COMPETITIVE MODEL: VOTING-BASED ENSEMBLE LEARNING

Authors

Tricia Chia | Cyrus Lee | See Hui Yee

NUS FINTECH SOCIETY

nusfintech@gmail.com

Summary

In this report, we will be using 3 classifiers that we have learnt to build our trading model – Decision Tree, Logistic Regression and Random Forest.

Our model is deeply driven by the study *"Evaluating Machine Learning Classification for Financial Trading: An Empirical Approach"* by Gerlein, McGinnity, Belatreche, & Coleman (2016).

Unlike complex machine learning algorithms like Neural Network, these classifiers perform the best in stable environment (Gerlein, Mcginnity, Belatreche, & Coleman, 2016). As such, our trading algorithm focuses on *'EURUSD'* since it is the least volatile, yielding better results.

Our model made use of several technical indicators like Relative Position, Relative Strength Index (RSI), Momentum (MOM) and Moving Average Convergence Divergence (MACD) to generate signals for calls to action.

To prevent overfitting and reduce bias of our model, we employed the ensemble voting method which combines predictions made by the 3 classifiers. Risk management strategies like Stop-Loss & Take-Profit (SLTP) were included as a condition for our model to decide whether to liquidate our position. After which, we discussed about the strengths and weaknesses of our model. Lastly, we included our backtesting result to illustrate how our model makes decisions.

Table of Contents

Our Trading Model

Machine Learning Algorithm

Selection of simpler machine learning algorithms -- Decision Tree, Random Forest, and Logistic Regression was preferred over complex algorithms like Neural Networks as they usually perform much better during normal market behaviour, i.e. when there are no significant economic crises (stable environment) (Ibid.). Moreover, given the lack of computational power, we are unable to properly train neural networks which only tend to perform better when the training data size is very large.

Currency Pair

In our selection of the currency pair to work with, we considered the volatilities of the currency pairs since our model will perform the best in a stable environment unlike complex models like Neural Network which provide better accuracies by learning patterns from volatile currency pairs (Ibid.).

Table of The Most Volatile Currency Pairs

Major pairs	
Currency pair	Volatility (in points per day)
EUR/USD	75.35
GBP/USD	138.6
USD/JPY	130.05
USD/CHF	78.3
AUD/USD	82.9
NZD/USD	89.75
USD/CAD	109.7
Avg.	**100.66**

Cross pairs	
Currency pair	Volatility (in points per day)
AUD/CAD	90.95
AUD/CHF	85.2
AUD/JPY	94.95
AUD/NZD	111.2
CAD/CHF	72.25
CAD/JPY	106
CHF/JPY	129.85
EUR/AUD	147.45
EUR/CAD	119.2
EUR/CHF	64.95
EUR/GBP	85.9
EUR/JPY	132.85
EUR/NZD	176.65
GBP/AUD	225.25
GBP/CAD	205.1
GBP/CHF	152.2
GBP/JPY	217.9
GBP/NZD	266.45
NZD/JPY	99.95
Avg.	**136.01**

Exotic pair	
Currency pair	Volatility (in points per day)
USD/BRL	418.5
USD/DKK	428.6
USD/HKD	31.85
USD/ILS	244.75
USD/INR	241.25
USD/SEK	723.9
USD/SGD	76.1
USD/TRY	377.2
Avg.	**317.77**

The Most Volatile Currency Pairs – Table (data from 26-01-18)

https://fxssi.com/most-volatile-currency-pairs

Out of the more commonly used currency pairs, *'EURUSD'* is the least volatile. As *'EURUSD'* is widely used in the market, the demand and supply for it (i.e. liquidity) is high. Since volatility of the currency pair is related to its liquidity, the price of *'EURUSD'* is harder to change, and hence less volatile. Hence, we chose to work with *'EURUSD'*.

Technical Indicators

As for the technical indicators, we have selected relative position, RSI, momentum and MACD as the features for our models.

Relative Position

Relative position stores the prices of our currency-pairs within the rolling window period which in our model is 55 after min-max scaling such that the prices have values between 0 and 1. Since prices of the currency-pairs might vary largely in terms of magnitude, there can be biases as higher prices will be assigned higher weights. Furthermore, we made use of PCA in our model, which tends to skew towards data with high magnitudes due to their high variances. Hence, scaling is carried out to reduce biases. After scaling the prices, we then store these scaled prices and use them as one of the technical indicators to make predictions.

Relative Strength Index

Relative Strength Index (RSI) is a momentum oscillator that measures the speed and change of price movements. Traditionally, RSI values that exceed 70 indicate overbought or overvalued conditions and implies that price of the currency-pair will go down soon, hence it is best to go short when RSI is nearing 70. Values below 30 indicate oversold or undervalued conditions and implies that price of currency-pair might go up soon, thus making it suitable to go long when RSI is approaching 30. However, one disadvantage of solely using RSI as an indicator is that sudden sharp price changes can cause it to go up and down repeatedly, thus creating inaccurate buy and sell signals. Hence, it is more effective when combined with other technical indicators.

Momentum

The momentum indicator calculation is used to identify the strength of a price movement. It compares the most recent currency-pair price with a previous closing price of a certain period, which in our model, is a rolling window period of 55 days. If the indicator has a value above 100%, the current currency-pair price is greater than the price 55 days ago and might continue rising, generating a buy signal (long position) whereas if value is below 100%, the current price is lower and might continue dropping, thus generating a sell signal (short position). How far the indicator is below or above 100 indicates how fast the price is changing.

Moving Average Convergence Divergence

Moving Average Convergence Divergence is a form of trend-following momentum indicator that shows the relationship between the moving averages of prices by calculating the difference between the slow and fast period moving averages. For our MACD indicator, we made use of the Exponential Moving Average (EMA) as our moving average type. The fast and slow EMAs of our indicator are of 27 and 55 days respectively. The MACD line is the difference between the 27-EMA and 55-EMA. When the line maintains above zero for a period of time, the trend is likely to be increasing and this provides a potential buy signal. However, when they stay below zero for a sustained period of time, the trend is likely to be decreasing and this indicates a potential sell signal.

Process for Model Updates & Symbol Selection

```
    # Principle Component Analysis
    def PCA(self, prices_in_df):

        # Normalize all Values in DataFrame
        normalized_price = (prices_in_df - prices_in_df.mean())/prices_in_df.std()
```

As explained in our model write-up, before doing exploratory data analysis, we normalized our data through z-score standardisation using the PCA() function. This is to ensure that there will not be much variation in the magnitude of the currency-pair prices which allows fairer comparison across currency pairs.

```
if master_table.tail(1).iloc[0][4 + self.sym_n] == 1.0 and \
   master_table.tail(1).iloc[0][5 + self.sym_n] == 1.0 and \
   master_table.tail(1).iloc[0][6 + self.sym_n] == 1.0 and \
   self.trading_symbol not in self.long_list and \
   self.trading_symbol not in self.short_list :

    self.SetHoldings(self.trading_symbol, 1)
    self.long_list.append(self.trading_symbol)
    self.Debug("long")

if self.trading_symbol in self.long_list:

        cost_basis = self.Portfolio[self.trading_symbol].AveragePrice

        if  ((price <= float(0.995) * float(cost_basis)) or (price >= float(1.01) * float(cost_basis))):
            self.SetHoldings(self.trading_symbol, 0)
            self.long_list.remove(self.trading_symbol)
            self.Debug("liquidate long")
```

After testing multiple ways of running our models (single classifier and different combinations of the 3 classifiers), we realised that the model with the best performance is a combination of the 3 classifiers. Hence, we used ensemble voting method which combined the predictions for each model to decide on which trading position to take. This prevents overfitting of our model and improve our model's performance by reducing biases. Along with this, we included risk management strategies such as stop loss and take profits. This serves as a condition for our model when deciding whether to liquidate the position as mentioned in the model write-up.

We also tried our model using different currency pairs, after which we concluded that our model works the best with *'EURUSD'* since it performs better in stable environment as explained earlier.

```
# Reverse the Currency Pair Ratio
def reverseCurr(self, prices_in_df, extra_sym):

    for i in extra_sym:
        prices_in_df[i] = 1/prices_in_df[i]

    return prices_in_df
```

To further improve the user-friendliness and flexibility of our trading model from our prelims, we included more currency pairs other than the base currency symbols as well as a reverseCurr() function to reverse the ratio of the prices for the users to explore.

Strengths and Weaknesses

The Plus Side

We utilized the ensemble voting method before determining what is the position to take and whether to sell or buy the currency pair. This ensures that our algorithm is not overly biased and makes decisions based on only one algorithm, reducing the risk of taking the wrong position during the trading process.

In times of stable market conditions and a stable currency pair, our model is able to make more accurate predictions compared to complex algorithms as complex algorithms tend to overfit in stable environments (Ibid.).

Making use of simpler algorithms also speeds up the rate at which our model carries out trading and prevents potential loss of ticks in a trading environment where even the seconds are taken into account. Interpretability is also higher, allowing users who might not have a strong grasp of machine learning concepts to have an easier time understanding how to use the model.

The Flip Side

The model we used does not perform well in volatile environments as simpler machine learning algorithms are not sophisticated enough to catch unseen circumstances reliably (Ibid.).

In addition, equal weightages were given to all 3 models, and a decision was made only when all 3 were unanimous. Implementation of soft voting can be done, whereby the better the performance of the model, the higher the weightage assigned to that model (Ibid.). Each model then predicts the probability of each signal (taking a short position, long position, and holding onto current position). The average of probabilities for each signal is then calculated taking the weights into consideration, and then used to generate the final signal. This can help to further reduce inaccuracies while at the same time taking into consideration other models' predictions rather than just one prediction.

Given More Time

Apart from the inclusion of more machine learning model in our ensemble that can be achieved from further research and experimentation, there are still certain features of our model that we can still improve further.

From our Model Write-Up,

```
''' Download Historical Data [User-Defined Resolution] '''
currency_slices = self.History([i], self.data_period, Resolution.Daily)
```

We have explained that our model's historical data intake only works in Daily Resolution.

In reality, we should be able to change between different Resolution types. However, to achieve this flexibility we would require much more complex and sophisticated data manipulation tailored to our model.

In addition,

```
'''
Get Attribute (Feature Selection) [User-Defined]
NOTE: This is a 'Choose-One'
NOTE: Multi-Attribute Features can be a Future Improvement

Possible Features to Choose From:
    high, low, close
    askopen, askhigh, asklow, askclose
    bidopen, bidhigh, bidlow, bidclose
'''
currency_close = currency_bars['close']
```

our pre-set user-defined feature selection only works with ONE feature to be selected from the currency bar taken from the History function.

In reality, we should be able to have more than one feature to add complexity to our model training. However, to achieve this level of complexity we would require much more complex and sophisticated data manipulation tailored to our model.

Both of these stated improvements could have been done given more time, experience and exposure.

With time, we can also delve deeper into understanding the study of forex markets, even experimenting more with many other aspects of forex trading. We recognize our inexperience and at the same time, acknowledge that experience is key in learning the ways of forex trading. Building stronger technical foundations and getting much more attuned to and comfortable in a trading platform (QuantConnect) may prove to be evidently beneficial to us.

Food for Thought

Backtest vs. Live

During the Live period, the average returns was -0.665% over a span of two weeks. For the backtest from 7th November to 21st November, we had a positive return rate of 1.76% and a net profit of $1,166.67. Thus, our model performed much better during the backtest period.

The reason being, our model was built to make use of daily resolution to perform trades per day. However, during the initial live period when we encountered runtime issues on our own server, we sought for help and were told to change our resolution to hourly, which our model was not designed for, in an attempt to troubleshoot. Afterwards, we were informed that we no longer had to run the model by ourselves. Thus, we changed our resolution back to Daily and shared the model again before the models went live. However, looking at the order log uploaded, it seems that our model was running on the hourly resolution instead of daily:

Deploy	Time	Symbol	Price	Quantity	Type	Status	Value
L-be7e29	2018-11-12T08:00:00.002854Z	EURUSD	1.12574	-64032	Market	Filled	-72083.4
L-be7e29	2018-11-12T08:00:00.002854Z	EURUSD	1.12587	64032	Market	Filled	72091.71
L-be7e29	2018-11-12T09:00:00.000591Z	EURUSD	1.12597	-64046	Market	Filled	-72113.9
L-be7e29	2018-11-12T09:00:00.000591Z	EURUSD	1.1261	64046	Market	Filled	72122.2
L-be7e29	2018-11-12T10:00:00.000535Z	EURUSD	1.12548	-64058	Market	Filled	-72096
L-be7e29	2018-11-12T10:00:00.000535Z	EURUSD	1.12561	64058	Market	Filled	72104.33
L-be7e29	2018-11-12T11:00:00.00039Z	EURUSD	1.12595	-64058	Market	Filled	-72126.1
L-be7e29	2018-11-12T11:00:00.00039Z	EURUSD	1.12607	64058	Market	Filled	72133.79
L-be7e29	2018-11-12T12:00:00.069769Z	EURUSD	1.12612	-64061	Market	Filled	-72140.4
L-be7e29	2018-11-12T12:00:00.069769Z	EURUSD	1.12624	64061	Market	Filled	72148.06
L-be7e29	2018-11-12T13:00:00.089176Z	EURUSD	1.127	-64039	Market	Filled	-72172
L-9f35f68	2018-11-12T18:00:00.000475Z	EURUSD	1.12426	-60	Market	Filled	-67.4556
L-d4cac51	2018-11-14T11:00:00.000833Z	EURUSD	1.12689	62	Market	Filled	69.86718
L-1614797	2018-11-15T04:00:00.000525Z	EURUSD	1.13318	108	Market	Filled	122.3834
L-a6c4ad9	2018-11-15T16:00:00.000847Z	EURUSD	1.13183	-98	Market	Filled	-110.919
L-a6c4ad9	2018-11-15T16:00:00.000847Z	EURUSD	1.13194	64027	Market	Filled	72474.72
L-0af56e3	2018-11-16T03:00:00.00799Z	EURUSD	1.13345	-63657	Market	Filled	-72152
L-23ed9a5	2018-11-16T16:00:00.000387Z	EURUSD	1.13955	190	Market	Filled	216.5145
L-23ed9a5	2018-11-18T23:00:00.000011Z	EURUSD	1.14177	63467	Market	Filled	72464.72
L-23ed9a5	2018-11-19T00:00:00.000049Z	EURUSD	1.14129	63323	Market	Filled	72269.91

Trades were made by hour.

Trades Summary Download Trades

Date Time	Symbol	Type	Price	Quantity	Operation	Status	Tag
2018-11-07 00:00:00	EURUSD	Market	$1.14563 USD	-87,000	Sell	Filled	
2018-11-10 00:00:00	EURUSD	Market On Open	$1.13222 USD	87,000	Buy	Filled	
2018-11-20 00:00:00	EURUSD	Market	$1.1445 USD	-88,000	Sell	Filled	

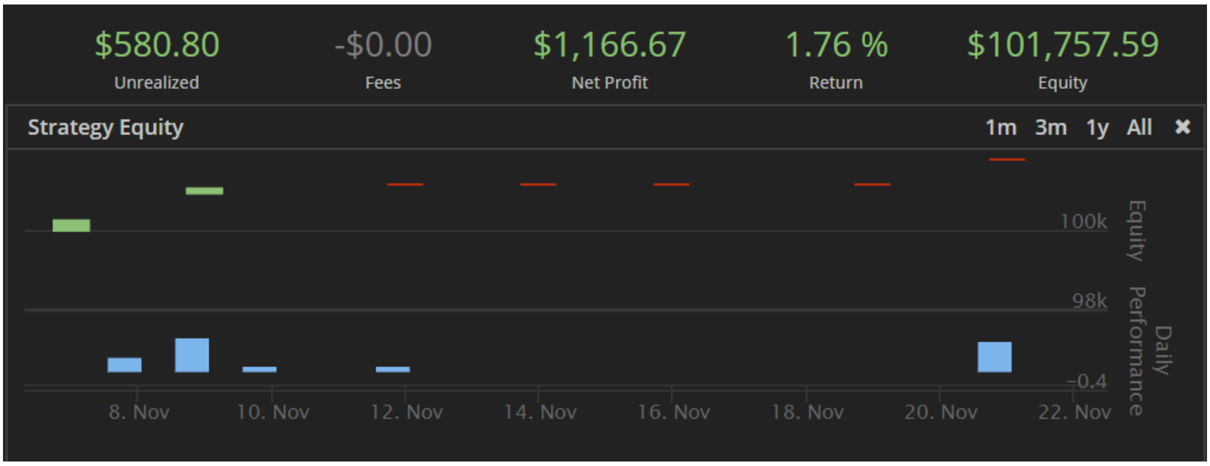
During the back test period from 7th November to 21st November, we made 3 trades and all 3 trades were carried out on 3 different days during midnight.

This resulted in the live-trading results faring worse than our back test results.

We only realised that the hourly model might have been deployed instead of the correct one when the live-trading log for every group was uploaded onto IVLE 2 days before the submission of the project, which by then was already too late to voice out.

From this however, we can tell that our model is indeed suited for daily resolution, and does not fare well in hourly resolution. We believe that our model will perform better if the daily model was deployed, and feel regrettable that such poor live results were obtained.

Backtest Analysis



Trades Summary Download Trades

Date Time	Symbol	Type	Price	Quantity	Operation	Status	Tag
2018-11-07 00:00:00	EURUSD	Market	$1.14563 USD	-87,000	Sell	Filled	
2018-11-10 00:00:00	EURUSD	Market On Open	$1.13222 USD	87,000	Buy	Filled	
2018-11-20 00:00:00	EURUSD	Market	$1.1445 USD	-88,000	Sell	Filled	

The 3 major trades carried out during the backtesting period were taking a short position on 7th November (i.e. selling the currency pair), liquidating the short position on 10th November (i.e. buying back the currency pair), and taking a short position again on 20th November. To decide whether to take a short position or not, we made use of the voting method. If all 3 models were unanimous in providing a -1 signal, it shows that all models predict that the price will drop further, thus making it profitable to sell now at a higher price and buy back the currency pair at a lower price in the future. A short position is hence taken. The model liquidated the short position on 10th Nov as the price of the currency pair ($1.133555) was below the threshold of 99% of the cost basis ($1.14563), making it advantageous to buy it at a lower price and wait for the price to increase to sell it again at a higher price to make profit.

Decision Analysis

The model has employed the straightforward use of stop-loss & take-profit strategies alongside the cost basis of our portfolio to determine the execution of liquidation of existing assets.

However, the call to action for taking on long, short and even holding positions goes through a much more complex and sophisticated process to determine. Generation of the technical indicators requires complex computational processes which proved to be a difficult feat for the average human to achieve, justifying the need for computing power that helps us to discern what we cannot from just looking at the price charts.

Our model made use of PCA to come up with the covariance matrix, eigenvectors, and eigenvalues that determines the projection scores to be used alongside the base technical indicators and in turn, generation of the stop, long, or hold signals. All of these variables and data generated, especially the eigenvectors (principal components) to minimize dimensionality yet preserve most information, require computational power for elegant data handling and sophisticated methods to derive.

As such, it is rather difficult for us to see how the percentage changes in the projection scores, which in turn affects our model's decision to take a certain position, were generated. Nevertheless, algorithmic trading serves to execute actions for humans by accurately identifying trading opportunities perceived by their individual designers. Hence, we put trust and confidence in our model and its decision-making process.

Model Write-Up

Instructions in Running the Model

```
'''
NOTE:
    BLOCK COMMENTS   - USER-DEFINED PARAMETERS
                     - DEBUG BLOCKS
    LINE COMMENTS    - CODE-SNIPPET DOCUMENTATION
'''
```

To start off with, our code has included 2 types of in-text comments to improve user-friendliness. All users have to do is to identify sections with the block comments for user-defined settings before running the model.

The Foundations

```
''' Set Start Date [User-Defined] '''
self.SetStartDate(2018, 10, 23)
''' Set End Date [User-Defined] '''
self.SetEndDate(2018, 11, 6)
''' Set Strategy Cash [User-Defined] '''
self.SetCash(100000)
```

Set the **Start** and **End** dates, along with the starting **Cash**.

Base Currency Symbols

```
'''
Currencies Selected to Provide Extra Information for Prediction Model [User-Defined]
'''
self.symbols = ["NZDUSD", "AUDUSD", "GBPUSD", "EURUSD",]
```

This is the **combination of currency pairs** that is deemed to be the **best** through various trial and errors that **should always be used**.

In our case, we are trading *'EURUSD'* and hence, the 4 currency pairs shown above relating to *'USD'* are what we identified as the best combination of currency pairs through many iterations of tests.

Extra Currency Symbols

```
'''
Extra Currency Pairs to be Chosen by User for Extra Features [User-Defined]
NOTE: The Best-Found Combination is the 4 Above.
NOTE: Add Currency Pair from List Below into 'Self.Extra_Sym'
NOTE: Overcrowding Currency Pairs May Cause Overfitting

List of Currency Pairs to Choose From:
    "USDSEK"
    "USDJPY"
    "USDCAD"
    "USDNOK"
    "USDSEK"
    "USDCHF"
    "USDZAR"
'''
self.extra_sym = []
self.symbols = self.symbols + self.extra_sym # Extend Symbols List
self.sym_n = len(self.extra_sym) # For Dynamically Locating Signals
```

On top of the Base Currency Symbols, **additional currency pairs** can be added to the extra_sym list for **further tests and experimentation**.

In our case, we are trading *'EURUSD'* and hence, a list of currency pairs relating to 'USD' is shown to improve user-friendliness.

Trading Symbol

```
''' Target Trading Symbol [User-Defined] '''
self.trading_symbol = "EURUSD"
```

Set your target **Trading Symbol**.

In our case, it is *'EURUSD'*.

Asset Resolution

```
for i in self.symbols:
    ''' [User-Defined Resolution] '''
    self.AddForex(i, Resolution.Daily)
```

Set your **Asset Resolution**.

Periods

```
''' Historical Data Period [User-Defined] '''
self.data_period = 300
''' Rolling Window Period [User-Defined] '''
self.window_period = 55
```

Set your **Historical Data** period, the amount of data the model calls for training.

Set your **Rolling Window** period for model training.

Historical Data Resolution

```
''' Download Historical Data [User-Defined Resolution] '''
currency_slices = self.History([i], self.data_period, Resolution.Daily)
```

For our model, our pre-set only works with historical data in *Daily Resolution*.

In reality, we should be able to change between different Resolution types. However, to achieve this flexibility we would require much more complex and sophisticated data manipulation. This will serve as a future improvement we can implement to our model.

Feature Selection

```
'''
Get Attribute (Feature Selection) [User-Defined]
NOTE: This is a 'Choose-One'
NOTE: Multi-Attribute Features can be a Future Improvement

Possible Features to Choose From:
    high, low, close
    askopen, askhigh, asklow, askclose
    bidopen, bidhigh, bidlow, bidclose
'''
currency_close = currency_bars['close']
```

For our model, our pre-set only works with *ONE* feature to be selected from the currency bar taken from the History function.

In reality, we should be able to have more than one feature to add complexity to our model training. However, to achieve this level of complexity we would require much more complex and sophisticated data manipulation. This will also serve as a future improve that we can implement to our model.

Data Munging

Standardizing Currency Symbols

```
        # Reverse all USD-base currency pairs
        prices_in_df = self.reverseCurr(prices_in_df, self.extra_sym)
```

In our model, we made sure to process data with *'USD'* as the **latter** in all currency pairs. Hence, we created a function to **reverse all symbols** with *'USD'* as the **former** in currency pairs as shown below.

```
    # Reverse the Currency Pair Ratio
    def reverseCurr(self, prices_in_df, extra_sym):

        for i in extra_sym:
            prices_in_df[i] = 1/prices_in_df[i]

        return prices_in_df
```

Scaling Data

```
        # Normalize all Values in DataFrame
        normalized_price = (prices_in_df - prices_in_df.mean())/prices_in_df.std()
```

In the PCA() function, we first **normalize** all data with through **z-score standardization** before proceeding to do exploratory data analysis on our data.

Null Values

```
        # Relative Strength Index
        RSI = talib.RSI(np.array(proj_scores), period)
        RSI = RSI[~np.isnan(RSI)]

        # Momentum
        MOM = (proj_scores / proj_scores.shift(period)).dropna() * 100

        # Moving Average Convergence-Divergence
        MACD_slow = period
        MACD_fast = int(floor(period * 0.5))
        MACD, MACD_signal, MACD_hist = talib.MACDEXT(np.array(proj_scores), fastperiod = MACD_fast, \
                                                    fastmatype = MA_Type.EMA, slowperiod = MACD_slow, \
                                                    slowmatype = MA_Type.EMA, signalperiod = 2, \
                                                    signalmatype = 0)
        MACD = MACD[~np.isnan(MACD)]
```

```
        percentage_change = proj_scores.pct_change().dropna() # Percentage Change Among Projection Scores
```

Much caution has been taken in dealing with null values, and generally the solution to take is **excluding them** since the data sample is big enough to compensate for the loss of data.

Feature Selection

```python
    # Generating Technical Indicators
    def indicator(self, period, proj_scores):

        # Relative position
        Relative_Position = []
        for i in range(period, len(proj_scores)):
            Relative_Position.append((proj_scores[i] - min(proj_scores[(i - period):i])) \
                                    /(max(proj_scores[(i - period):i]) - min(proj_scores[(i - period):i])))

        # Relative Strength Index
        RSI = talib.RSI(np.array(proj_scores), period)
        RSI = RSI[~np.isnan(RSI)]

        # Momentum
        MOM = (proj_scores / proj_scores.shift(period)).dropna() * 100

        # Moving Average Convergence-Divergence
        MACD_slow = period
        MACD_fast = int(floor(period * 0.5))
        MACD, MACD_signal, MACD_hist = talib.MACDEXT(np.array(proj_scores), fastperiod = MACD_fast, \
                                             fastmatype = MA_Type.EMA, slowperiod = MACD_slow, \
                                             slowmatype = MA_Type.EMA, signalperiod = 2, \
                                             signalmatype = 0)
        MACD = MACD[~np.isnan(MACD)]

        # All Technical Indicators as Independent Variables for Prediction Models in DataFrame
        predictor_in_df = pd.DataFrame({'RP': Relative_Position, 'RSI': RSI, 'MOM': MOM, 'MACD': MACD}, \
                                    index = proj_scores.index[period:])

        return predictor_in_df
```

```python
        '''
        Get Attribute (Feature Selection) [User-Defined]
        NOTE: This is a 'Choose-One'
        NOTE: Multi-Attribute Features can be a Future Improvement

        Possible Features to Choose From:
            high, low, close
            askopen, askhigh, asklow, askclose
            bidopen, bidhigh, bidlow, bidclose
        '''
        currency_close = currency_bars['close']
```

On top of the **user-defined feature** that users can select from the Instructions prior to this section, there are **4 major features** selected for model training. Namely, Relative Position, RSI, MOM and MACD.

Identifying Correlation

```python
        # Obtain Sample Covariance by
        # 1. Taking 'normalized_price' DataFrame as a Matrix
        # 2. Transposing it
        # 3. Executing Dot Product on Original Against Transposed 'normalized_price'
        # 4. Dividing by Length of Matrix
        covariance = normalized_price.T.dot(normalized_price) / (len(normalized_price) - 1)

        '''
        Debug 5: Checking if Prices are Normalized and Sample Covariance Values Are Obtained Properly
        '''
        #self.Debug("Normalized Price: ")
        #self.Debug(normalized_price.head(5))
        #self.Debug("Covariance: ")
        #self.Debug(covariance.head(5))

        # Retrieve Eigen Decomposition of Sample Covariance Matrix
        eigenvalues, eigenvectors = np.linalg.eig(covariance.dropna())

        # Retrieve Projection Scores by
        # 1. Taking 'normalized_price' DataFrame and 'eigenvectors[0]' as Matrices
        # 2. Transposing 'eigenvectors[0]'
        # 2. Executing Dot Product on 'normalized_price' Against the Transposed 'eigenvectors[0]'
        proj_scores = normalized_price.dot(eigenvectors[0].T)

        return proj_scores
```

In the PCA() function, a **covariance matrix** is generated using the normalized closing prices of the respective currency pairs in our portfolio.

The covariance is a measure of the **directional relationship between the returns on the different currency pairs**. A positive covariance means that the returns move together while a negative covariance means returns move inversely.

Afterwards, **eigen decomposition** is carried out on the covariance matrix to obtain the eigenvectors, otherwise known as the **principal components**, and their respective eigenvalues.

Every eigenvalue relating to their respective eigenvectors tells us the **importance** of that eigenvector in explaining the **correlation** between the currency pairs. The eigenvector with the largest eigenvalue will be the largest principal component, and also the direction along which the data has the most variance.

Thus, choosing this eigenvector **preserves most information** while **reducing dimensionality** within the data. We then find the **dot product** of normalized closing prices between currency pairs and the transposed largest eigenvector to get the dataset in its reduced dimensionality form in terms of the largest principal component.

Use of Performance Measurements

Application

```python
        # For Each Window
        for i in range(0, no_of_windows - 2):

            # Retrieve Projection Scores & Weights from Principle Component Analysis
            proj_scores = self.PCA(prices_in_df[(i * period):((i + 3) * period)])

            # Retrieve Technical Indicators (Extended Features)
            predictor_in_df = self.indicator(period, proj_scores)

            ''' Get Train & Test Data [User-Defined Sizes] '''
            train_x = predictor_in_df[:int(len(predictor_in_df) / 2)]
            train_y = self.generateSignal(proj_scores[int(len(proj_scores) / 3 - 1):int(len(proj_scores) / 3 * 2
                                          threshold)
            test_x = predictor_in_df[int(len(predictor_in_df) / 2):]

            # Get List of Predictions
            prediction_in_list = self.classifier(model, train_x, train_y.values.ravel(), test_x)

            dates.extend(test_x.index)          # Populate List of Dates
            signals.extend(prediction_in_list)  # Populate List of Signals

        # Converting List of Signals & Dates into DataFrame
        return pd.DataFrame({'signal': signals}, index = dates)
```

```python
    # Generating Response Variables
    def generateSignal(self, proj_scores, threshold):

        signals = [] # List of Signals
        percentage_change = proj_scores.pct_change().dropna() # Percentage Change Among Projection Scores

        # Generate Signals Based on Percentage Change of Projection Scores
        '''
        Signals:
            1   -   Take Long Position
            0   -   Hold Position
            -1  -   Take Short Position
        '''
        for i in percentage_change:

            if i  > threshold:
                signals.append(1)

            elif i < -threshold:
                signals.append(-1)

            else:
                signals.append(0)

        # Returns a DataFrame Consisting of Signals and Percentage Change Indexes
        return pd.DataFrame({'signal': signals}, index = percentage_change.index)
```

Making use of **projection scores** generated from our PCA, our model generates the **response signals** appropriate for the projection scores before sending them to train our models alongside the **combination of features** we have generated from the indicator() function and our user-defined currency bar attribute.

Overfitting

```python
        '''
        Signals:
            1   -   Take Long Position
            0   -   Hold Position
            -1  -   Take Short Position
        '''
        signals_DT_in_df = self.predict('DecisionTree', prices_in_df, period = self.window_period, threshold = 0.03)
        signals_RF_in_df = self.predict('RandomForest', prices_in_df, period = self.window_period, threshold = 0.03)
        signals_LR_in_df = self.predict('LogisticRegression', prices_in_df, period =  self.window_period, threshold = 0.03)

        # Concatenate all Information into a Master Table
        prices_in_df = prices_in_df[-len(signals_DT_in_df):]
        master_table = pd.concat([prices_in_df, signals_DT_in_df, signals_RF_in_df, signals_LR_in_df], axis = 1).dropna()
```

Firstly, complex Machine Learning models perform better in volatile environments but tend to overfit in stable environments (Ibid.). Additionally, simpler Machine Learning models - although unable to learn from unexpected circumstances accurately - are able to perform better in stable environments (Ibid.). The constituents of our **ensemble method** that we employed are **simpler machine learning models** namely, Decision Tree, Random Forest and Logistic Regression that works well with stable currency pairs like *'EURUSD'*.

Secondly, the use of ensemble method encourages **a voting system that reduces bias and potential overfitting damages**.

Risk Management

```
        # Calls to Run Ensemble - Decision Tree, Random Forest, Logistic Regression [User-Defined Threshold]
        # Returns a DataFrame of Predicted Signals Based on Selected Features
        '''
        Signals:
            1   -   Take Long Position
            0   -   Hold Position
           -1   -   Take Short Position
        '''
        signals_DT_in_df = self.predict('DecisionTree', prices_in_df, period = self.window_period, threshold = 0.03)
        signals_RF_in_df = self.predict('RandomForest', prices_in_df, period = self.window_period, threshold = 0.03)
        signals_LR_in_df = self.predict('LogisticRegression', prices_in_df, period =  self.window_period, threshold = 0.03)

        # Concatenate all Information into a Master Table
        prices_in_df = prices_in_df[-len(signals_DT_in_df):]
        master_table = pd.concat([prices_in_df, signals_DT_in_df, signals_RF_in_df, signals_LR_in_df], axis = 1).dropna()
```

```
        if master_table.tail(1).iloc[0][4 + self.sym_n] == 1.0 and \
           master_table.tail(1).iloc[0][5 + self.sym_n] == 1.0 and \
           master_table.tail(1).iloc[0][6 + self.sym_n] == 1.0 and \
           self.trading_symbol not in self.long_list and \
           self.trading_symbol not in self.short_list :

            self.SetHoldings(self.trading_symbol, 1)
            self.long_list.append(self.trading_symbol)
            self.Debug("long")

        if self.trading_symbol in self.long_list:

                cost_basis = self.Portfolio[self.trading_symbol].AveragePrice

                if  ((price <= float(0.995) * float(cost_basis)) or (price >= float(1.01) * float(cost_basis))):
                    self.SetHoldings(self.trading_symbol, 0)
                    self.long_list.remove(self.trading_symbol)
                    self.Debug("liquidate long")

        if master_table.tail(1).iloc[0][4 + self.sym_n] == -1.0 and \
           master_table.tail(1).iloc[0][5 + self.sym_n] == -1.0 and \
           master_table.tail(1).iloc[0][6 + self.sym_n] == -1.0 and \
           self.trading_symbol not in self.long_list and \
           self.trading_symbol not in self.short_list :

            self.SetHoldings(self.trading_symbol, -1)
            self.short_list.append(self.trading_symbol)
            self.Debug("short")

        if self.trading_symbol in self.short_list:

                cost_basis = self.Portfolio[self.trading_symbol].AveragePrice

                if  ((price <= float(0.99) * float(cost_basis)) or (price >= float(1.005) * float(cost_basis))):
                    self.SetHoldings(self.trading_symbol, 0)
                    self.short_list.remove(self.trading_symbol)
                    self.Debug("liquidate short")
```

After concatenating the signals by all 3 models into a master table, we made used of a **voting system based on ensemble learning method** to determine what actions to take. If the **3 signals given by the models for that data set are all '1'** and that the currency-pair is **not in the long or short list**, then a **long position will be taken**, and the currency-pair will be added into the list of currency-pairs that the user has taken a long position. After which, the price of the currency-pair is checked against 2 conditions:

1. Depending on **whether we have positions Long**,
2. For positions in Long, we **liquidate**:
    - If current price of the currency we are trading is **equal to or below** a threshold of **99.5% of our cost-basis (stop-loss)**
    - If current price of the currency we are trading is **equal to or above** a threshold of **101% of our cost-basis (take-profit)**

In this case, we take the cost basis to be the average price of the currency-pair. When **either one of the conditions is met**, it will **liquidate the long position** in which the base currency is sold to the market.

On the other hand, if all **3 signals given by the models for that data are '-1'** and that the currency-pair is **not in the long or short list**, a **short position will be taken**, and currency-pair will be added to the list of currency-pairs with short position. Similarly, the price of the currency pair is then checked against another 2 conditions:

1. Depending on **whether we have positions Short**,
2. For positions in Short, we **liquidate**:
   - If current price of the currency we are trading is **equal to or below** a threshold of **99% of our cost-basis (take-profit)**
   - If current price of the currency we are trading is **equal to or above** a threshold of **100.5% of our cost-basis (stop-loss)**

When **either one of the conditions mentioned earlier is satisfied**, it will **liquidate short position** in which the base currency will be purchased.

- End of Report -

REFERENCES

Gerlein, E. A., Mcginnity, M., Belatreche, A., & Coleman, S. (2016). Evaluating machine learning classification for financial trading: an empirical approach. Retrieved from https://www.sciencedirect.com/science/article/pii/S0957417416000282