UNIVERSITY OF TORONTO

AI in Finance
Microsoft Stock Price Prediction

•••

Overview

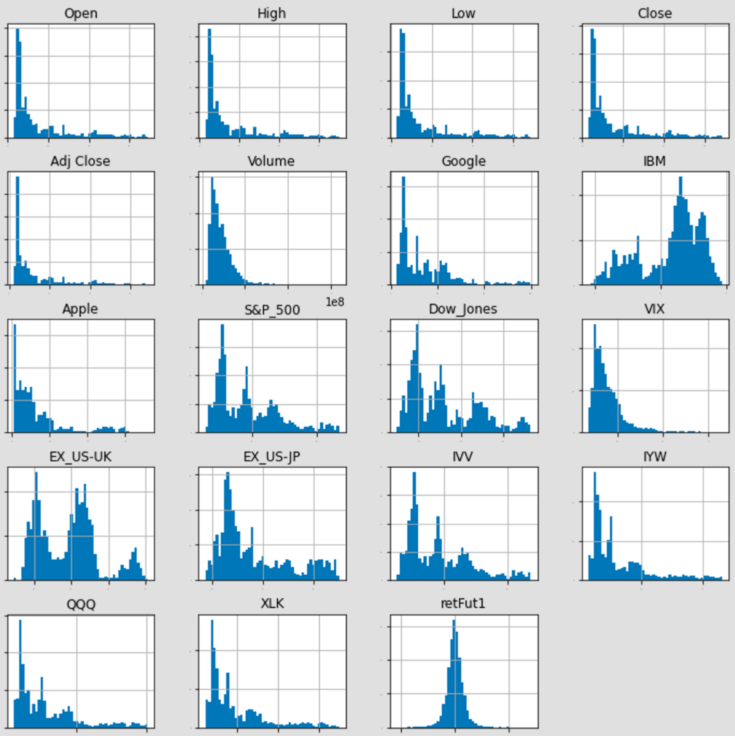
1. Performing Microsoft stock price prediction through various machine learning models and evaluation performances.
2. The project follows three sections as :
   a. Data collection & feature engineering
   b. ML model implementation
   c. Evaluating Performances
3. ML models used has been broadly classified in two section as of Regression and Classification which includes Ridge, Lasso, SVR, SVM, KNN, etc.
4. Custom feature engineering functions, loss functions and metrics functions have been used in evaluating performances and other state of art techniques in predicting stocks of the major companies.

Data Collection & Pre-Processing (Feature Engineering):

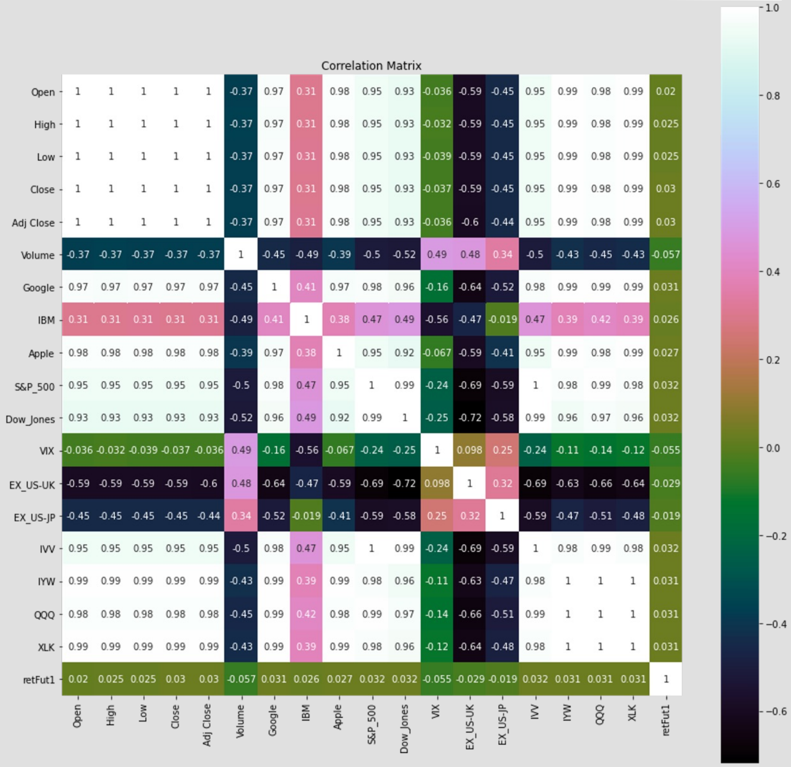
Data Collection

1. All the data has been fetched through yahoo finance (https://ca.finance.yahoo.com/)
2. Data can be classified into 4 categories:
    a. Stocks: MSFT, GOOGL, AAPL, IBM
    b. Indexes : SPY, DIA, VIX
    c. Currency : FXY, FXB
    d. ETF : QQQ, XLK, IYW, IVV
3. The frequency of the data used is daily.
4. The data being used is starting from 1st March, 2007 to 16th December, 2021.
5. The Adjusted Close value is being considered as the features/columns. Additionally for Microsoft stock, OCHLV as features are also considered.

Exploratory Data Analysis



The above histogram shows the distribution for each series individually. Further, it can be deduced as how the frequency of the different features affecting the predicting feature.



Correlation Matrix

	Open	High	Low	Close	Adj Close	Volume	Google	IBM	Apple	S&P_500	Dow_Jones	VIX	EX_US-UK	EX_US-JP	IVV	IYW	QQQ	XLK	retFut1
Open	1	1	1	1	1	-0.37	0.97	0.31	0.98	0.95	0.93	-0.036	-0.59	-0.45	0.95	0.99	0.98	0.99	0.02
High	1	1	1	1	1	-0.37	0.97	0.31	0.98	0.95	0.93	-0.032	-0.59	-0.45	0.95	0.99	0.98	0.99	0.025
Low	1	1	1	1	1	-0.37	0.97	0.31	0.98	0.95	0.93	-0.039	-0.59	-0.45	0.95	0.99	0.98	0.99	0.025
Close	1	1	1	1	1	-0.37	0.97	0.31	0.98	0.95	0.93	-0.037	-0.59	-0.45	0.95	0.99	0.98	0.99	0.03
Adj Close	1	1	1	1	1	-0.37	0.97	0.31	0.98	0.95	0.93	-0.036	-0.6	-0.44	0.95	0.99	0.98	0.99	0.03
Volume	-0.37	-0.37	-0.37	-0.37	-0.37	1	-0.45	-0.49	-0.39	-0.5	-0.52	0.49	0.48	0.34	-0.5	-0.43	-0.45	-0.43	-0.057
Google	0.97	0.97	0.97	0.97	0.97	-0.45	1	0.41	0.97	0.98	0.96	-0.16	-0.64	-0.52	0.98	0.99	0.99	0.99	0.031
IBM	0.31	0.31	0.31	0.31	0.31	-0.49	0.41	1	0.38	0.47	0.49	-0.56	-0.47	-0.019	0.47	0.39	0.42	0.39	0.026
Apple	0.98	0.98	0.98	0.98	0.98	-0.39	0.97	0.38	1	0.95	0.92	-0.067	-0.59	-0.41	0.95	0.99	0.98	0.99	0.027
S&P_500	0.95	0.95	0.95	0.95	0.95	-0.5	0.98	0.47	0.95	1	0.99	-0.24	-0.69	-0.59	1	0.98	0.99	0.98	0.032
Dow_Jones	0.93	0.93	0.93	0.93	0.93	-0.52	0.96	0.49	0.92	0.99	1	-0.25	-0.72	-0.58	0.99	0.96	0.97	0.96	0.032
VIX	-0.036	-0.032	-0.039	-0.037	-0.036	0.49	-0.16	-0.56	-0.067	-0.24	-0.25	1	0.098	0.25	-0.24	-0.11	-0.14	-0.12	-0.055
EX_US-UK	-0.59	-0.59	-0.59	-0.59	-0.6	0.48	-0.64	-0.47	-0.59	-0.69	-0.72	0.098	1	0.32	-0.69	-0.63	-0.66	-0.64	-0.029
EX_US-JP	-0.45	-0.45	-0.45	-0.45	-0.44	0.34	-0.52	-0.019	-0.41	-0.59	-0.58	0.25	0.32	1	-0.59	-0.47	-0.51	-0.48	-0.019
IVV	0.95	0.95	0.95	0.95	0.95	-0.5	0.98	0.47	0.95	1	0.99	-0.24	-0.69	-0.59	1	0.98	0.99	0.98	0.032
IYW	0.99	0.99	0.99	0.99	0.99	-0.43	0.99	0.39	0.99	0.98	0.96	-0.11	-0.63	-0.47	0.98	1	1	1	0.031
QQQ	0.98	0.98	0.98	0.98	0.98	-0.45	0.99	0.42	0.98	0.99	0.97	-0.14	-0.66	-0.51	0.99	1	1	1	0.031
XLK	0.99	0.99	0.99	0.99	0.99	-0.43	0.99	0.39	0.99	0.98	0.96	-0.12	-0.64	-0.48	0.98	1	1	1	0.031
retFut1	0.02	0.025	0.025	0.03	0.03	-0.057	0.031	0.026	0.027	0.032	0.032	-0.055	-0.029	-0.019	0.032	0.031	0.031	0.031	1

Looking at the correlation plot above, we see some correlation of the predicted variable with other features. S&P 500, Dow jones, and IVV are having the highest correlation score with the predicted MSFT stock value

Feature Engineering

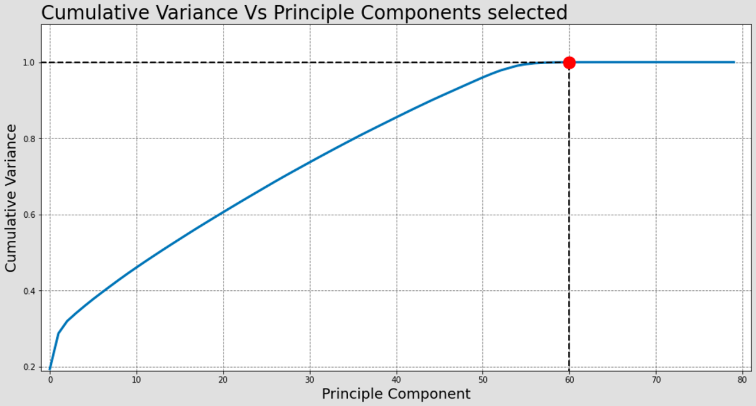
Technical indicators used to enhance the signal within the price data:

1. Adj. Close Lag : We used few different lags.
2. Momentum Features : We used the momentum features and max number of momentum lag used is 10. This feature will help us know the overall trend of the price movement and hopefully allow our model to make predictions on the basis of overall momentum inclination.
3. Exponential Moving Average : We will be using Exponential Moving Average of `10`, `21`, `65` and `100` as these are some of the most important Exponential Moving Average as per some Stock Market Gurus such as Julian Komar.
4. RSI : This is one of the most used technical indicators used by many professionals as it provide excellent price divergence indications. We used **time frame of 14 days** which is mostly used by professionals.
5. ADX : This is a great indicator which will help us quantify the strength of the trend our stock is experiencing.
6. Parabolic SAR : This indicator helps technical traders determine trend direction and potential trend reversal.

Feature Selection

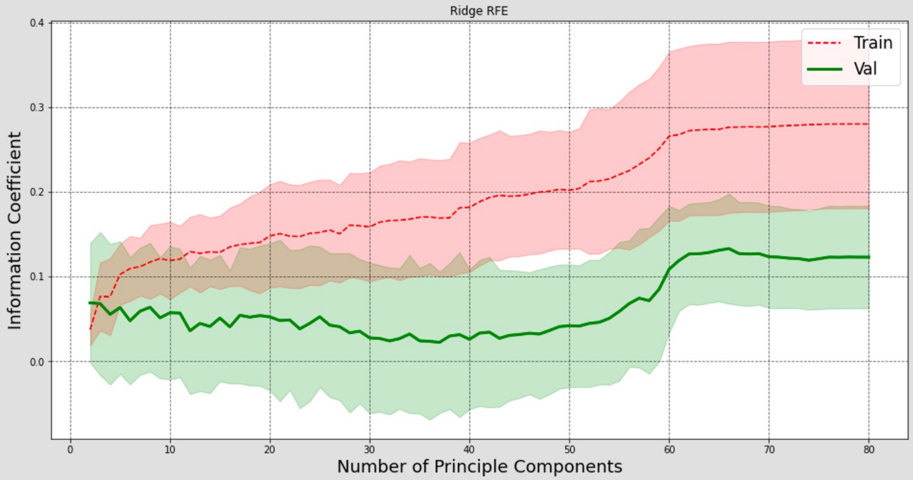
For Feature selection we have used Both Recursive Feature Elimination and PCA, They have been applied in the following manner and order:

1. Step 1 - PCA : First of all PCA was applied. The reason to do so was to drop the extra dimensions using PCA. This step will strengthen the signal and allow for complex non-linear models to fit the data without overfitting it. We have not tried to reduce the dimensionality of the dataset drastically using PCA as it would cause reduction in useful variance in the data other than noise in last few principle components.
2. Step 2 -Recursive Feature Elimination : Second, to further reduce the dimensionality of the dataset and select features which helped strengthen the signal in the data, hence improving the performance while reducing the dimensionality by dropping less relevant feature engineered features, we used Recursive Feature Elimination technique. Through the learning curve plotted it revealed that we actually observed increase in validation information coefficient by dropping less features deemed less important by Ridge regression, while also reducing overfitting!!



Plot Analysis: From the above plot it is quite clear that we atleast had 25 out of 80 dimensions in our dataset which was nothing but noise and all the relevant variance in the dataset can be compressed to just 55 dimensions, Hence using the above technique we have dropped the noisy dimensions. This is will allow our Tree ensemble models to fit the data better and avoiding the scenario where they become prone to fitting on noise in the data.

Feature Selection

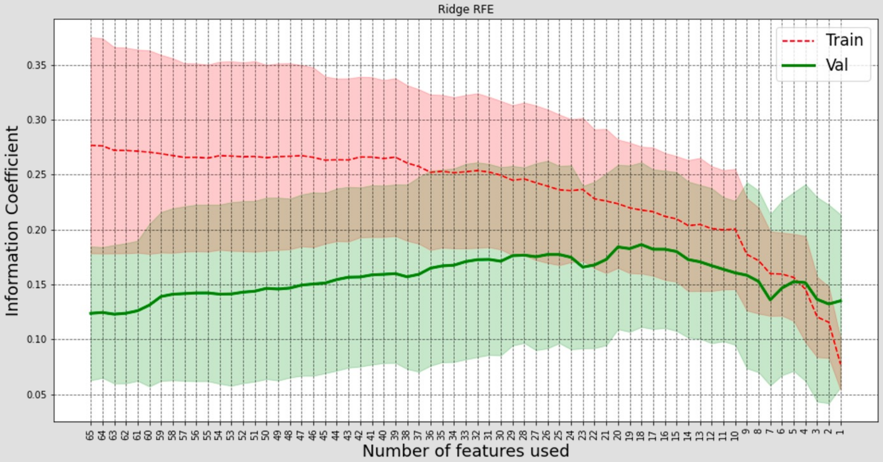


Plot Analysis: From the above plot we can clearly see that when we start to remove the principle components we observe and increase in validation score from 80 to 63 Principle components, After that the validation accuracy starts to reduce as we start to drop relevant signal in the dataset. Hence we will compress the dataset down to around 65 dimensions!!

Other custom feature selection functions are used in the project as mentioned below:

1. feature_drop_scorer function : This function returns the mean_scores_train, std_scores_train, mean_scores_val and std_scores_val by dropping the unwanted features which can cause the overfitting issues. The process follows as:
   a. Feature Engineering Function is passed through the pipeline to fit into the training & test data
   b. Ridge regression estimator is used to return the above mentioned values from feature_drop_scorer function
2. get_features function : This function returns valid feature number as per combination searched.

Feature Selection



We can observe from the plot that as we start to remove the features starting from the least important to more important ones, we observe:

1. Constant reduction in Overfitting
2. Steady increase in validation score until we are left the 20 most important features.

This shows that out of all the principle components, there exist few which provide most information without providing extra information which might confuse the model into creating a sub optimal regression line!!

Finally rather than selecting 20 best features as per above plot, we will be selecting 30 features, This is because the features selected above are on the basis of importance by ridge regression hence the best features provide most information linearly which can be harnessed by ridge regression efficiently, but non linear model such as tree ensembles might be able to find non linear relation which linear regression could not find, hence we will take few extra top features for tree ensembles to work with.

ML Model Implementation:

Approach

- Compare individual supervised machine learning algorithms (regression vs. classification) for stock price prediction
- Grid search for hyperparameter tuning with performance metrics using Time Series Split Cross-Validation for each of the regression and classification algorithms
- Compare model performance and select the best ones for model stacking

List of algorithms for evaluation

Regression

- Linear Regression (Lasso & Ridge)
- LinearSVR
- NuSVR
- KNN
- Decision Tree
- Random Forest
- Extra Trees
- Gradient Boosting
- XGBoost

Classification

- Logistic Regression
- LinearSVC
- NuSVC
- KNN
- Decision Tree
- Random Forest
- Extra Trees
- Gradient Boosting
- XGBoost

Code example for individual model tuning and grid search

Example 1 - Ridge Regression

```
ridge = Ridge()
param_grid_ridge = {'alpha':np.linspace(0.05, 10, 100)}
grid_ridge = GridSearchCV(ridge, param_grid_ridge,
cv=TimeSeriesSplit(5), scoring=myscorer, n_jobs=-1,
verbose=1)
grid_ridge.fit(X_train_fe_reduced_s, y_train)
```

Note: the Scoring method implemented for regression algorithms is customized as Spearman's Correlation:

```
def information_coefficient(y_true, y_pred):
    rho, pval = spearmanr(y_true,y_pred)
    return rho
myscorer = make_scorer(information_coefficient,
greater_is_better=True)
```

Example 2 - Logistic Regression

```
logreg = LogisticRegression()

param_grid_logreg = {'C':np.linspace(0.01, 1,
100)}

grid_logreg = GridSearchCV(logreg,
param_grid_logreg, cv=TimeSeriesSplit(5),
scoring='accuracy', n_jobs=1, verbose=1)

grid_logreg.fit(X_train_fe_reduced_s, yc_train)
```

Summary of GridSearchCV results for regression models

Table 1 - Tuned regression model parameters

Model	Parameter Search Grid	Optimal Model Parameters
Ridge()	{'alpha':np.linspace(0.001, 10, 100)}	{'alpha': 0.001}
Lasso()	{'alpha':np.logspace(-7, -2, 100)}	{'alpha': 1.8307382802953698e-06}
LinearSVR()	{'C':np.linspace(0.0005, 1, 100), 'epsilon':[0.0]}	{'C': 0.10145959595959596, 'epsilon': 0.0}
NuSVR()	{'nu':[0.0, 0.1, 0.2, 0.3, 0.4, 0.5], 'C':np.linspace(0.005, 0.08, 50), 'kernel':['linear']}	{'C': 0.06775510204081633, 'kernel': 'linear', 'nu': 0.3}
KNeighborsRegressor()	{'n_neighbors':[i for i in range(1, 100)]}	{'n_neighbors': 88}
DecisionTreeRegressor()	{'max_depth':[i for i in range(3, 20)] + [None], 'min_samples_split':[i for i in range(2, 10)]}	{'max_depth': 4, 'min_samples_split': 3}

Summary of GridSearchCV results for regression models (Cont.)

Model	Parameter Search Grid	Optimal Model Parameters
RandomForestRegressor()	{'n_estimators':[700],'min_samples_split':[5, 7, 9, 13], 'max_features':[0.5, 0.8, 1.0],'max_depth':[5,7,9]}	{'max_depth': 5, 'max_features': 0.5, 'min_samples_split': 9, 'n_estimators': 700}
ExtraTreesRegressor()	{'n_estimators':[800],'min_samples_split':[5, 7, 9, 13], 'max_features':[0.5, 0.8, 1.0],'max_depth':[5, 7, 9]}	{'max_depth': 5, 'max_features': 0.5, 'min_samples_split': 5, 'n_estimators': 800}
GradientBoostingRegressor()	{'n_estimators':[100, 200, 400, 600, 1000], 'max_depth':[3], 'max_features':[0.3, 0.5, 0.7], 'min_samples_split':[5, 7, 9, 13]}	{'n_estimators': 100, 'min_samples_split': 13, 'max_features': 0.5, 'max_depth': 3}
XGBRegressor()	{'learning_rate':[0.05, 0.1],'n_estimators':[500, 1000, 1100, 1200],'max_depth':[3, 4, 5], 'max_features':[0.3, 0.5, 0.7,0.9],'min_samples_split':[5, 7, 9, 13] }	{'n_estimators': 1200, 'min_samples_split': 9, 'max_features': 0.7, 'max_depth': 5, 'learning_rate': 0.05}

Summary of GridSearchCV results for classification models

Table 2 - Tuned classification model parameters

Model	Parameter Search Grid	Optimal Model Parameters
LogisticRegression()	{'logreg__C':np.linspace(0.01, 1, 100)}	{'logreg__C': 0.14}
LinearSVC()	{'svc__C':np.linspace(0.05, 1, 100)}	{'svc__C': 0.76010101010101}
NuSVC()	{'nu':linspace(0.01, 1.0), 'kernel':['linear']}	{'kernel': 'linear', 'nu': 0.9191836734693878}
KNeighborsClassifier()	{'n_neighbors':[i for i in range(1, 100)]}	{'n_neighbors': 17}
DecisionTreeClassifier()	{'max_depth':[i for i in range(3, 20)] + [None], 'min_samples_split':[i for i in range(2, 10)]}	{'max_depth': 3, 'min_samples_split': 2}

Summary of GridSearchCV results for classification models (Cont.)

Model	Parameter Search Grid	Optimal Model Parameters
RandomForestClassifier()	{'n_estimators':[700],'min_samples_split':[5, 7, 9, 13], 'max_features':[0.5, 0.8, 1.0],'max_depth':[5,6,7]}	{'max_depth': 6, 'max_features': 0.5, 'min_samples_split': 5, 'n_estimators': 700}
ExtraTreesClassifier()	{'n_estimators':[800],'min_samples_split':[5, 7, 9, 13], 'max_features':[0.7, 0.8, 1.0],'max_depth':[5, 6, 7]}	{'max_depth': 6, 'max_features': 0.8, 'min_samples_split': 7, 'n_estimators': 800}
GradientBoostingClassifier()	{'n_estimators':[100, 200, 400, 600, 1000], 'max_depth':[3], 'max_features':[0.3, 0.5, 0.7], 'min_samples_split':[5, 7, 9, 13]}	{'n_estimators': 100, 'min_samples_split': 13, 'max_features': 0.3, 'max_depth': 3}
XGBClassifier()	{'learning_rate':[0.05, 0.1], 'n_estimators':[500, 1000, 1100, 1200], 'max_depth':[3, 4, 5, 6, 7], 'max_features':[0.3, 0.5, 0.7],'min_samples_split':[5, 7, 9, 13]}	{'n_estimators': 1100, 'min_samples_split': 13, 'max_features': 0.7, 'max_depth': 7, 'learning_rate': 0.1}

Stacking Ensemble

Why stacking?

- Predictions from baseline models are used as training dataset for the final aggregating model and the model is trained to generalize the result by focusing on ML models that are best at predicting a particular sample more. Summary of stacked models for regression and classification approaches are provided below. Model performance in terms of CAGR and Sharpe ratio are included in the previous summary tables for the stacking ensembles.

Stacked Regression Model

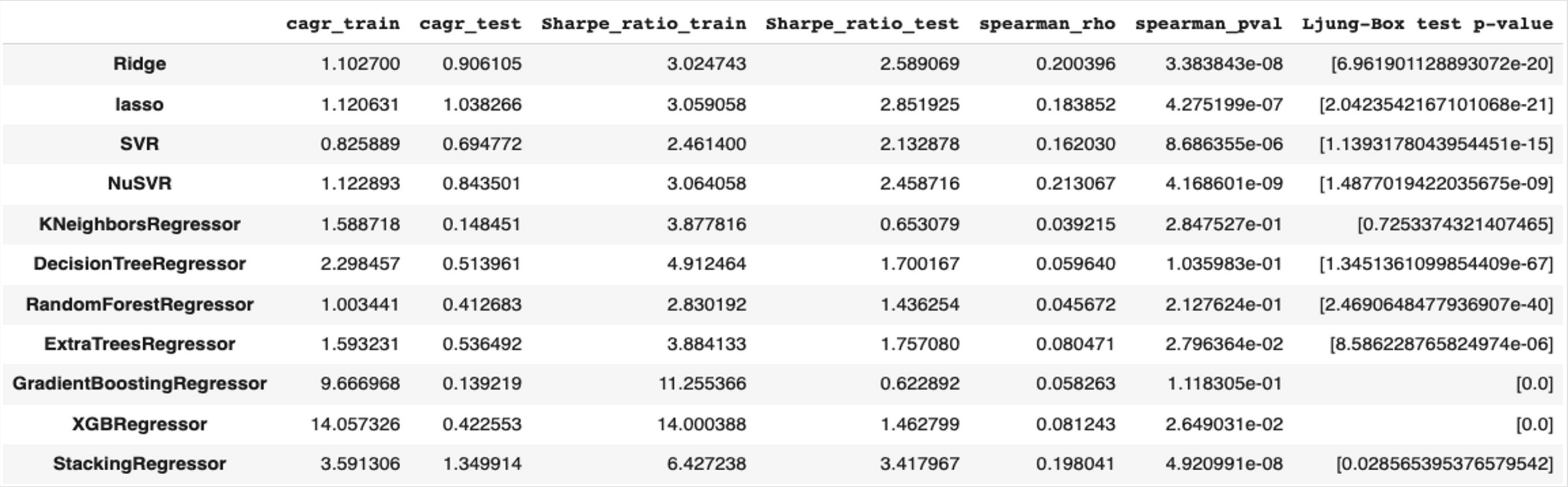
- Selected Estimators for stacking:
  `Ridge(),Lasso(),LinearSVR(),NuSVR(),DecisionTreeRegressor(),KNeighborsRegressor()`
- Spearman’s rho::
  Training: 0.229 | Testing: 0.172

Stacked Classification Model

- Selected Estimators for stacking:
  `LogisticRegression(),LinearSVC(),NuSVC()`
- Accuracy Score::
  Training: 0.565 | Testing: 0.576

Regression Model Performance:

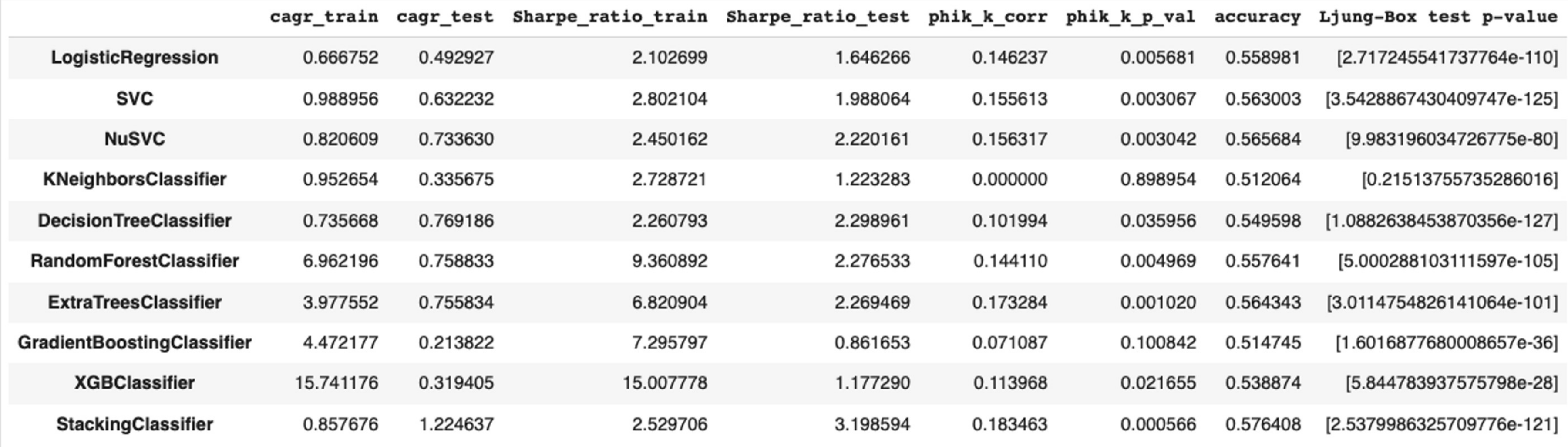
Summary of CAGR, Sharpe Ratio, Spearman's Correlation and White Reality Check P-value for tuned regression models are summarized in the table below:



	cagr_train	cagr_test	Sharpe_ratio_train	Sharpe_ratio_test	spearman_rho	spearman_pval	Ljung-Box test p-value
Ridge	1.102700	0.906105	3.024743	2.589069	0.200396	3.383843e-08	[6.961901128893072e-20]
lasso	1.120631	1.038266	3.059058	2.851925	0.183852	4.275199e-07	[2.0423542167101068e-21]
SVR	0.825889	0.694772	2.461400	2.132878	0.162030	8.686355e-06	[1.1393178043954451e-15]
NuSVR	1.122893	0.843501	3.064058	2.458716	0.213067	4.168601e-09	[1.4877019422035675e-09]
KNeighborsRegressor	1.588718	0.148451	3.877816	0.653079	0.039215	2.847527e-01	[0.7253374321407465]
DecisionTreeRegressor	2.298457	0.513961	4.912464	1.700167	0.059640	1.035983e-01	[1.3451361099854409e-67]
RandomForestRegressor	1.003441	0.412683	2.830192	1.436254	0.045672	2.127624e-01	[2.4690648477936907e-40]
ExtraTreesRegressor	1.593231	0.536492	3.884133	1.757080	0.080471	2.796364e-02	[8.586228765824974e-06]
GradientBoostingRegressor	9.666968	0.139219	11.255366	0.622892	0.058263	1.118305e-01	[0.0]
XGBRegressor	14.057326	0.422553	14.000388	1.462799	0.081243	2.649031e-02	[0.0]
StackingRegressor	3.591306	1.349914	6.427238	3.417967	0.198041	4.920991e-08	[0.028565395376579542]

Classification Model Performance:

Summary of CAGR, Sharpe Ratio, Spearman's Correlation and White Reality Check P-value for tuned classification models are summarized in the table below:



	cagr_train	cagr_test	Sharpe_ratio_train	Sharpe_ratio_test	phik_k_corr	phik_k_p_val	accuracy	Ljung-Box test p-value
LogisticRegression	0.666752	0.492927	2.102699	1.646266	0.146237	0.005681	0.558981	[2.717245541737764e-110]
SVC	0.988956	0.632232	2.802104	1.988064	0.155613	0.003067	0.563003	[3.5428867430409747e-125]
NuSVC	0.820609	0.733630	2.450162	2.220161	0.156317	0.003042	0.565684	[9.983196034726775e-80]
KNeighborsClassifier	0.952654	0.335675	2.728721	1.223283	0.000000	0.898954	0.512064	[0.21513755735286016]
DecisionTreeClassifier	0.735668	0.769186	2.260793	2.298961	0.101994	0.035956	0.549598	[1.0882638453870356e-127]
RandomForestClassifier	6.962196	0.758833	9.360892	2.276533	0.144110	0.004969	0.557641	[5.000288103111597e-105]
ExtraTreesClassifier	3.977552	0.755834	6.820904	2.269469	0.173284	0.001020	0.564343	[3.0114754826141064e-101]
GradientBoostingClassifier	4.472177	0.213822	7.295797	0.861653	0.071087	0.100842	0.514745	[1.6016877680008657e-36]
XGBClassifier	15.741176	0.319405	15.007778	1.177290	0.113968	0.021655	0.538874	[5.844783937575798e-28]
StackingClassifier	0.857676	1.224637	2.529706	3.198594	0.183463	0.000566	0.576408	[2.5379986325709776e-121]

Analysis on model performance

- **Tree Ensembles:**
  It is observed that tree ensembles have a tendency to overfit which is most significant with the Gradient Boosting trees. The effect can be observed in the high cagr and sharpe ratio for training set compared to low values for testing set. The reason is that the stock price dataset has lots of similar kind of noise in the training data that was dependent on time frames and those noise might not be present in the testing data. There might be some forces in stock market which were influencing the stock price at some point of time but may not be present in time steps in future. Gradient boosting approach is prone to overfit due to the model complexity, the inherent noise in the stock price makes it more difficult to find an optimal model parameter set that can generalize to both training and testing sets.

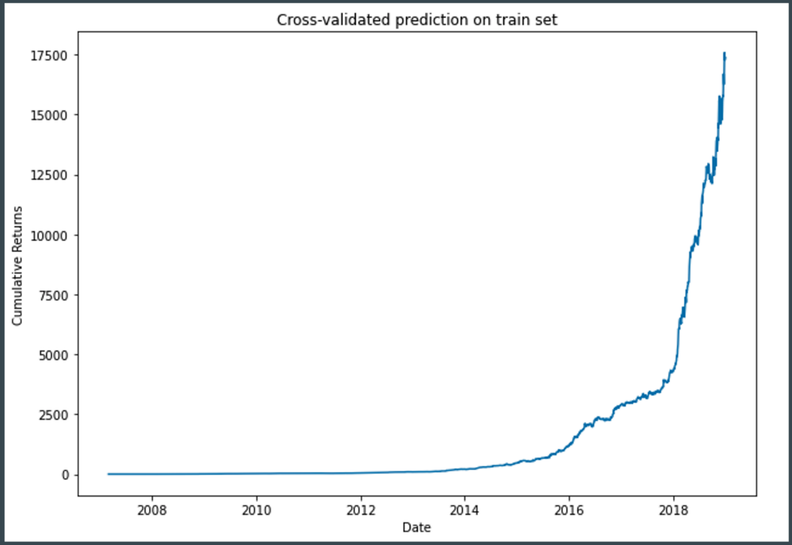
- **Linear Models :**
  Contrary to the case of tree ensembles, linear models showed less overfitting with similar cagr and sharpe ratios between training and testing sets. Further, it is observed that the sharpe ratio and cagr for lasso and ridge regression are highest as compared to the other models. This shows how well linear models are performing owing to their simplicity as compared to non-linear models.

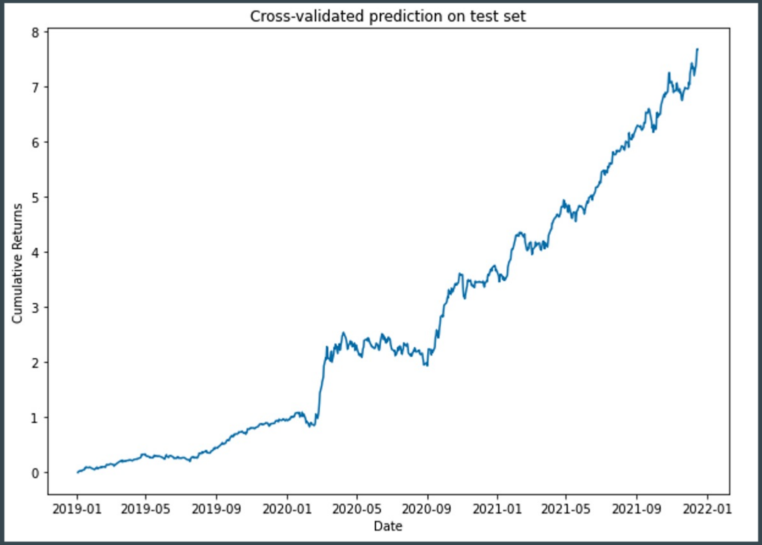
- **Stacking:**
  After stacking, the models are able to generalize even further and are able to result in higher training and testing cagr and sharpe ratios.

UNIVERSITY OF TORONTO

Final Recommended Model - Stack Regression

The stacking ensemble of regression models are recommended as the optimal model to perform stock price prediction, based on its enhanced performance at predicting the highest CAGR and Sharpe ratio with minimal overfitting and more generalization. The cross-validated prediction on train and test sets are shown as below:





UNIVERSITY OF TORONTO

Evaluate the effectiveness of the stack regression model

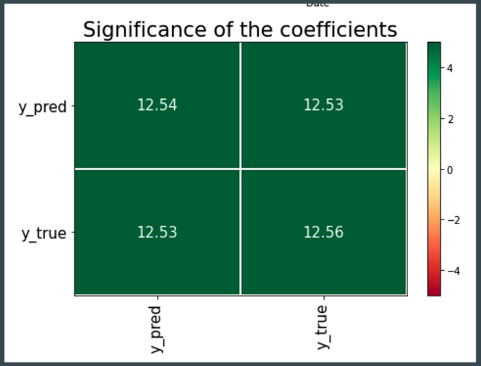


Figure 1. Significance of correlation between ground truth and predicted targets

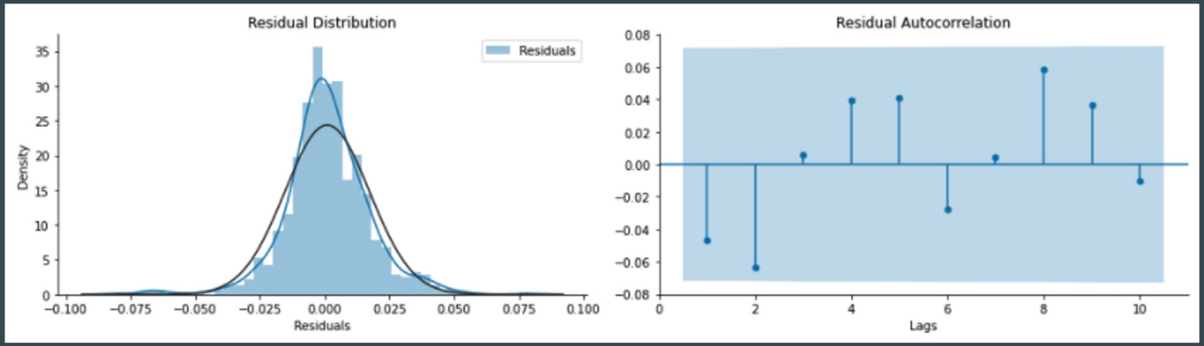


Figure 2. Residual distribution and autocorrelation

Interpretation on the plots:

Observing from the significance matrix (Figure 1), a high value of the correlation coefficient indicates that the correlation between test y_pred and y_true is statistically significant. The residual histogram and ACF plot showed that the residuals are normally distributed and the autocorrelation between residuals are not large enough to have noticeable impact on the forecasts or the prediction intervals.

White Reality Check on Stack Regression Model

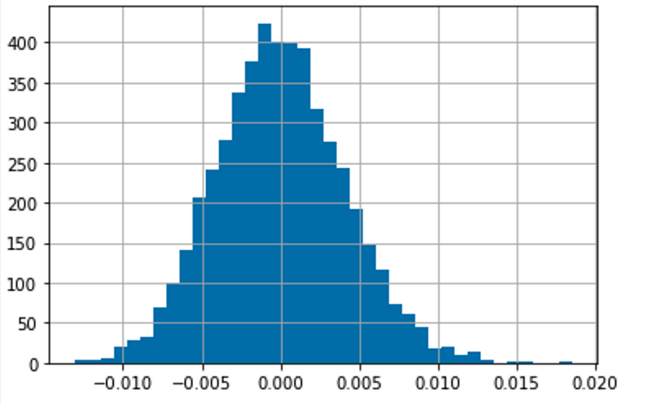


Figure 1. Bootstrapped sampling distribution

```
average return: 0.014570

97.5% confidence interval threshold for the
theoretical averages: [-0.00758637  0.00808576]

Reject Ho = The population distribution of rule
returns has an expected value of zero or less
(because p_value is small enough)

p_value:0.00019999999999997797
```

The p-value is calculated based on the count that are greater than the average return. It gives a direct estimate of the data-mining bias. The small p-value of 0.0002 is small enough to reject the null hypothesis that the prediction of the best model has no predictive superiority over a given benchmark mode, permitting account to be taken of the effects of data snooping. A small p-value also indicates that there is a strong correlation between the residuals, meaning the predicted values and true values are closely related.

Thank You!!!