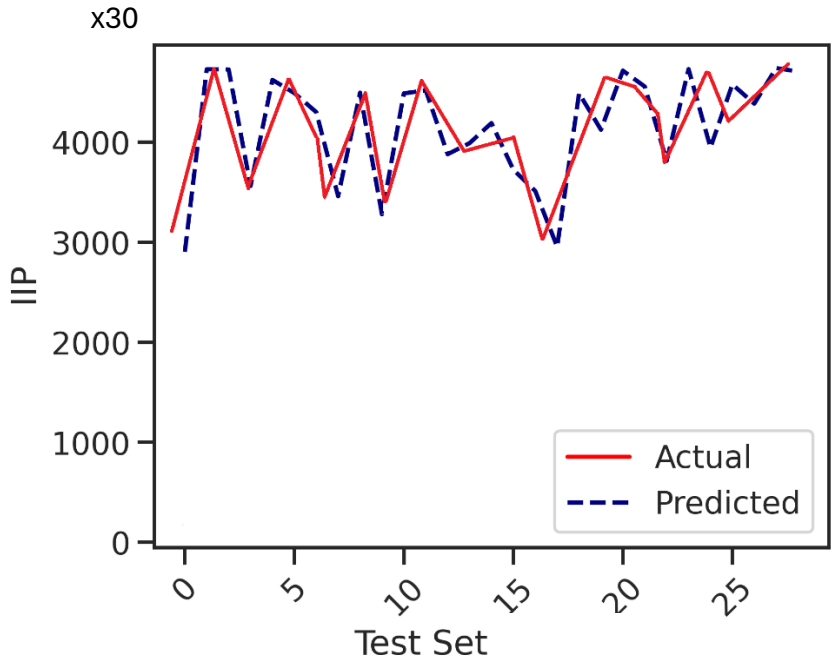
LSTM RNN

The Python code used for the LSTM RNN is as follows:

```
[ ]  # LSTM
     import numpy as np
     import matplotlib.pyplot as plt
     import tensorflow as tf
     from pandas import read_csv
     from keras.models import Sequential
     from keras.layers import Dense
     from keras.layers import LSTM
     from sklearn.preprocessing import MinMaxScaler
     from sklearn.metrics import mean_squared_error
     # convert an array of values into a dataset matrix
     def create_dataset(dataset, look_back=1):
      dataX, dataY = [], []
      for i in range(len(dataset)-look_back-1):
         a = dataset[i:(i+look_back), 0]
      dataX.append(a)
      dataY.append(dataset[i + look_back, 0])
      return np.array(dataX), np.array(dataY)
     # fix random seed for reproducibility
     tf.random.set_seed(7)
     # load the dataset
     dataframe = read_csv('Book3.csv', usecols=[1], engine='python')
     dataset = dataframe.values
     dataset = dataset.astype('float32')
     # normalize the dataset
     scaler = MinMaxScaler(feature_range=(0, 1))
     dataset = scaler.fit_transform(dataset)
     # split into train and test sets
     train_size = int(len(dataset) * 0.67)
     test_size = len(dataset) - train_size
     train, test = dataset[0:train_size,:], dataset[train_size:len(dataset),:]
     # reshape into X=t and Y=t+1
     look_back = 3
     trainX, trainY = create_dataset(train, look_back)
     testX, testY = create_dataset(test, look_back)
     # reshape input to be [samples, time steps, features]
     trainX = np.reshape(trainX, (trainX.shape[0], 1, trainX.shape[1]))
     testX = np.reshape(testX, (testX.shape[0], 1, testX.shape[1]))
```

```
# create and fit the LSTM network
model = Sequential()
model.add(LSTM(4, input_shape=(1, look_back)))
model.add(Dense(1))
model.compile(loss='mean_squared_error', optimizer='adam')
model.fit(trainX, trainY, epochs=100, batch_size=1, verbose=2)
# make predictions
trainPredict = model.predict(trainX)
testPredict = model.predict(testX)
# invert predictions
trainPredict = scaler.inverse_transform(trainPredict)
trainY = scaler.inverse_transform([trainY])
testPredict = scaler.inverse_transform(testPredict)
testY = scaler.inverse_transform([testY])
# calculate root mean squared error
trainScore = np.sqrt(mean_squared_error(trainY[0], trainPredict[:,0]))
print('Train Score: %.2f RMSE' % (trainScore))
testScore = np.sqrt(mean_squared_error(testY[0], testPredict[:,0]))
print('Test Score: %.2f RMSE' % (testScore))
# shift train predictions for plotting
trainPredictPlot = np.empty_like(dataset)
trainPredictPlot[:, :] = np.nan
trainPredictPlot[look_back:len(trainPredict)+look_back, :] = trainPredict
# shift test predictions for plotting
testPredictPlot = np.empty_like(dataset)
testPredictPlot[:, :] = np.nan
testPredictPlot[len(trainPredict)+(look_back*2)+1:len(dataset)-1, :] = testPredict
# plot baseline and predictions
plt.plot(scaler.inverse_transform(dataset))
plt.plot(trainPredictPlot)
plt.plot(testPredictPlot)
plt.show()
```

The following graph shows the time series prediction based on the IIP figures for Primary/Basic Goods for the period June 2012 – June 2022. Each month has been assigned a “test set number”, starting from zero. The plots for the actual data and the values predicted by the LSTM RNN are as follows.



As is evident from the graph, the LSTM predictions closely align with the actual values: the R^2-score is 0.85. The vertical axis (IIP) has been scaled by a factor of 30 for better visibility due to closeness of values.

Discussion

The analysis shows that the two models can predict IIP numbers in excess of six months from the month last fed. However, we recommend setting the six-month mark as the line post which the models will have to be retrained with real data. This comes from an approach taken by other researchers, albeit with traditional statistical analysis instead of ML techniques (Sodhi et al, 2013).

The R^2-score for SVR applied to Capital Goods (2012 – 2022) stands good at 0.80. For the sectoral/Division-level indices, while the R^2-score is not substantial, the regression curves achieved are fair, given the high degree of variation of the IIP.

The LSTM RNN predicts IIP values for Primary/Basic Goods (2012 – 2022) to a high degree of agreement.

Limitations

The techniques used herein can provide meaningful predictions only for "normal" years where there are no major deviations from the regular business cycles. The 2008 crisis caused a downturn in economic activity – models constructed based on the data till 2007 failed to predict the same and are hence omitted even though the results may not be too far off the mark. Additionally, this research cannot predict extraordinary situations based on pre-existing data, e.g. industrial production hit a severe low during the COVID-19 pandemic, to the extent that India's real GDP shrunk from 2019 to 2020.

Policy Recommendations

One of the observations made was that the complexity of the models increased significantly with increasing data. While the models can account for general variations, inconsistencies in the data collated by the CSO became apparent at various stages. This is in agreement with the debate between the CSO and R Nagaraj (1999). It is, therefore, imperative that the primary sources of data be thoroughly investigated. The Statistical Divisions within the various government departments need to be strengthened.

Secondly, the Indian IIP excludes figures of construction, gas, and water supply – factors of significant importance, especially in the context of developing countries. The UN Statistics Division, too, suggests the inclusion of these factors in IIP calculations. However, the CSO cites

problems in data availability to not include them in the all-India IIP. This reiterates the case presented in the previous recommendation.

Furthermore, the UN Statistics Division suggests revision of the IIP base year at five-year intervals. The problems due to the infrequent revision in India became apparent at a stage when the models were being trained on the use-based category-wise IIP. The data for the period 2012-2017 is available with two base years: 2004-05 and 2011-12. The decision to make the switch from 2004-05 to 2011-12 was made only in 2017 – eight years after the previous revision in 2009. This made it difficult to model the use-based IIP figures. The unavailability of a proper linking factor further underscores the issue. While there are extra-economic & political ramifications of updating the base year, a compromise can be made to the extent of making the IIP base-year linking factor readily available – at the cost of model simplicity.

Conclusion

Applying machine learning to forecast volatile indices like the IIP is a challenging but interesting exercise. This project is a small attempt towards the same. While the results may not be in complete agreement with actual data, the degree of success achieved with relatively simple methods is a testament to the power of ML techniques. Any open ends are left for exploring ML in finance & economy and possible future work in this domain.

References

Ministry of Statistics and Programme Implementation, Government of India. “Index of Industrial Production”. IIP - MoSPI

Nuffield Foundation. “Measuring Inflation using Laspeyres Index”. Laspeyres Index - Nuffield Foundation

Open Government Data (OGD) Platform India. “Index of Industrial Production”. IIP - OGD

Yamada, T (2016). “Methodological Recommendations for the Compilation of the Index Numbers of Industrial Production (IIP)”, UN-ESCWA Training Workshop on Short-term Economic Indicators. Archive - UN-ESCWA

Sodhi, M S, J Sharma, S Singh and A Walia (2013). "A Robust and Forward-looking Industrial Production Indicator", *Economic & Political Weekly, 48*(48), pp 126–130. JSTOR

Nagaraj, R (1999). "How Good Are India's Industrial Statistics? An Exploratory Note", *Economic & Political Weekly*, *36*(6), pp 350–355. JSTOR

Singhi, M C (2009). "Index of Industrial Production & Annual Survey of Industries", Working Paper, Department of Industrial Policy & Promotion, Ministry of Commerce & Trade, Govt. of India. Archive - Indian Statistics

Géron, A (2019). "Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow – Concepts, Tools, and Techniques to Build Intelligent Systems" – 2nd Edition, *O'Reilly*. Google Books

Wikipedia. "Long short-term Memory". LSTM - Wikipedia

Brownlee, J (2022). "Time Series Prediction with LSTM Recurrent Neural Networks in Python with Keras", *Deep Learning for Time Series,* Machine Learning Mastery. Time Series Prediction - LSTM RNN

Theodore, T B and Ince H (2000). "Support vector machine for regression and applications to financial forecasting", Proceedings of the IEEE-INNS-ENNS International Joint Conference on Neural Networks. IJCNN 2000. Neural Computing: New Challenges and Perspectives for the New Millennium, IEEE. IEEE Xplore

Appendix – I: Support Vector Machine – SVM Regression

Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow – Concepts, Tools, and Techniques to Build Intelligent System – 2nd Edition

Book by Aurélien Géron, *O'Reilly*

[Relevant Excerpts]

Chapter 5: Support Vector Machines

Non-Linear SVM Classification

Gaussian RBF Kernel (pp 160 – 161)

Just like the polynomial features method, the similarity features method can be useful with any Machine Learning algorithm, but it may be computationally expensive to compute all the additional features, especially on large training sets. Once again the kernel trick does its SVM magic, making it possible to obtain a similar result as if you had added many similarity features. Let's try the SVC class with the Gaussian RBF kernel:

```
rbf_kernel_svm_clf = Pipeline([
        ("scaler", StandardScaler()),
        ("svm_clf", SVC(kernel="rbf", gamma=5, C=0.001))
    ])
rbf_kernel_svm_clf.fit(X, y)
```

This model is represented at the bottom left in Figure 5-9. The other plots show models trained with different values of hyperparameters gamma (γ) and C. Increasing gamma makes the bell-shaped curve narrower (see the lefthand plots in Figure 5-8). As a result, each instance's range of influence is smaller: the decision boundary ends up being more irregular, wiggling around individual instances. Conversely, a small gamma value makes the bell-shaped curve wider: instances have a larger range of influence, and the decision boundary ends up smoother. So γ acts like a regularization hyperparameter: if your model is overfitting, you should reduce it; if it is underfitting, you should increase it (similar to the C hyperparameter).

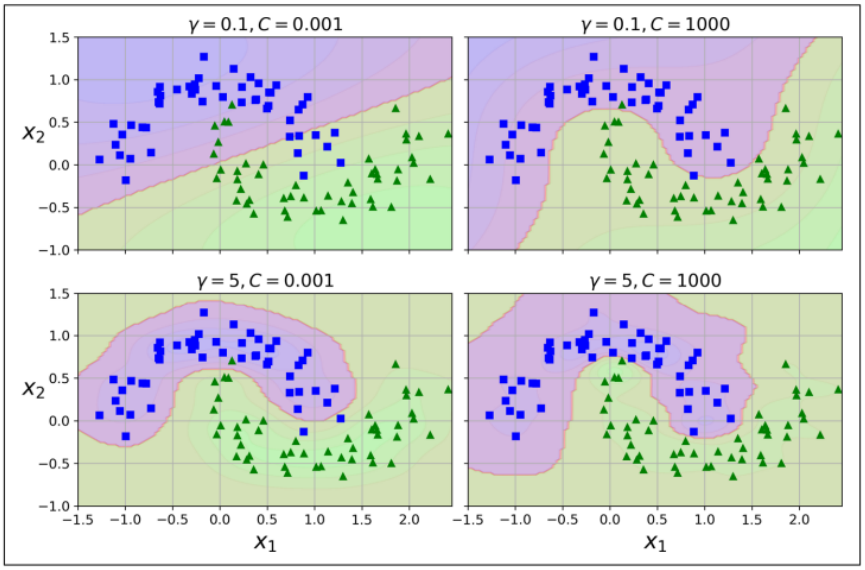


Figure 5-9. SVM classifiers using an RBF kernel

Other kernels exist but are used much more rarely. Some kernels are specialized for specific data structures. String kernels are sometimes used when classifying text documents or DNA sequences (e.g., using the string subsequence kernel or kernels based on the Levenshtein distance)

SVM Regression (pp 162 – 164)

As mentioned earlier, the SVM algorithm is versatile: not only does it support linear and nonlinear classification, but it also supports linear and nonlinear regression. To use SVMs for regression instead of classification, the trick is to reverse the objective: instead of trying to fit the largest possible street between two classes while limiting margin violations, SVM Regression tries to fit as many instances as possible on the street while limiting margin violations (i.e., instances off the street). The width of the street is controlled by a hyperparameter, ϵ. Figure 5-10 shows two linear SVM Regression models trained on some random linear data, one with a large margin (ϵ = 1.5) and the other with a small margin (ϵ = 0.5).

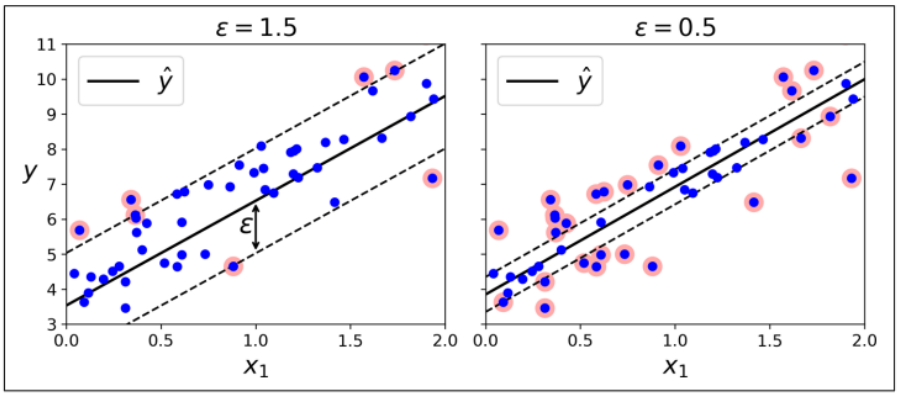


Figure 5-10. SVM Regression

Adding more training instances within the margin does not affect the model's predictions; thus, the model is said to be ϵ-insensitive. You can use Scikit-Learn's LinearSVR class to perform linear SVM Regression. The following code produces the model represented on the left in Figure 5-10 (the training data should be scaled and centered first):

```
from sklearn.svm import LinearSVR

svm_reg = LinearSVR(epsilon=1.5)
svm_reg.fit(X, y)
```

To tackle nonlinear regression tasks, you can use a kernelized SVM model. Figure 5-11 shows SVM Regression on a random quadratic training set, using a second-degree polynomial kernel. There is little regularization in the left plot (i.e., a large C value), and much more regularization in the right plot (i.e., a small C value).

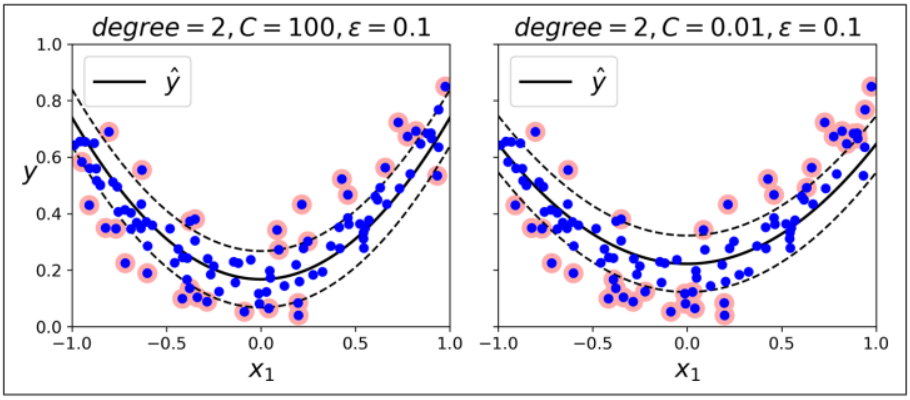


Figure 5-11. SVM Regression using a second-degree polynomial kernel

The following code uses Scikit-Learn's SVR class (which supports the kernel trick) to produce the model represented on the left in Figure 5-11:

```
from sklearn.svm import SVR

svm_poly_reg = SVR(kernel="poly", degree=2, C=100, epsilon=0.1)
svm_poly_reg.fit(X, y)
```

The SVR class is the regression equivalent of the SVC class, and the LinearSVR class is the regression equivalent of the LinearSVC class. The LinearSVR class scales linearly with the size of the training set (just like the LinearSVC class), while the SVR class gets much too slow when the training set grows large (just like the SVC class).

Appendix – II: Long Short-Term Memory – LSTM

Deep Learning

Book by Ian Goodfellow, Yoshua Bengio, Aaron Courville, *MIT Press*

[Relevant Excerpts]

Chapter 10: Sequence Modeling: Recurrent and Recursive Nets

10.10 The Long Short-Term Memory and Other Gated RNNs

As of this writing, the most effective sequence models used in practical applications are called *gated RNNs*. These include the *long short-term memory* and networks based on the *gated recurrent unit*. Like leaky units, gated RNNs are based on the idea of creating paths through time that have derivatives that neither vanish nor explode. Leaky units did this with connection weights that were either manually chosen constants or were parameters. Gated RNNs generalize this to connection weights that may change at each time step. Leaky units allow the network to **accumulate** information (such as evidence for a particular feature or category) over a long duration. However, once that information has been used, it might be useful for the neural network to **forget** the old state. For example, if a sequence is made of sub-sequences and we want a leaky unit to accumulate evidence inside each sub-subsequence, we need a mechanism to forget the old state by setting it to zero. Instead of manually deciding when to clear the state, we want the neural network to learn to decide when to do it. This is what gated RNNs do.

10.10.1 LSTM

The clever idea of introducing self-loops to produce paths where the gradient can flow for long durations is a core contribution of the initial long short-term memory (LSTM) model (Hochreiter and Schmidhuber, 1997). A crucial addition has been to make the weight on this self-loop conditioned on the context, rather than fixed (Gers et al., 2000). By making the weight of this self-loop gated (controlled by another hidden unit), the time scale of integration can be changed dynamically. In this case, we mean that even for an LSTM with fixed parameters, the time scale of integration can change based on the input sequence, because the time constants are output by the model itself. The LSTM has been found extremely successful in many applications, such as unconstrained handwriting recognition (Graves et al., 2009), speech recognition (Graves et al., 2013; Graves and Jaitly, 2014), handwriting generation (Graves, 2013), machine translation (Sutskever et al., 2014), image captioning (Kiros et al., 2014b; Vinyals et al., 2014b; Xu et al., 2015) and parsing (Vinyals et al., 2014a). The LSTM block diagram is illustrated in Fig. 10.16. The corresponding forward propagation equations are given below, in the case of a shallow recurrent network architecture.

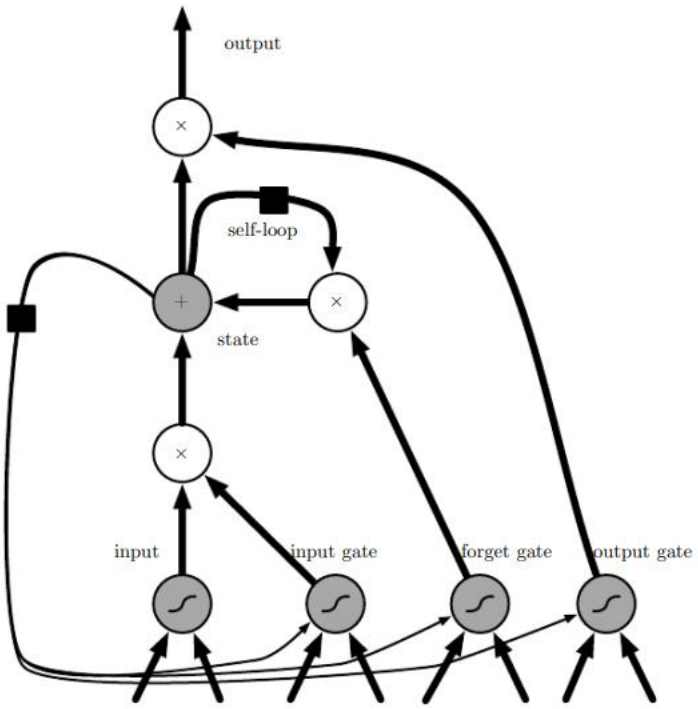


Figure 10.16: Block diagram of the LSTM recurrent network "cell." Cells are connected recurrently to each other, replacing the usual hidden units of ordinary recurrent networks. An input feature is computed with a regular artificial neuron unit. Its value can be accumulated into the state if the sigmoidal input gate allows it. The state unit has a linear self-loop whose weight is controlled by the forget gate. The output of the cell can be shut off by the output gate. All the gating units have a sigmoid nonlinearity, while the input unit can have any squashing nonlinearity. The state unit can also be used as an extra input to the gating units. The black square indicates a delay of a single time step.

Deeper architectures have also been successfully used (Graves et al., 2013; Pascanu et al., 2014a). Instead of a unit that simply applies an elementwise nonlinearity to the affine transformation of inputs and recurrent units, LSTM recurrent networks have "LSTM cells" that have an internal recurrence (a self-loop), in addition to the outer recurrence of the RNN. Each cell has the same inputs and outputs as an ordinary recurrent network, but has more parameters and a system of gating units that controls the flow of information. The most important component is the state unit s (t) i that has a linear self-loop similar to the leaky units described in the previous section. However, here, the self-loop weight (or the associated time constant) is controlled by a forget gate unit $f_i^{(t)}$ (for time step t and cell i), that sets this weight to a value between 0 and 1 via a sigmoid unit:

$$f_i^{(t)} = \sigma\left(b_i^f + \sum_j U_{i,j}^f x_j^{(t)} + \sum_j W_{i,j}^f h_j^{(t-1)}\right),$$

where $\boldsymbol{x}^{(t)}$ is the current input vector and $\boldsymbol{h}^{(t)}$ is the current hidden layer vector, containing the outputs of all the LSTM cells, and $\boldsymbol{b}^f$, $\boldsymbol{U}^f$, $\boldsymbol{W}^f$ are respectively biases, input weights and recurrent weights for the forget gates. The LSTM cell internal state is thus updated as follows, but with a conditional self-loop weight $f_i^{(t)}$:

$$s_i^{(t)} = f_i^{(t)} s_i^{(t-1)} + g_i^{(t)} \sigma\left(b_i + \sum_j U_{i,j} x_j^{(t)} + \sum_j W_{i,j} h_j^{(t-1)}\right),$$

where ***b***, ***U*** and ***W*** respectively denote the biases, input weights and recurrent weights into the LSTM cell. The external input gate unit $g_i^{(t)}$ is computed similarly to the forget gate (with a sigmoid unit to obtain a gating value between 0 and 1), but with its own parameters:

$$g_i^{(t)} = \sigma\left(b_i^g + \sum_j U_{i,j}^g x_j^{(t)} + \sum_j W_{i,j}^g h_j^{(t-1)}\right).$$

The output $h_i^{(t)}$ of the LSTM cell can also be shut off, via the output gate $q_i^{(t)}$, which also uses a sigmoid unit for gating:

$$h_i^{(t)} = \tanh\left(s_i^{(t)}\right) q_i^{(t)}$$
$$q_i^{(t)} = \sigma\left(b_i^o + \sum_j U_{i,j}^o x_j^{(t)} + \sum_j W_{i,j}^o h_j^{(t-1)}\right)$$

which has parameters $\boldsymbol{b}^o$, $\boldsymbol{U}^o$, $\boldsymbol{W}^o$ for its biases, input weights and recurrent weights, respectively. Among the variants, one can choose to use the cell state $s_i^{(t)}$ as an extra input (with its weight) into the three gates of the *i*-th unit, as shown in Fig. 10.16. This would require three additional parameters.

LSTM networks have been shown to learn long-term dependencies more easily than the simple recurrent architectures, first on artificial data sets designed for testing the ability to learn long-term dependencies (Bengio et al., 1994; Hochreiter and Schmidhuber, 1997; Hochreiter et al., 2001), then on challenging sequence processing tasks where state-of-the-art performance was obtained (Graves, 2012; Graves et al., 2013; Sutskever et al., 2014). Variants and alternatives to the LSTM have been studied and used and are discussed next.

Appendix – III: Classification of Industrial Products & Activities

Use-based Classification of Industrial Products

Primary goods

Capital goods

Intermediate goods

Infrastructure/Construction goods

Consumer durables

Consumer non-durable

NIC 2008 Section-level Classification

A – Agriculture, forestry and fishing

B – Mining and quarrying

C – Manufacturing

D – Electricity, gas, steam and air conditioning supply

E – Water supply; sewerage, waste management and remediation activities

F – Construction

G – Wholesale and retail trade; repair of motor vehicles and motorcycles

H – Transportation and storage

I – Accommodation and Food service activities

J – Information and communication

K – Financial and insurance activities

L – Real estate activities

M – Professional, scientific and technical activities

N – Administrative and support service activities

O – Public administration and defence; compulsory social security

P – Education

Q – Human health and social work activities

R – Arts, entertainment and recreation

S – Other service activities

T – Activities of households as employers; undifferentiated goods & services producing activities of households for own use

U – Activities of extraterritorial organizations and bodies

NIC 2008 Division-level Classification under Section C – Manufacturing

10 – Manufacture of food products

11 – Manufacture of beverages

12 – Manufacture of tobacco products

13 – Manufacture of textiles

14 – Manufacture of wearing apparel

15 – Manufacture of leather and related products

16 – Manufacture of wood and products of wood and cork, except furniture; manufacture of articles of straw and plaiting materials

17 – Manufacture of paper and paper products

18 – Printing and reproduction of recorded media

19 – Manufacture of coke and refined petroleum products

20 – Manufacture of chemicals and chemical products

21 – Manufacture of pharmaceuticals, medicinal chemical and botanical products

22 – Manufacture of rubber and plastics products

23 – Manufacture of other non-metallic mineral products

24 – Manufacture of basic metals

25 – Manufacture of fabricated metal products, except machinery and equipment

26 – Manufacture of computer, electronic and optical products

27 – Manufacture of electrical equipment

28 – Manufacture of machinery and equipment n.e.c.

29 – Manufacture of motor vehicles, trailers and semi-trailers

30 – Manufacture of other transport equipment

31 – Manufacture of furniture

32 – Other manufacturing

33 – Repair and installation of machinery and equipment