Extreme Stock Returns Forecasting with Machine Learning

Yuan Chen

Email: chen2243@wisc.edu

Campus ID: 9082757429

Introduction

Finance Practitioners, especially those who engage in risk management, are concerned with extreme stock returns: jackpot returns and crashes, defined as extremely good and bad returns. Accurately predicting jackpots and crashes facilitates risk management practice as well as an equity investment. Each individual stock has different volatility. Hence, extreme returns for each stock are different. For example, with the same mean of 0%, the chance of a 5% return on a stock with a standard deviation of 1% is much less than that on a stock with a standard deviation of 20%. Hence, instead of using absolute stock returns such as 10% or 15% per day, we calculate the 10^{th} and 90^{th} percentile of the historical daily stock returns over the past year for each individual stock. Then we define jackpot returns as unexpected daily returns on an individual stock that is higher than the 90^{th} percentile of the historical daily stock returns over the past year for this stock and crashes as unexpected daily returns on an individual stock that are lower than the 10^{th} percentile of the historical daily returns over the past year for this stock, respectively.

We utilize several market liquidity indicators and firm financial characteristics documented in previous financial literature as our predictors. Market liquidity indicators include one trading day lagged bid-ask-spread defined by the difference between daily ask quote and daily bid quote scaled by the ask quote (e.g., Clark, Robert A., et al. (1992)), one trading day lagged liquidity defined as daily trading volume scaled by the total number of shares outstanding (e.g., Bong-Soo Lee and Oliver M. Rui (2002)), and one trading day lagged market capitalization defined as the product of stock price and the total number of shares outstanding (e.g., Fama and French (1992)). Firm financial characteristics include Book Leverage defined as total debt (Compustat item dlcq + dlttq) scaled by the total assets (Compustat item atq) over the past quarter (e.g. Welch (2004)), Market Leverage defined as total debt (Compustat item dlcq + dlttq) scaled by the total market value of assets (Compustat item atq – ceqq + cshoq* prccq) over the past quarter (e.g., Welch (2004)), book to market ratio defined as book value of equity (Compustat item ceqq) scaled by market capitalization (CRSP item abs(prc)*shrout) over the past quarter (e.g., Fama and French (1992)), and return on equity defined as Income Before Extraordinary Items (Compustat item ibq) scaled by one quarter lagged book value of equity (Compustat item ceqq) over the past quarter (e.g., Fama and French (2015), Hou et al. (2015)).

We focus on jackpot forecasting by defining jackpots as one and other returns as zero. We use Random Forest, Kth nearest neighbors algorithm (KNN), Support Vector Machine, and Neural Network to forecast jackpots returns. Since lots of observable and unobservable factors in our economy and society affect stock market movements, none of the algorithms has extremely great performance in predicting extreme stock returns. However, we find that Random Forest performs the best among the four algorithms we explore in this work. It improves all three metrics – Accuracy, Precision, and Recall by a non-negligible amount if trained by the undersampled data.

Our work contributes to the literature that explores the jackpot returns. Previous finance literature focuses on exploring jackpots and crashes as explanatory variables. Conrad et al. (2014) find that investors

prefer stocks with a higher probability of jackpots. Bali et al. (2011) find that extreme positive returns in the past month is negatively and significantly related to the expected cross-sectional stock returns. As academic researchers focus on answering fundamental research questions, our work is the first to predict extreme stock returns using machine learning algorithms to our best knowledge. It facilitates the application of machine learning in investment practice.

Data and Methods

We obtain daily stock return data, as well as stock volume, bid-ask spread, short interest ratio data, from the Center for Research in Security Prices database (CRSP) and firm financial data such total assets, book value of short term debt, book value of long term debt, book value of equity, and income before extraordinary items from Compustat. The total dataset contains ~6000 stocks from 2000-2020, with 251 trading days in each year. This problem can be treated as a binary classification task with a One-Against-All strategy. If the daily return is on the top 10 percentile, it's labeled as 1, and 0 if it's in the lower 90 percentile. In this study, I compare the performance of some common ML algorithms: Random Forest (RF), K Nearest Neighbors (KNN), Support Vector Machine (SVM), and Neural Network (NN).

Random Forest:

Random Forest is an extension of the decision tree we learned in CS760. Even though a decision tree is a standard supervised learning algorithm, it often suffers from overfitting and low generalization accuracy. In the Random Forest algorithm, a dataset partition is sampled randomly to train a decision tree. Then a set of similar decision trees are built to form an ensemble. The prediction is made by collecting predicted labels from each tree and deciding the final class by majority vote. Random Forest offers better Accuracy than decision trees and is often used in recent finance research with machine learning elements. In this work, the RandomForestClassifier from Scikit-Learn library is used.

K Nearest Neighbors

In KNN classification, each data point's label is assigned by the majority of its k nearest neighbors. And the prediction is made in the same way. The classification is done by the KneighborsClassifier package from sklearn.neighbors.

Support Vector Machine

SVM is a simple yet effective classification algorithm in many cases. Many solutions exist in standard Logistic Regression algorithms and lead to poor generalization of the classifier. SVM looks for the max margin (gap) between the support vectors, and this provides a restrain to the decision boundary obtained by the training and improves the performance. The SVC and SGDClassifier with hinge loss from the Scikit-Learn library are used in this study.

Neural Network

Neural Network is one of the most popular machine learning algorithms in recent years, and it offers unparalleled Accuracy for many challenging problems and is the core of deep learning. Due to the complexity of the dataset, Neural Network could be the best tool to achieve decent Accuracy with high speed. And the existing frameworks are well equipped to deal with a large dataset with batch training implementations. In this work, the Pytorch framework is used to train the dataset.

Oversampling and Undersampling

Due to the nature of the dataset in question, we are looking for the top 10 percentile of the return. The dataset is imbalanced data with only ~10% data that should be positively classified. Training on an imbalanced dataset usually causes a model to perform poorly on the minority class. Therefore, I conducted data augmentation in this study using Synthetic Minority Oversampling Technique (SMOTE) to balance the data. In the meantime, some studies also suggest undersampling of the dataset simultaneously to gain even better results. I will explore both methods and compare them to the model trained by original data. The oversampling and undersampling of data is achieved by SMOTE and the RandomUnderSampler function in the imbalance learn library.

Cross-Validation

The hyperparameters in each learning algorithm are tuned by 5-fold cross-validation. One of the data files is selected and be split randomly by the Kfold function from sklearn.model_selection. Four of the partitions are used to train, and one is used to test the result. The same process is repeated by five times with different test partitions, and the average of Accuracy, Precision, and Recall are reported to determine the best value of the hyperparameter.

Training and Data Analysis

Data preprocessing

The original data obtained from CRSP contains numerous errors and empty entries. Because most of them are in series, I can't replace them with mean or other meaningful values. Fortunately, compared to the large dataset, the 'wrong' data are still just a small portion of the entire dataset, so I can simply remove them. Then labels are added by comparing the return (RET) and the threshold of 10 percentile returns in the past year (PAST_y_p90). If the return is larger, label the data point with 1, and 0 otherwise. Finally, the data from each year are randomly separated into an 80% training set and 20% testing set. The two sets are saved into two separate files. After the data preprocessing, the percentage of positive labels of each year's data are calculated and listed in Table 1, and the average is ~10.4%.

Table 1 Percentage of positive label by year

2000	**2001**	**2002**	**2003**	**2004**	**2005**	**2006**	**2007**	**2008**	**2009**	**2010**
0.114	0.090	0.100	0.082	0.090	0.096	0.101	0.116	0.160	0.083	0.079
2011	**2012**	**2013**	**2014**	**2015**	**2016**	**2017**	**2018**	**2019**	**2020**	**Avg**
0.116	0.074	0.093	0.100	0.113	0.105	0.088	0.112	0.103	0.169	0.104

Random Forest

I start the Random Forest training by tuning the hyperparameters using cross-validation (cv) using the single-year data from 2020: 1) Ratio of oversampling and undersampling; 2) the number of trees in the Random Forest; 3) the number of splits in the Random Forest. The default values used in the cross-validation: number of trees = 100, number of splits = 2. The effects of various ratios of oversampling and undersampling are shown in Table 2. The table indicates that when the data is imbalanced, the algorithm can reach very high Accuracy when predicting the testing data. However, it's not a desirable result since 90% of the testing data are labeled as 0. The model trained by such imbalanced data can predict every data as 0 and lead to a 90% accuracy. So I have to take Precision and recall into consideration. It's clear

that oversampling alone (The diagonal) only slightly improves the Recall, and undersampling has a greater effect on how many label 1s are correctly predicted. In fact, another round cv with o=0.1 and u=1.0 resulting an Accuracy=0.546, Precision=0.118, and Recall=0.529. It appears the results are always a trade-off between Accuracy and Recall, if one increases, the other declines. Precision, on the contrary, is best when the data is not changed. Things get tricky here, what do we prioritize? A high Recall means we have more positive labels classified correctly, while a higher precision means I'm less likely to classify a 0 to be 1. From an investor's point of view, I may choose Precision over Recall because I have fewer false positives and am less likely to lose money because of a false positive. And I don't need to know all the stocks with extreme return, just a few of them to invest my money. Based on this perspective, I focus on the high Precision result in the rest of the study. However, I will also show the results that generate high Recall for comparison. The results in Table 2 show that oversampling provides very limited improvement over Recall, while undersampling provide significant boost to Recall. Thus, the data are not oversampled for the rest of the study. Instead, data are undersampled to a 1:1 ratio between 0 and 1 labels.

Table 2 Effect of oversampling (o) and undersampling (u)

	u=0.2	*u=0.4*	*u=0.6*	*u=0.8*	*u=1.0*
o=0.2	0.889 0.157 0.014	0.867 0.118 0.045	0.820 0.131 0.134	0.735 0.116 0.236	0.643 0.117 0.337
o=0.4		0.875 0.123 0.035	0.839 0.112 0.081	0.786 0.112 0.157	0.725 0.109 0.233
o=0.6			0.851 0.122 0.072	0.809 0.108 0.116	0.758 0.112 0.194
o=0.8				0.823 0.116 0.109	0.785 0.112 0.158
o=1.0					0.796 0.113 0.143

The three numbers in each cell represent the Accuracy, Precision, and Recall of the prediction. The number in oversampling corresponds to the percentage of the minority over the total sample size. The number in undersampling corresponds to the ratio between minority and majority data after the operation.

The effect of the number of trees and number of splits are tested by cross-validation as well, and the results are listed in Table 3 and Table 4. From the tables, we can see the best values are 80 trees and 3 splits, respectively.

Table 3 Effect of number of trees in the Random Forest

	20	**40**	**60**	**80**	**100**	**120**	**160**	**180**	**200**
ACCURACY	0.576	0.560	0.545	0.554	0.539	0.555	0.547	0.524	0.542
PRECISION	0.117	0.113	0.116	0.118	0.111	0.116	0.112	0.110	0.114
RECALL	0.476	0.479	0.518	0.518	0.500	0.503	0.490	0.511	0.508

Table 4 Effect of number of splits in the Random Forest

	2	3	4	5
ACCURACY	0.540	0.534	0.527	0.536
PRECISION	0.113	0.117	0.112	0.115
RECALL	0.505	0.541	0.521	0.522

Once the hyperparameters are determined, the values of undersampling, number of trees, and number of splits are set to 1.0, 80, and 3. Since the Random Forest package does not offer batch training function and a custom implementation is not trivial, I will train the data from each year individually and list the testing results in Tables 5 and 6. Using the original data, the model predicts the testing dataset with an average Accuracy of 89.5%, average Precision of 19.4%, and average Recall of 1.7%. Precision has a significant boost from the default 10% if predictions are generated randomly. The low Recall means, as expected, most positive labels are missed. The model trained by undersampled data predicts the testing dataset with an average Accuracy of 56.9% over 21 years, Precision with an average of 13.4%, and Recall with an average of 57.3%. All three metrics improve moderately over the 50%, 10%, 50% values if the predictions are generated randomly.

Table 5 Model performance using undersampled data

	2000	2001	2002	2003	2004	2005	2006	2007	2008	2009	2010
Accuracy	0.571	0.588	0.581	0.573	0.554	0.551	0.544	0.561	0.560	0.619	0.571
Precision	0.145	0.122	0.133	0.106	0.109	0.115	0.118	0.143	0.199	0.133	0.106
Recall	0.578	0.576	0.577	0.563	0.559	0.551	0.547	0.561	0.576	0.641	0.589
	2011	**2012**	**2013**	**2014**	**2015**	**2016**	**2017**	**2018**	**2019**	**2020**	**Avg**
Accuracy	0.582	0.570	0.553	0.562	0.557	0.575	0.552	0.553	0.559	0.615	0.569
Precision	0.157	0.097	0.113	0.126	0.137	0.138	0.107	0.135	0.127	0.245	0.134
Recall	0.594	0.570	0.550	0.567	0.556	0.579	0.560	0.555	0.558	0.618	0.573

Table 6 Random Forest performance using original data

	2000	2001	2002	2003	2004	2005	2006	2007	2008	2009	2010
Accuracy	0.883	0.907	0.896	0.915	0.909	0.901	0.896	0.88	0.832	0.913	0.918
Precision	0.219	0.236	0.218	0.17	0.174	0.164	0.161	0.197	0.243	0.217	0.13
Recall	0.018	0.014	0.014	0.007	0.007	0.008	0.008	0.013	0.024	0.016	0.006
	2011	**2012**	**2013**	**2014**	**2015**	**2016**	**2017**	**2018**	**2019**	**2020**	**Avg**
Accuracy	0.877	0.923	0.903	0.895	0.881	0.889	0.908	0.882	0.893	0.818	0.895
Precision	0.25	0.14	0.146	0.176	0.19	0.203	0.14	0.191	0.173	0.35	0.194
Recall	0.029	0.006	0.008	0.012	0.017	0.019	0.009	0.017	0.012	0.099	0.017

K Nearest Neighbors

In theory, KNN is the least prone to imbalanced data since each data point is only influenced by the nearby neighbors. The dataset's size or ratio should not be a problem while classifying each data point. Nonetheless, both original and undersampled data are still compared in this study.

A few parameters can be tuned in the KNN algorithm, while the number of neighbors used to classify is most important. I start the test with cross-validation to find the best k value with all the other hyperparameters set to the default by the KneighborsClassifier package. The results are shown in Table 7. Precision doesn't change much from 6 neighbors to 9 neighbors. After testing some other years' data, I decide to use 8 neighbors for this study.

Table 7 Effect of number of neighbors

	1	2	3	4	5	6	7	8	9	10
ACCURACY	0.82	0.888	0.873	0.893	0.889	0.895	0.894	0.896	0.895	0.896
PRECISION	0.119	0.131	0.129	0.14	0.139	0.144	0.144	0.145	0.146	0.139
RECALL	0.116	0.015	0.039	0.007	0.015	0.003	0.006	0.001	0.002	0

The model trained by original data provides an almost identical prediction compared to the Random Forest, with an average Accuracy of 89.5%, average Precision of 18.8%, and average Recall of 0.3%, all of which are equal or slightly worse than the Random Forest predictions. The model trained by the undersampled data is also somewhat worse than the Random Forest, which trades a 17.3% decrease of Recall for just a 5.5% increase in Accuracy. And both models have around 10% of Precision.

Table 8 KNN performance using undersampled data

	2000	2001	2002	2003	2004	2005	2006	2007	2008	2009	2010
Accuracy	0.623	0.636	0.628	0.632	0.621	0.619	0.62	0.62	0.613	0.648	0.621
Precision	0.128	0.107	0.115	0.094	0.096	0.102	0.108	0.126	0.179	0.109	0.087
Recall	0.405	0.412	0.406	0.401	0.386	0.38	0.382	0.388	0.394	0.448	0.397
	2011	**2012**	**2013**	**2014**	**2015**	**2016**	**2017**	**2018**	**2019**	**2020**	**Avg**
Accuracy	0.615	0.636	0.623	0.62	0.622	0.624	0.626	0.617	0.62	0.625	0.624
Precision	0.128	0.087	0.102	0.109	0.125	0.119	0.097	0.121	0.113	0.212	0.117
Recall	0.399	0.406	0.388	0.388	0.393	0.405	0.393	0.389	0.396	0.452	0.400

Table 9 KNN performance using original data

	2000	2001	2002	2003	2004	2005	2006	2007	2008	2009	2010
Accuracy	0.887	0.909	0.899	0.917	0.911	0.904	0.899	0.884	0.836	0.916	0.92
Precision	0.203	0.259	0.215	0.188	0.187	0.133	0.137	0.186	0.228	0.226	0.092
Recall	0.003	0.002	0.002	0.001	0.001	0.001	0.001	0.003	0.009	0.003	0
	2011	**2012**	**2013**	**2014**	**2015**	**2016**	**2017**	**2018**	**2019**	**2020**	**Avg**
Accuracy	0.883	0.925	0.906	0.899	0.887	0.894	0.912	0.888	0.897	0.827	0.895
Precision	0.133	0.207	0.211	0.162	0.184	0.176	0.141	0.216	0.159	0.299	0.188
Recall	0.002	0.001	0.001	0.001	0.002	0.002	0.001	0.002	0.002	0.021	0.003

Support Vector Machine

SVM is very prone to imbalanced data and leads to poor generalization. As shown in Table 5, the model trained by the imbalanced original data generates a very low Recall yet performs poorly on Accuracy. In the meantime, using the undersampled data with 1:1 classes, both of the Precision and the Recall improves significantly, and only a moderate decrease in Accuracy.

Table 10 Imbalanced original data vs. balanced data with undersampling using data2020

	IMBALANCED	BALANCED
ACCURACY	0.732	0.379
PRECISION	0.139	0.188
RECALL	0.115	0.812

The SGDClassifier from sklearn.linear_model is used to train the model. This package support batch training, and with a hinge loss function, it can train a large dataset batch by batch. I also tried the SCV package from sklearn.svm, which consumes too much time to classify each year's data. Thus, it is nearly impossible to tune the parameters and get a trained model in a reasonable time window. So the SVC package is not used in this study. To reduce the influence of outliers, I removed the years with black swan events like the 2009 financial crisis and 2019-2020 COVID-19 for the first training. The chunk size is set to 10^4, which means data points are read from the csv files each time and used to train the SGDClassifier. The results are listed in Table 6.

The SGDClassifer model trained by the whole dataset (minus the potential outliers) predicts the most recent year with exceptionally high Recall, with almost all positive labels identified. However, the low Accuracy and Precision implies that it tends to classify all the data points as 1. On top of that, the model predicts the early years with an extremely low success rate in all metrics. Clearly, It's not a reasonable model for my cause. It's reasonable to assume that policy, economic status, and other outside influences of the market changed quite a bit during the past years. The model trained from old data may not be useful at all.

For this reason, in the next step, I explore the time factor and try to find out how many prior years should be included to build a good model. During this process, I found that the SGDClassifier is very unstable. The Accuracy and Recall may swing 0.1 to 0.9 in each run when no parameter is changed. The randomness from the initial weight may play a significant role in the model. After tuning all the parameters in SGDClassifier, I can not find a stable training process and conclude that it may not be a good choice for this project.

Neural Network

Pytorch framework is used to program the Neural Network model. Due to the time limit, a simple model with only one hidden layer is used. The hidden layer contains 64 perceptrons. Data is transformed linearly between each layer, activated by the ReLu function, and normalized by the BatchNorm1d function. The final layer uses a sigmoid function to generate the prediction. BCEWithLogitsLoss from Pytorch is used as loss function, and I choose Adam as the optimizer. The chunk size used to read the data is set to 10^4, and batch size is set to 128.

Similar to the previous algorithms, the imbalanced and undersampled datasets produce significantly different results. As shown in Table 11, the original data trains a model with only 81.8% accuracy and can't predict the testing data at any decent rate. On the other hand, the balanced data trade ~14% Accuracy for a 35% increase in Recall and a slight increase in Precision.

Table 11 Imbalanced original data vs. balanced data with undersampling using data2000

	IMBALANCED	BALANCED
ACCURACY	0.818	0.679
PRECISION	0.195	0.226
RECALL	0.027	0.375

The second step is to determine the learning rate by cross-validation. A list of 11 learning rates with a range of [10^{-10}, 10^{-9} … 0.1, 1] is tested on both original and undersampled datasets. The results are listed in Tables 12 and 13. After examining the loss curve and the Precision, 10^{-3} will be used for undersampled data training and 10^{-4} for original data. Sometimes, the models produce no positive label at all and generate a "divide by zero" error. In this case, the "NA" value is filled in the table cell.

Table 12 Effect of different learning rates using undersampled data2000

	10^{-10}	10^{-9}	10^{-8}	10^{-7}	10^{-6}	10^{-5}	10^{-4}	10^{-3}	10^{-2}	10^{-1}	1
ACCURACY	0.514	0.497	0.515	0.514	0.515	0.526	0.515	0.515	0.516	0.500	0.500
PRECISION	0.511	0.598	0.511	0.526	0.516	0.565	0.626	0.669	0.674	0.617	0.500
RECALL	0.661	0.569	0.717	0.303	0.583	0.244	0.076	0.062	0.068	0.002	0.997

Table 13 Effect of different learning rates using original data2000

	10^{-10}	10^{-9}	10^{-8}	10^{-7}	10^{-6}	10^{-5}	10^{-4}	10^{-3}	10^{-2}	10^{-1}	1
ACC	0.797	0.765	0.377	0.541	0.35	0.851	0.886	NA	NA	NA	NA
PRE	0.137	0.136	0.119	0.112	0.117	0.159	0.158	NA	NA	NA	NA
REC	0.143	0.199	0.702	0.436	0.716	0.071	0	NA	NA	NA	NA

Then I tested two different training strategies, the first is using the full set of data to train the model, and the second is using each individual year to train the model. Then use the models are tested by the test dataset by year. The results are listed in Table 14 to Table 17. The model trained by full set of undersampled data shows a balanced prediction result, with an average of Accuracy of 66.8%, Precision of 13.1%, and Recall of 38.2%. The Precision in each year is above the 10% mark. On the other hand, the model trained by the full set of original data shows a much higher Accuracy with a value of 89.7%, a slightly higher Precision of 15.3%, and an almost non-exist Recall of 0%. The Precision value is also very unstable from year to year, ranging from 0% to over 30%. The result means the model predicts only a few 1 labels, and even fewer are labeled correctly. This leads to the large fluctuation of the Precision value.

Table 14 Model performance using the full set of undersampled data

	2000	2001	2002	2003	2004	2005	2006	2007	2008
Accuracy	0.742	0.828	0.786	0.813	0.728	0.657	0.63	0.57	0.663
Precision	0.157	0.154	0.15	0.125	0.119	0.117	0.116	0.138	0.196
Recall	0.296	0.199	0.245	0.212	0.323	0.394	0.403	0.521	0.356
	2011	**2012**	**2013**	**2014**	**2015**	**2016**	**2017**	**2018**	**Avg**
Accuracy	0.686	0.758	0.607	0.545	0.569	0.643	0.586	0.552	0.668
Precision	0.148	0.098	0.11	0.116	0.13	0.128	0.101	0.132	0.131
Recall	0.361	0.275	0.452	0.536	0.495	0.414	0.472	0.539	0.382

Table 15 Model performance using the full set of original data

	2000	2001	2002	2003	2004	2005	2006	2007	2008
Accuracy	0.888	0.909	0.9	0.917	0.911	0.904	0.899	0.885	0.84
Precision	0.111	0.171	0.054	0.167	0.2	0	0.133	0	0.25
Recall	0	0	0	0	0	0	0	0	0
	2011	**2012**	**2013**	**2014**	**2015**	**2016**	**2017**	**2018**	**Avg**
Accuracy	0.884	0.925	0.907	0.9	0.888	0.895	0.912	0.888	0.897
Precision	0	0	0.143	0.205	0.5	0.308	0.167	0.185	0.153
Recall	0	0	0	0	0	0	0	0	0

The models trained by each individual year show a similar phenomenon. With undersampled data, the Precision slightly increases to 14.8%. The Accuracy increases to 88.4%, while Recall reduces to 2.5%. With original data, the Precision decreases to 12.5%, while the other two metrics stay almost the same with less than 1% change. Precision again fluctuates from 7.6% to 19.9%, which is a smaller range than the model trained by the full set of data but still much larger than the one trained with individual undersampled data.

Table 16 Model performance using the individual year of undersampled data

	2000	2001	2002	2003	2004	2005	2006	2007	2008	2009	2010
Accuracy	0.879	0.906	0.894	0.899	0.91	0.901	0.896	0.88	0.84	0.898	0.914
Precision	0.194	0.129	0.14	0.119	0.151	0.122	0.142	0.179	0.1	0.154	0.114
Recall	0.023	0.007	0.012	0.036	0.004	0.005	0.005	0.011	0	0.049	0.013
	2011	**2012**	**2013**	**2014**	**2015**	**2016**	**2017**	**2018**	**2019**	**2020**	**Avg**
Accuracy	0.853	0.874	0.897	0.893	0.878	0.858	0.903	0.884	0.894	0.821	0.884
Precision	0.155	0.125	0.139	0.15	0.176	0.139	0.148	0.139	0.155	0.242	0.148
Recall	0.06	0.114	0.021	0.014	0.023	0.068	0.023	0.008	0.006	0.03	0.025

Table 17 Model performance using the individual year of original data

	2000	2001	2002	2003	2004	2005	2006	2007	2008	2009	2010
Accuracy	0.883	0.907	0.896	0.914	0.905	0.9 0	0.896	0.882	0.839	0.913	0.919
Precision	0.155	0.13	0.117	0.089	0.076	.084	0.098	0.147	0.199	0.087	0.126
Recall	0.008	0.004	0.006	0.004	0.006	0.004	0.004	0.005	0.002	0.004	0.004
	2011	**2012**	**2013**	**2014**	**2015**	**2016**	**2017**	**2018**	**2019**	**2020**	**Avg**
Accuracy	0.878	0.917	0.905	0.894	0.884	0.894	0.906	0.886	0.897	0.794	0.89
Precision	0.126	0.114	0.121	0.098	0.138	0.124	0.114	0.131	0.161	0.199	0.125
Recall	0.008	0.017	0.003	0.007	0.006	0.001	0.011	0.003	0.001	0.074	0.009

It's hard to interpret the results from the Neural Network and choose the better one for further examination. The model trained by the full set of undersampled data predicts a good amount of positive labels, which may lead to significant loss if I choose this model to help my investment. Therefore I decided the one trained by individual undersampled data provides slightly higher Precision is a better solution.

Conclusion

After summarizing all the training and testing results, I can conclude that none of the algorithms are particularly good at predicting extreme stock returns. However, it's as expected. Stock movements are usually affected by more than the stock indicators included in this study, news about the company, government policy, macro-economy status, and market hype all contribute to the stock movement. But these indicators do affect the stock price one way or another, so the models trained by these data can somewhat help identify the extreme stock returns. Random Forest performs the best among the four algorithms we explore in this work. It improves all three metrics – Accuracy, Precision, and Recall by a non-negligible amount if trained by the undersampled data. If I need as many positive labels predicted correctly as possible, this is my choice. If I want fewer false positives, so I don't invest in the wrong stock, Random Forest trained by the original data also provides the highest Precision.

For future work, there are much more fine-tuning can be done to find the optimal since the algorithms used in this study are primarily in their basic form. For example, the model training and prediction are based on random sampling while stock market movement is a time series based data. Time series forecasting using Concurrent Neural Network or Recurrent Neural Network could be a good start point to further improve the performance (Bryan Lim and Stefan Zohren, 2020). Triangulation can also be used to help identify the stock with the extreme returns. If multiple models here predict the same stock will have an extreme return or models trained by other data generate the same prediction, it increases my chance of having the right stock to invest in.

References:

Bali, T. G., N. Cakici, R. F. Whitelaw. Maxing out: Stocks as lotteries and the cross-section of expected returns, Journal of Financial Economics, Volume 99, Issue 2, 2011, p427-446.

Clark, Robert A., John J. McConnell, and Manoj Singh "Seasonalities in NYSE Bid-Ask Spreads and Stock Returns in January." The Journal of Finance, vol. 47, no. 5, [American Finance Association, Wiley], 1992, p1999–2014, https://doi.org/10.2307/2329007.

Conrad, J., N. Kapadia, and Y. Xing, Death and jackpot: Why do individual investors hold overpriced stocks?, Journal of Financial Economics, Volume 113, Issue 3, 2014, p455-475.

Fama, E.F. and French, K.R.. A five-factor asset pricing model. The Journal of Finance, 47: p427-465, 1992. https://doi.org/10.1111/j.1540-6261.1992.tb04398.x

Fama, E.F. and French, K.R.. The Cross-Section of Expected Stock Returns. The Journal of Financial Economics, 116(1), p1-22, 2015. https://doi.org/10.1016/j.jfineco.2014.10.010

Kewei Hou, Chen Xue, Lu Zhang, Digesting Anomalies: An Investment Approach, The Review of Financial Studies, Volume 28, Issue 3, March 2015, p650–705, https://doi.org/10.1093/rfs/hhu068

Lee, B. and Oliver M. Rui. The dynamic relationship between stock returns and trading volume: Domestic and cross-country evidence, Journal of Banking & Finance, Volume 26, Issue 1, 2002, p51-78.

Welch, Ivo. "Capital Structure and Stock Returns." Journal of Political Economy, vol. 112, no. 1, The University of Chicago Press, 2004, p106–31, https://doi.org/10.1086/379933.

Bryan Lim and Stefan Zohren, Time Series Forecasting With Deep Learning: A Survey, Philosophical Transactions A, 2020, arXiv:2004.13408