STRICTLY PRIVATE AND CONFIDENTIAL

Price returns Prediction using LSTM RNNs

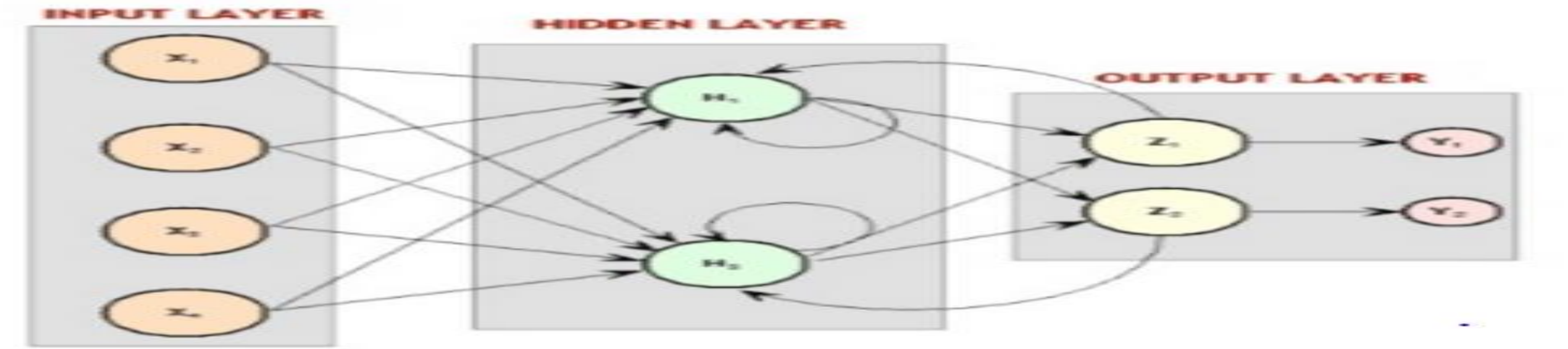
Sujit Khanna

Email: sujit.khanna88@gmail.com

Ph:+91-9820237171

Recurrent Neural Networks

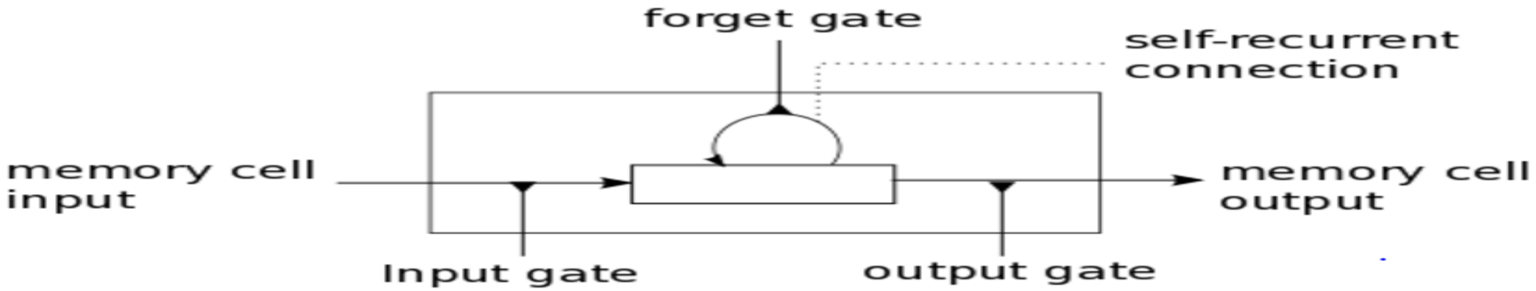
- Class of ANNs with connections between units to form direct cycles
- RNNs make use of sequential information and allows information to persist
- RNNs can be thought of as multiple copies of the same network, each passing information to a successor



- This makes RNNs are useful for tasks related to text and speech recognition
- However, the classical RNN architecture faces the vanishing gradient problem
- Vanishing Gradient Problem: Gradient decreases exponentially as number of layers to train increases with front layer learning slowly
- This makes the model only remember recent events and forgets the more distant past
- By solving the vanishing gradient problem we can model long-term dependency present in data

Long Short Term Memory Units

- LSTMs are capable of solving the vanishing gradient problem and learning long-term dependencies
- LSTM RNN is a form of deep learning algorithm that contains LSTM blocks instead of regular units
- LSTM block contains gates that determine when the input is significant enough to remember, when it should forget and when it should output the value



- Input gate: A sigmoid layer that decides which values need to be updated
- Forget gate: A sigmoid layer that decides whether the information needs to be thrown or saved
- Output gate: A sigmoid layer that decides what parts of cell state will be passed to output, the cell output is generally passed through a hyperbolic function before being passed to output gate
- These gates help the model keep its memory longer when needed and ignore when not required

Data Processing and Feature Creation

- Input features created based on Price returns of Nifty Futures
- The inspiration behind creation of features comes from Taylor's series expansion
- According to Taylor's series expansion:
- $f(x)= f(a) + f'(a)(x-a) + f''(a)(x-a)^2/2! + f'''(a)(x-a)^3/3! + \ldots\ldots\ldots$
- We, therefore, use the derivatives of returns as the input features in our model
- The returns derivatives up to 3rd order are calculated at lags of 5, 10 and 20 minutes
- The input features and corresponding forward returns(i.e. the output) are the normalized between 0 to 1 using MinMax Scalar
- The scaled input features and outputs are then bucketed into 10 classes.
- Output classes are further One Hot Encoded before being passed to the network for training the model
- One Hot Encoding transforms the categorical data to formats that work better with classification algorithms and make predicted outputs less noisy

Model Types and Specification

- For initial analysis we evaluate performance of multiple variants of RNNs, the models included in our scope are
  - A simple RNN (MLP architecture)
  - LSTM RNN
  - Stateful LSTM RNN
  - Bi-Directional LSTM RNN

- Number of Layers: 2
- Units in Layers:
  - Layer 1: 20 Units
  - Layer 2: 20 Units

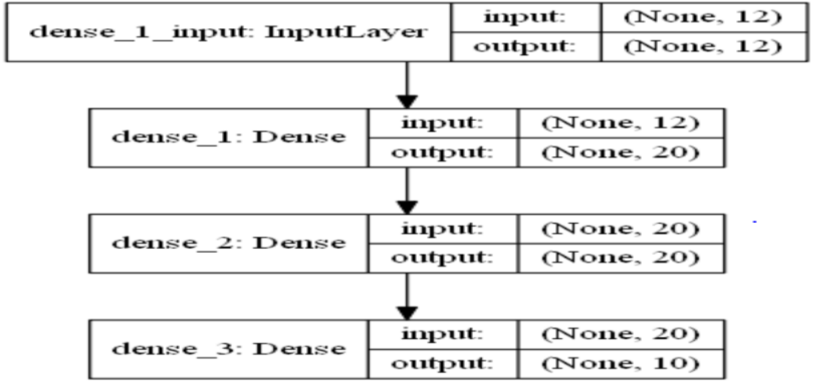
- Model Specifications:
  - LSTM Activation Units: tanh
  - Optimizers: Adam
  - Dense Activation Units: Softmax
  - Loss Function: Categorical Crossentropy
  - Metrics: Accuracy
  - nb_epochs: 20

Stateful and Bi-Directional LSTM RNNs

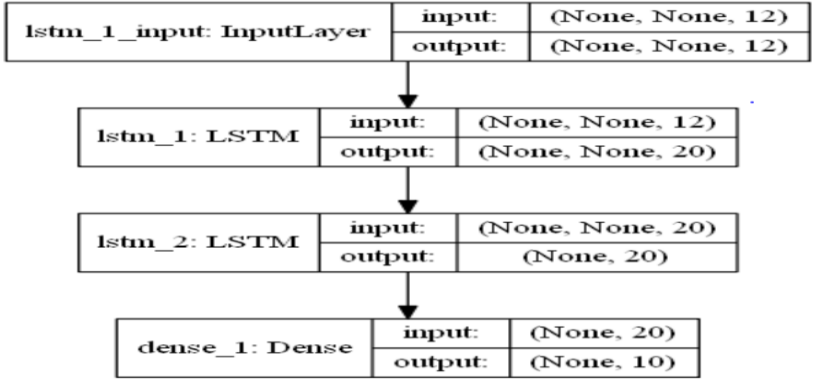
- **<u>Stateful LSTM RNNs:</u>**
  - States computed for the samples in one batch will be reused as initial states for the samples in the next batch
  - The stateless implementation of LSTMs in keras resets the state of the network after each batch
  - Exposes itself to the entire sequence of the batch to learn the inter-dependencies, rather using dependencies explicitly provided to the network.
  - Batch size is explicitly specified as a dimension on the input shape and the same batch is used when making predictions
- **<u>Bi-directional LSTM RNNs:</u>**
  - BRNNs were introduced to increase the amount of information available to the network
  - The principle purpose of BRNN is to split the neurons of regular RNN into 2 directions i.e. positive time direction (forward states) and negative time direction(backward states)
  - They are trained to predict both positive and negative time directions simultaneously
  - The final classification result is generated by combining the score of results produced by both hidden layers

Stateful and Bi-Directional LSTM RNNs

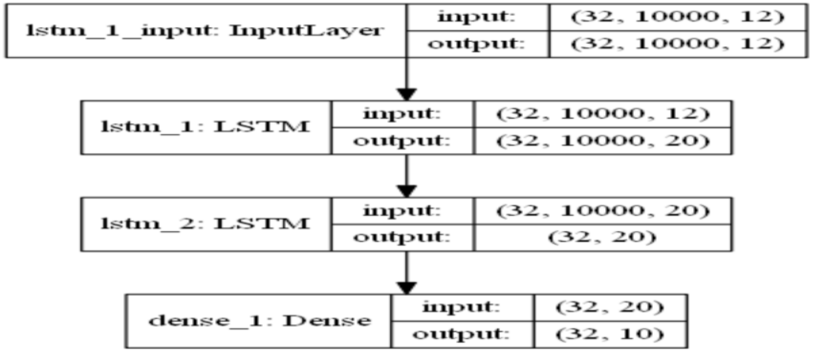
MLP/Simple RNN



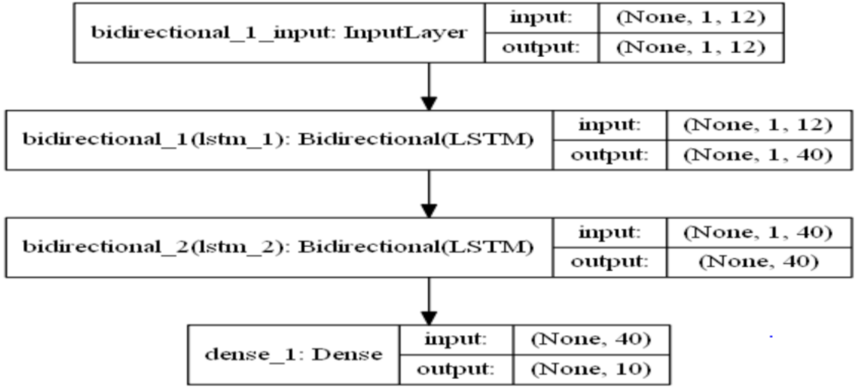
LSTM RNN



Stateful LSTM RNN



Bi-directional LSTM RNN



Model Selection

- Data used in evaluation ranges from 03-Jan-2015 to 31-Dec-2015, with train-test split ratio of 80/20 ratio
- From table below networks using variants of LSTM architecture show considerable outperformance as compared to a regular architecture
- Of all the models LSTM RNN architecture provides the best out of sample model performance
- However Stateful LSTM model also provides an edge when compared to the performance of MLP and Bi-Directional Models

Model Type	OS Loss	OS Accuracy
Regular RNN(MLP)	12.62	16.30%
LSTM RNN	1.813	29.12%
Stateful LSTM RNN	1.823	29.14%
Bi-Directional LSTM RNN	1.843	28.03%

Hyper-Parameter Optimization

- Performed a grid search to find best optimizer and activation unit for the LSTM Network

Optimizer Grid	OS Loss	OS Accuracy
SGD	1.847	27.16%
RMSprop	1.815	28.48%
Adagrad	1.813	29.33%
Adadelta	1.812	28.80%
Adam	1.802	29.12%
Adamax	1.805	28.54%
Nadam	1.813	28.19%

Activation Grid	OS Loss	OS Accuracy
tanh	1.802	29.12%
elu	1.801	29.09%
relu	1.801	29.28%
sigmoid	1.826	28.54%
hard_sigmoid	1.824	28.14%
linear	1.809	28.93%

- The table below shows the Hyper-Parameter grid search optimization results for Learning rate and Decay of the best fit Optimizer

Learning rate	0.001	0.01	0.1	0.2	0.3	0.001	0.01	0.1	0.2	0.3	0.001	0.01	0.1	0.2	0.3	0.001	0.01	0.1	0.2	0.3
Decay	0	0	0	0	0	0.001	0.001	0.001	0.001	0.001	0.01	0.01	0.01	0.01	0.01	0.1	0.1	0.1	0.1	0.1
OS Loss	1.81	1.85	2.20	2.27	13.53	1.82	1.82	2.19	2.19	13.53	1.94	1.83	2.19	2.19	13.53	2.20	1.96	2.19	2.19	13.53
OS Accuracy	31.1%	28.8%	14.7%	14.7%	16.1%	29.3%	29.8%	14.7%	14.7%	16.1%	25.6%	29.6%	14.7%	14.7%	16.1%	18.1%	24.4%	14.7%	14.7%	16.1%

Model Training and Evaluation

- Based on best fit hyper-parameters we train the model on a rolling basis, the data set under consideration ranges from 4-Jan-16 to 25-Jan-17
- We train the model on rolling basis over a period of 125 days and test Out-of-Sample performance over 25 days
- This provides us with total OS prediction of 125 days
- Both LSTM and Stateful LSTM models were evaluated in this study, with model parameters as specified below
    - Units, Layers: 20 ,2
    - Optimizer: Adam
    - Activation Function: Relu
    - Optimizer Parameters: Learn- 0.001, Decay-0.0
    - nb_epochs:100
    - loss: categorical_crossentropy
    - metrics=accuracy
- Metrics used for model evaluation include Confusion Matrix, Precision, Recall and F-Score

LSTM Model Evaluation

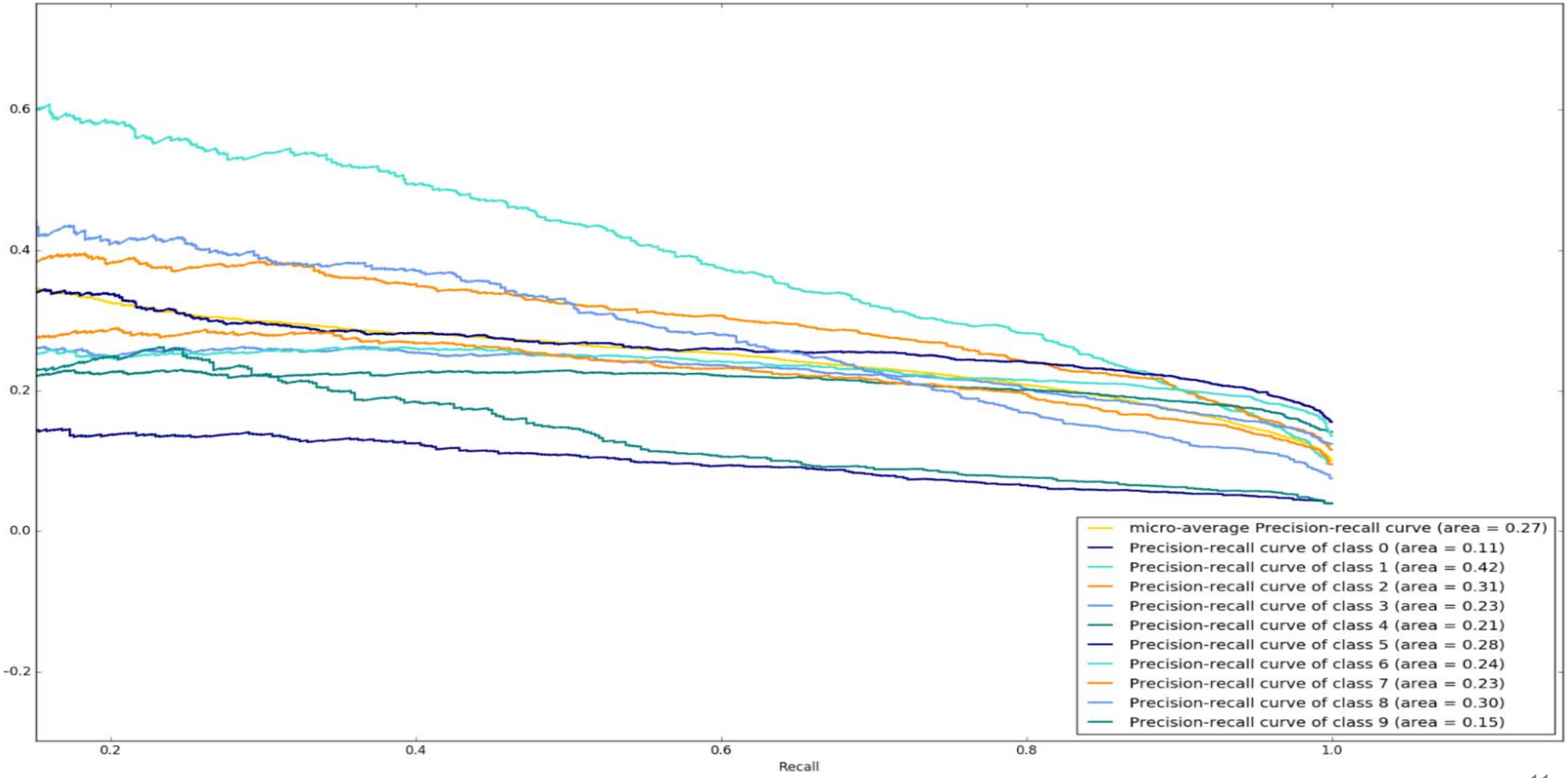
Confusion Matrix										
Class	1	2	3	4	5	6	7	8	9	10
1	12	94	94	71	62	39	11	4	4	1
2	20	323	294	130	79	27	14	3	2	1
3	8	156	451	251	162	62	30	7	5	2
4	5	40	304	352	288	156	72	19	7	1
5	3	12	132	322	400	302	171	49	19	5
6	4	3	40	162	360	492	349	106	23	6
7	3	2	19	77	248	332	423	187	60	6
8	0	3	3	20	100	170	273	243	115	6
9	0	2	4	8	44	84	140	196	205	4
10	1	0	5	16	33	62	72	87	118	6

Class	Average Precision
1	0.11
2	0.42
3	0.31
4	0.23
5	0.21
6	0.28
7	0.24
8	0.23
9	0.30
10	0.15

Metrics	Multiclass	Conditional Long	Conditional Short
Precision	0.288	0.956	0.976
Recall	0.255	0.992	0.998
F-Score	0.256	0.973	0.986

Metrics	Accuracy
MultiClass	29.00%
Conditional Long	91.40%
Conditional Short	95.22%

Precision-Recall Curve LSTM



Stateful LSTM Model Evaluation

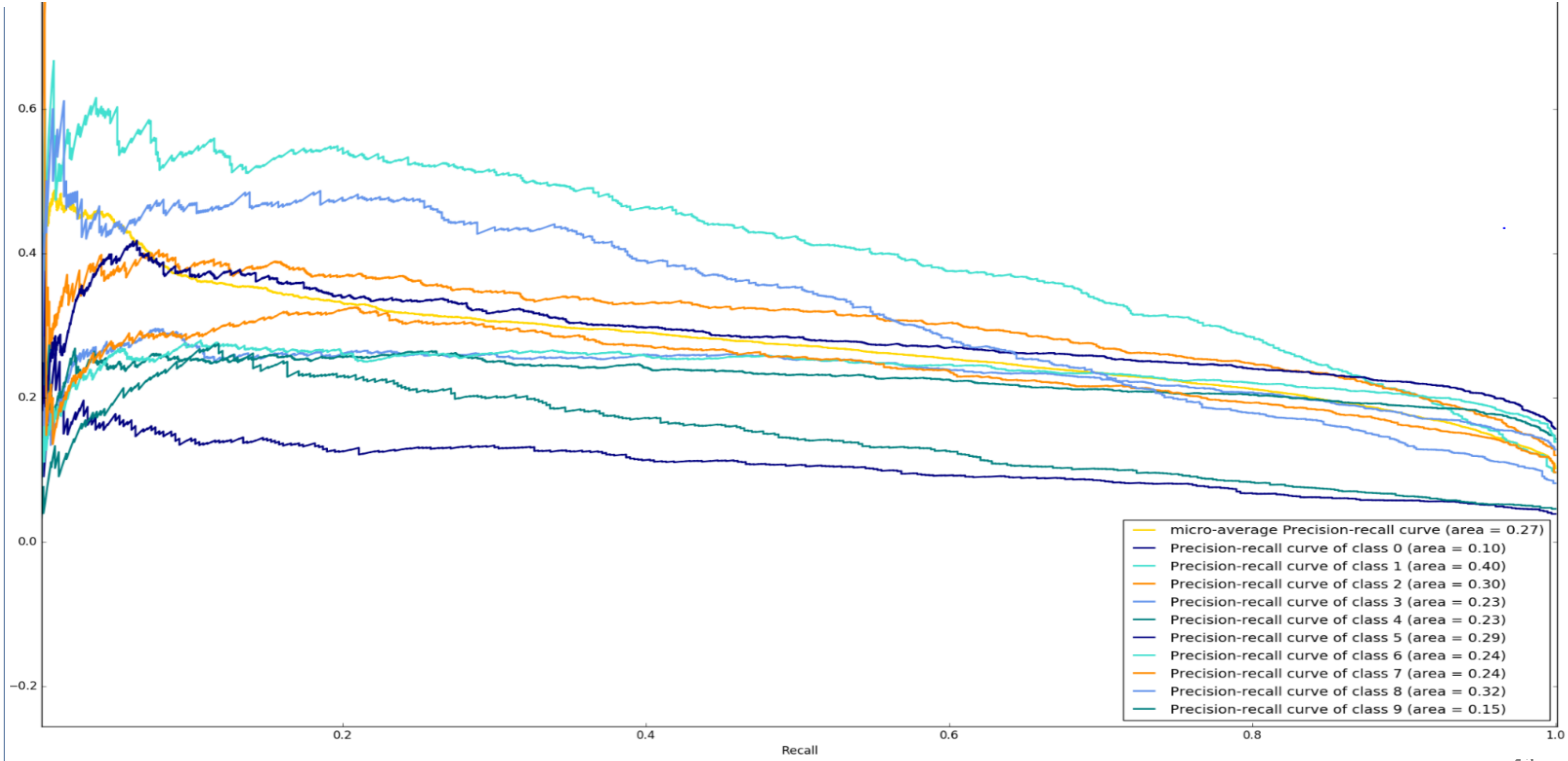
Confusion Matrix										
Class	1	2	3	4	5	6	7	8	9	10
1	10	73	108	61	69	35	20	2	2	0
2	37	257	319	125	66	35	17	3	1	0
3	10	136	423	285	140	76	27	5	4	0
4	7	35	258	387	263	190	84	16	2	1
5	0	8	136	313	345	348	221	33	11	2
6	2	5	34	153	248	560	437	85	17	3
7	2	2	17	69	162	386	492	147	68	9
8	0	1	2	26	58	176	312	228	127	2
9	0	1	5	6	37	90	147	163	234	3
10	0	0	4	15	35	46	91	74	130	3

Class	Average Precision
1	0.1
2	0.4
3	0.3
4	0.23
5	0.22
6	0.29
7	0.24
8	0.24
9	0.32
10	0.15

Metrics	Multiclass	Conditional Long	Conditional Short
Precision	0.285	0.97	0.976
Recall	0.258	0.995	0.998
F-Score	0.257	0.9822	0.986

Metrics	Accuracy
MultiClass	29.62%
Conditional Long	94.03%
Conditional Short	96.41%

Precision-Recall Curve Stateful LSTM



Strategy Model Example

- The output of the LSTM RNN can be monetized by using it as an input for a trading strategy model
- To analyze the performance of the network we created a simple trading model that uses the predicted classes of the model to initiate long/short positions in Nifty Futures
- The strategy logic is defined below:
  - If predicted class(t)<=X and jump Ind(t)<=Y and Large Ind(t)>=Z, go Short
  - If predicted class(t-1)<=X and predicted class(t)<=W and jump Ind(t)<=Y and Large Ind(t)>=Z, go Short
  - If predicted class(t)>=A and jump Ind(t)<=Y and Large Ind(t)>=Z, go Long
  - If predicted class(t-1)>=B and predicted class(t)>=B and jump Ind(t)<=Y and Large Ind(t)>=Z, go Long
- Above rules were applied to the predicted output to generate trading signals, the table below shows the performance of the strategy

Metrics	Long	Short	Total
Trades	12	32	44
Winning Trades	6	16	22
Hit Rate	50%	50%	50%
Trade Returns	1.10%	1.31%	2.41%
Average Return	0.092%	0.041%	0.055%

Conclusion

- Standalone accuracy metrics do not seem very encouraging, however conditional metrics provide very impressive OS performance
- Provides a clear outperformance over Simple RNN or MLP architecture
- The networks used in the study were restricted in terms of number of hidden layers and units used, the OS performance can be enhanced further by using deeper networks
- Extending the feature base to other non price series base features like Order Book and Index Options data can further improve the explanatory power of the network
- The model provides excellent conditional classification results which did not translate to superior strategy performance
- Using a better normalization technique that doesn’t assume a normal distribution for feature creation can improve strategy performance

Future Scope

- Model complexity can be increased by using advanced activations units like LeakyReLU, PReLU, SReLU etc. to improve performance of the network
- Using regularization and dropout techniques to reduce overfitting for deeper networks
- Modelling Stock market environment using broad market features
- Performing feature compression on a higher dimensional representation through autoencoders, PCA, t-sne and using it as input for the trained network

Thank You

Contact Details:
Ph no: +91-9820237171
Email: sujit.khanna88@gmail.com