**Model**

With the data set up in the previous section, the central question of the research becomes a supervised binary classification problem: we aim to predict the output variable given input variables, through machine learning models. In this section, I will give an introduction to the models that I used in this research. Also, I will explain a special modelling challenge in this research — the imbalanced data challenge, and the way to mitigate the problem.

Model selection

In this research, I fit three models to the data: logistic regression, Random Forests (RF), and Recurrent Neural Networks (RNN).

Logistic regression

Logistic regression is one of the most widely used binary classifiers. It is easy to implement, and the interpretation of its results is straightforward. Logistic regression models the probability of the binary output variable through the logistic transformation. Equation~\ref{eq:logit} shows the formula of the logistic regression, where X is the design matrix of the input variables and Y is the dummy variable of the bubble. From the formula, we see that the logistic transformation of the probability of bubbles is just the log-odds of bubbles. Therefore, we can also interpret the logistic regression as modelling the log-odds by linear models.

While the logistic regression is easy to implement, it is limited in that it does not capture the non-linear pattern in the data. The relationship between the input variables and the dependent variables (after the logistic transformation) is restricted to a linear form. However, bubbles and financial markets are complicated, and it is reasonable that the true relationship is beyond linearity. This is why I employ more complex models such as Random Forests and Recurrent Neural Networks aiming to model the non-linear pattern in the data.

Random Forests

The RF is an ensemble learning method for decision trees. It fits many decision trees to the data through bootstrap and randomly excluding some features. Then, it aggregates the predictive results of all decision trees to yield the final result. Besides being a non-linear classifier, the main benefit of the RF is that the aggregation mitigates the overfitting problem of the simple decision trees method. According to a study conducted by \textcite{LinSurvey}, soft classification techniques, including ensemble linear methods, “appear to be the direction for future research” in the context of financial crisis prediction. As a strong ensemble learning classifier, the RF method is a suitable candidate for this research.

Recurrent Neural Networks

Deep learning models, or artificial neural networks with multiple hidden layers, are becoming popular in financial research. They are often shown to have superior results and traditional methods. The RNN is a family of deep learning methods that are good at predicting sequential natured data, including time-series data. In this research, I choose a specific type of RNNs — RNNs with bidirectional long short-term memory networks (RNN + BiLSTM). The long short-term memory (LSTM) is a neural network unit that helps the model to learn from the long-term pattern in the data. The bidirectional component enables the model to detect patterns in the sequential data from both directions. Readers who are interested in knowing more about how the RNN model works can refer to the appendix written by \textcite{Bash2020} and the article of \textcite{lstm}.

In the context of forecasting financial time series, previous research has shown that RNNs with BiLSTM have generally better results than traditional time series prediction methods like ARIMA. \parencite{Siami2019} A recent asset bubbles detection study uses this method and gets has satisfactory results. \parencite{Bash2020} Based on previous studies and the nature of my study, I select this method as a candidate for the best classifier. However, as we will see in the result section, the performance of the RNN-BiLSTM model is disappointing.

The challenge of imbalanced data

Apart from the model selection, this research is also faced with a special challenge of imbalanced data. As shown in Table ~\ref{tab:imbalance}, the distribution of the output variable is highly imbalanced. There are way more zeros (“non-bubble”) than ones (“bubble”). This imbalanced distribution is expected since bubbles and market crashes are rare phenomena, yet it could cause trouble for the modelling.

The default behaviour of a typical machine learning model fitting is to minimize the average loss to all data training data. Since the non-bubble case represents the over-majority of the data, without any special treatments, the models are expected to focus more on the non-bubble cases than the bubble cases. However, it is undesirable because the research is interested in predicting bubbles. As an extreme case, even a naïve classifier that predicts zero in all cases would have a high accuracy rate. Its prediction is right most of the time since the market does not have bubbles most of the time. Of course, the naïve classifier is completely useless and offers no practical value. To make the result meaningful, we want to make the predictors less prone to be useless naïve predictors.

Besides, from a practical perspective, the loss caused by the failure of predicting a market crash is more important than that caused by a false alarm. The latter may lead to an over-conservative investment or policy decisions that limit the gains, but an unexpected market crash can cause huge financial loss and social costs. This practical concern also suggests that we should focus more on correctly predicting the bubbles than the default model-fitting behaviours based on output variable distribution.

To tackle this challenge, I have tried a couple of ways, including changing decision thresholds, re-sampling for re-weighting, and adopting an asymmetric loss function. These methods essentially make the models focus more on the bubble cases than the non-bubble cases. Each model should only adopt one of the above methods to address the imbalanced data problem; otherwise, there will be an overcorrection. Also, to reflect the preference for the prediction result in bubble cases, I use the balanced accuracy rate rather than the overall accuracy rate as my model evaluation metrics.

Changing the decision thresholds

For all the binary classification methods that I select, they would first predict a probability of the bubble before making a categorical decision. The default decision threshold, as suggested by Baynes’ theorem of classification, is 0.5. For example, given a specific set of input data, the model may predict that the probability of the existence of bubbles is 0.4. Since the 0.4 is smaller than 0.5, the model would make a categorical prediction of non-bubble.

We can lower the decision threshold to make the model more likely to make a “bubble” decision. A lower bubble probability is enough to trigger an alarm of the bubble. If we changed the decision threshold to 0.3, in the above example, the model would give a “bubble” prediction since 0.4 is greater than 0.3. In this way, we make the model treat the bubble probability with higher weights. It can offset the over-representation problem of non-bubble cases.

How to decide the new decision threshold? Setting the threshold to 0.3 seems arbitrary. In this research, I use two methods to determine the new threshold. First, the model can use the prevalence of the bubble cases, i.e., the frequency of 0, as the threshold. This would offset the imbalanced data problem completely. Also, the model can treat the threshold as a hyperparameter and use cross-validation (CV) to tune it. This method is particularly suitable for the RF model because the out-of-bag samples of the RF can be easily used for the CV.

Re-sampling for re-weighting

We can also directly change the re-weight in the data through resampling. For each bubble case, we sample the data more than once, while in the non-bubble case we only sample it once. Essentially, the method replicates the bubble data. Using this method, we can make the ratio between the bubble cases and non-bubble cases close to one. As a result, the data is almost perfectly balanced.

Asymmetric loss function

A loss function for classification problems is a function that characterizes the misclassification. When fitting a classifier, we minimize the loss function on the training data to make a better prediction. The default loss function for most binary classification algorithms is the binary cross-entropy function. As shown in Equation~\ref{eq:entropy}, the loss function is symmetrical for the bubble misclassification and non-bubble misclassification.

If we want to make the model focus more on the bubble prediction, we can make the model treat a bubble misclassification as a bigger loss than a non-bubble misclassification. To achieve these, we can adopt an asymmetrical loss function. This method is most easy to implement in the RNN model since the TensorFlow package, a widely used deep learning library, enables us to use a customized loss function for model fitting. In this research, I use the focal loss function, which is proposed by \textcite{focal}. The function takes the form as Equation~\ref{eq:focal}. \(\alpha)\ and \(\gamma)\ are hypermeters that determine how asymmetric the loss should be and the further customized treatments for misclassification cases. I set them as the same as the original paper.

Model evaluation

I use the balanced accuracy as the model evaluation metric. It is defined as the average value of sensitivity and specificity. Sensitivity measures the proportion of correctly predicted positive (bubble) cases, while specificity measures the proportion of correctly predicted negative (non-bubble) cases. A formula form of the balanced accuracy is shown in Equation~\ref{eq:ba}.

Balanced Accuracy = True Negative/Negative + True Positive / Positive

This metrics is preferred compared to a simple, overall accuracy rate. It separately measures the model performance on positive (bubble) cases and on negative (non-bubble) cases. Under this metric, a naïve classifier that predicts all cases as non-bubble can only achieve 50% since it has 0 sensitivity.