MATH 5799 - Machine Learning in Finance

Independent Study Final Report

Forecasting Credit spreads using Machine learning

Chamundeswari Koppisetti
May 2nd, 2020
chamundeswari.koppisetti@uconn.edu
Masters in Applied Financial Mathematics
University of Connecticut

Acknowledgements

I want to express my sincere gratitude to my professor Edward Perry for his valuable guidance, sharing his knowledge, his monitoring throughout the period of my independent study, and by being always available for providing helpful advice**.**

1. Introduction and literature review

Credit Spreads are predictors of future economic activity. When credit spreads are thin, it signals investor confidence in the economy. In contrast, widening credit spreads show deteriorating conditions in credit markets and investor nervousness about the overall business environment. The wider the spread the more serious investor concerns are about the economy.

Credit spread is the difference in current market yields between corporate bonds and government bonds of similar maturities. Credit spread increases when you move from investment grade to speculative grade resulting in the additional payoff from taking extra risks.

As part of my independent study, I am forecasting credit spreads of investment grade corporate bonds (limited data from Bloomberg due to impact from Covid 19). Predicting credit spreads can help to hedge between corporate and government bonds (buy or sell) based on if spreads widen or tighten. Note that I am not tracking spreads on individual bonds, but the model developed can be extended.

2. Model evolution

There are different streams of models. One of the types are dynamic models which includes equilibrium models like CIR and Vasicek models to describe the dynamics of interest rates and Heston model to describe volatility of underlying assets. Also arbitrage-free models like Ho & Lee, Hull & White etc. Although a well-defined stochastic differential equation is a good starting point, typically it cannot completely capture the whole dynamics. One of the reasons is the limitation of parameters.

So, we turn to different knowledge-discovery and machine learning techniques. The main motivation behind this approach is to address the limitation handle complex non-linear relationships between variables (in addition to seasonality and the presence of structural breaks)

Will focus on some regression models that have been proved to perform well in financial markets. Our goal is to use these models for predicting the credit spread. The models considered

are linear regression, Decision trees, Random Forests, Bayesian Additive Regression (BART), Gradient Boosted trees and Long Short Term Memory (LSTM) Networks.

3. Data description

In order to predict future credit spreads, it is necessary to base our model on previous data gathered for all indicators. So in this study, the data used are time series over approximately the last one decade from multiple sources yahoo finance, QUANDL, CBOE, FRED Economic Data

3.1 Explanatory variables

3.1.1. Market variables

SP 500 index: To represent the variations of equities, we chose to consider the main index as a proxy of their variations

CBOE VIX - 30 day expected volatility, **CBOE S&P 500 6-Month VIX**: To gauge the market expectation of stock market volatility. Captures the future probability of default as a common forward-looking risk metric. Can also be considered as a proxy of liquidity in the market.

10-Year Treasury Constant Maturity minus Federal Funds Rate: As credit spreads can be affected by the dynamics of supply and demand, the Federal fund rate explains the effect of this fundamental factor of supply and demand in our data. FOMC determines the FED fund rate considering a variety of economic factors: employment, consumer spending and income, business investments etc.

3 month to 30 year Treasury constant maturity rates: 3 month to 30 year yields estimated from the average yields of a variety of Treasury securities with different maturities derived from the Treasury yield curve

TED Spread : Spread between 3 Month LIBOR based on US dollars and 3 month T-bill which indicates perceived credit risk (to quantify the sentiment, defining here as confidence in the economy)

3.1.2. Macro-economic variables

10-Year Breakeven Inflation Rate: To view the average 10-year expectation for the inflation rate among market participants. Increased real activity could spell investor confidence or, alternatively, signal inflationary pressures, both of which affect credit spreads, but differently.

As inflation increases, so do the riskless yields, and the credit spread can increase as well. However, the effects of rising inflation on the long and short credit spreads can be different.

Unemployment Rate: View data of the unemployment rate, or the number of people 16 and over actively searching for a job as a percentage of the total labour force. Consumer confidence, which is forward indicator of consumer spending can be viewed through GDP growth, unemployment rate

Industrial Production- Total index: The industrial production (IP) index measures the real output of all relevant establishments located in the United States, regardless of their ownership, but not those located in U.S. territories. IP index also measures the real activity index which is a forward indicator of GDP growth.

3.2. Target variable:

The target variable is credit spread of US investment grade corporate bonds over US treasuries, which is for e.g. Moody's Seasoned Aaa Corporate Bond Yield Relative to Yield on 10-Year Treasury Constant Maturity.

4. Feature Engineering

Since this is a time series problem, to capture the seasonality, trends and volatility in the data – different features are created. For example developed, slope and curvature (beta) factors to represent the entire term structure of treasury rates.

a. Slope is defined as the difference between 30-year and 3-month rate
b. Curvature is defined as 0.5*(3 year + 5 year) - 0.25*(3 month + 6 month + 7 year + 10 year) rates
c. Rolling Autocorrelation with lags 5 days, 20 days, 60 days are created to capture the effect in 1 week, 1 month and 3 months approximately (considered trading days). This is defined as lagged correlation to measure the relationship between credit spreads current value and its past values.

5. Forecasting models and model evaluation

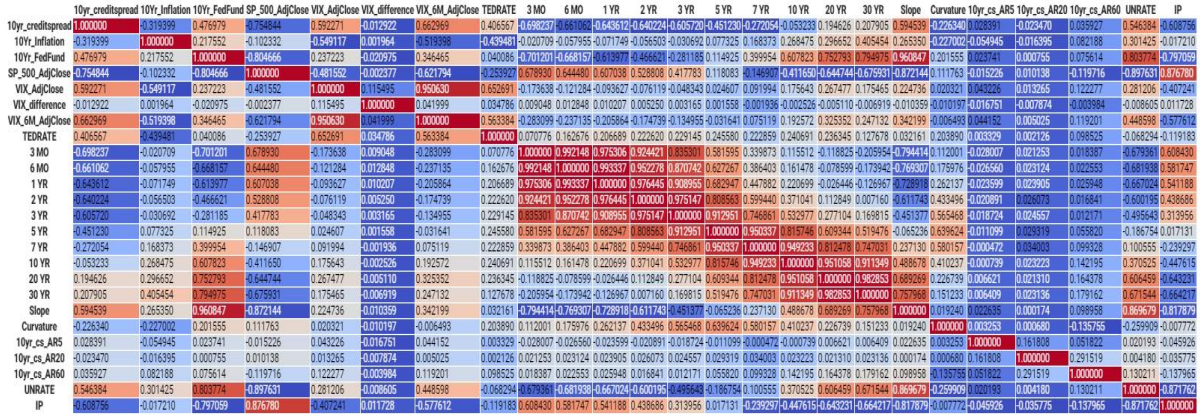
5.1. Linear regression

Linear regression methods can use different attributes in the data to make the credit spread predictions. Different attributes used in our case are S&P 500, TED Spread, CBOE VIX,

3 month to 30 year treasury rates etc. and hence this is a multiple regression model with more than one predictor variable. One of the basic assumptions about the error terms here is they are not auto correlated and have no correlation with predictor variables. Because, when independent variables are correlated, it indicates that changes in one variable are associated with shifts in another variable. The stronger the correlation, the more difficult it is to change one variable without changing another and it becomes difficult for the model to estimate the relationship between each independent variable and the dependent variable.

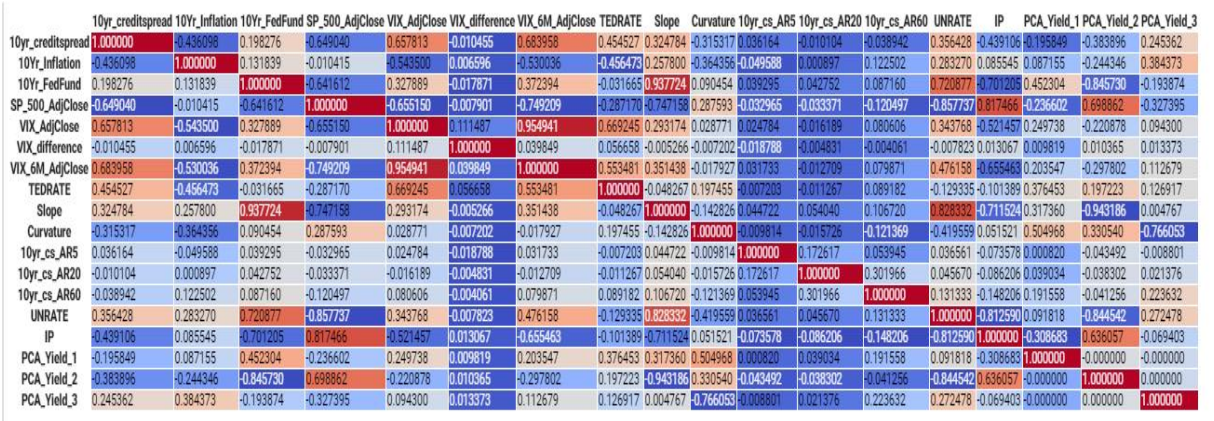
Constructed a correlation matrix for all the attributes and it is observed that there exists strong collinearity between different variables. $\rho\, \boldsymbol{x}, \boldsymbol{y} = \boldsymbol{E}[\boldsymbol{xy}] - \boldsymbol{E}[\boldsymbol{x}]\boldsymbol{E}[\boldsymbol{y}] \,/\, (\sqrt{(\boldsymbol{E}[\boldsymbol{y}^2] - \boldsymbol{E}[\boldsymbol{x}]^2)}\, \sqrt{(\boldsymbol{E}[\boldsymbol{y}^2] - \boldsymbol{E}[\boldsymbol{x}]^2)}$

The below figure shows the variables and correlation coefficients:



	10yr_creditspread	10Yr_Inflation	10Yr_FedFund	SP_500_AdjClose	VIX_AdjClose	VIX_difference	VIX_6M_AdjClose	TEDRATE	3 MO	6 MO	1 YR	2 YR	3 YR	5 YR	7 YR	10 YR	20 YR	30 YR	Slope	Curvature	10yr_cs_AR5	10yr_cs_AR20	10yr_cs_AR60	UNRATE	IP
10yr_creditspread	1.000000	-0.319399	0.476979	-0.754844	0.592271	0.012922	0.662969	0.406567	0.698237	0.661062	0.643612	0.640224	0.605720	0.451230	0.272054	0.053233	0.194626	0.207905	0.594539	0.226340	0.028391	0.023470	0.039927	0.546384	0.608756
10Yr_Inflation	-0.319399	1.000000	0.217552	-0.102332	-0.549117	0.001964	-0.519398	-0.439481	-0.020709	-0.057955	-0.071749	-0.056503	-0.030692	-0.077325	-0.168373	-0.268475	-0.296652	-0.405454	-0.265350	-0.227002	-0.054945	-0.016395	-0.082188	-0.301425	-0.017210
10Yr_FedFund	0.476979	0.217552	1.000000	-0.804666	0.237223	-0.020975	0.346465	0.040086	-0.701201	-0.668157	-0.613977	-0.466621	-0.281185	-0.114925	-0.399954	-0.607823	-0.752793	-0.794975	-0.960847	-0.201555	-0.023741	-0.000755	-0.075614	-0.803774	-0.797059
SP_500_AdjClose	-0.754844	-0.102332	-0.804666	1.000000	-0.481552	-0.002377	-0.621794	-0.253927	0.678930	0.644480	0.607038	0.528808	0.417783	0.118083	-0.146907	-0.411650	-0.644744	-0.679931	-0.872144	-0.111763	-0.015226	-0.010138	-0.119716	-0.897631	-0.876780
VIX_AdjClose	0.592271	-0.549117	0.237223	-0.481552	1.000000	0.115495	0.950630	0.652691	-0.173638	-0.121284	-0.093627	-0.076119	-0.048343	0.024607	0.091994	0.175643	0.267477	0.175465	0.224736	0.020321	0.043226	0.013265	0.122277	0.281206	-0.407241
VIX_difference	-0.012922	0.001964	-0.020975	-0.002377	0.115495	1.000000	0.041999	0.034786	0.009048	0.012848	0.010207	0.005250	0.003165	0.001558	-0.001936	-0.002526	-0.005110	-0.006919	-0.010359	-0.010197	-0.016751	-0.007874	-0.003984	-0.008605	0.011728
VIX_6M_AdjClose	0.662969	-0.519398	0.346465	-0.621794	0.950630	0.041999	1.000000	0.563384	-0.263099	-0.237135	-0.205864	-0.174739	-0.134955	-0.031641	0.075119	0.192572	0.325352	0.247132	0.342199	-0.006493	0.044152	0.005025	0.119201	0.448598	-0.577612
TEDRATE	0.406567	-0.439481	0.040086	-0.253927	0.652691	0.034786	0.563384	1.000000	0.070776	0.162676	0.206689	0.222620	0.229145	0.245580	0.222859	0.240691	0.236345	0.127678	0.032161	0.203890	0.003329	0.002126	0.098525	-0.068294	-0.119183
3 MO	-0.698237	-0.020709	-0.701201	0.678930	-0.173638	0.009048	-0.263099	0.070776	1.000000	0.992148	0.975306	0.924421	0.835301	0.581595	0.399873	0.115512	-0.118825	-0.205954	-0.794410	0.112001	-0.028007	0.021253	0.018387	-0.679361	0.608430
6 MO	-0.661062	-0.057955	-0.668157	0.644480	-0.121284	0.012848	-0.237135	0.162676	0.992148	1.000000	0.993337	0.952278	0.870742	0.627257	0.386403	0.161478	-0.078599	-0.173942	-0.769307	0.175976	-0.026560	0.023124	0.022553	-0.681938	0.581747
1 YR	-0.643612	-0.071749	-0.613977	0.607038	-0.093627	0.010207	-0.205864	0.206689	0.975306	0.993337	1.000000	0.976445	0.908955	0.682947	0.447882	0.220699	-0.026446	-0.126967	-0.728918	0.262137	-0.023599	0.023905	0.025948	-0.667024	0.541188
2 YR	-0.640224	-0.056503	-0.466621	0.528808	-0.076119	0.005250	-0.174739	0.222620	0.924421	0.952278	0.976445	1.000000	0.975147	0.808563	0.599440	0.371041	0.112849	0.007160	-0.611743	0.433496	-0.020891	0.026073	0.016841	-0.600195	0.438686
3 YR	-0.605720	-0.030692	-0.281185	0.417783	-0.048343	0.003165	-0.134955	0.229145	0.835301	0.870742	0.908955	0.975147	1.000000	0.912951	0.746861	0.532977	0.277104	0.169815	-0.451377	0.565468	-0.018724	0.024657	0.012171	-0.495643	0.313956
5 YR	-0.451230	0.077325	0.114925	0.118083	0.024607	0.001558	-0.031641	0.245580	0.581595	0.627267	0.682947	0.808563	0.912951	1.000000	0.950337	0.815746	0.609344	0.519476	-0.065236	0.639624	-0.011099	0.029319	0.055820	-0.186754	0.017131
7 YR	-0.272054	0.168373	0.399954	-0.146907	0.091994	0.001936	0.075119	0.222859	0.399873	0.386403	0.447882	0.599440	0.746861	0.950337	1.000000	0.949233	0.812478	0.747031	0.237130	0.580157	-0.000472	0.034003	0.099328	0.100555	-0.239297
10 YR	-0.053233	0.268475	0.607823	-0.411650	0.175643	0.002526	0.192572	0.240691	0.115512	0.161478	0.220699	0.371041	0.532977	0.815746	0.949233	1.000000	0.951058	0.911349	0.488676	0.410237	-0.000739	0.023223	0.142195	0.370525	-0.447615
20 YR	0.194626	0.296652	0.752793	-0.644480	0.267477	0.005110	0.325352	0.236345	-0.118825	-0.078599	-0.026446	0.112849	0.277104	0.609344	0.812473	0.951058	1.000000	0.982853	0.689269	0.226739	0.006621	0.021310	0.164378	0.606459	-0.643231
30 YR	0.207905	0.405454	0.794975	-0.675931	0.175465	0.006919	0.247132	0.127678	-0.205954	-0.173942	-0.126967	0.007160	0.169815	0.519476	0.747031	0.911349	0.982853	1.000000	0.757968	0.151233	0.006409	0.023136	0.179162	0.671544	-0.664217
Slope	0.594539	0.265350	0.960847	-0.872144	0.224736	0.010359	0.342199	0.032161	-0.794414	-0.769307	-0.728918	-0.611743	-0.451377	-0.065236	0.237130	0.488678	0.689269	0.757968	1.000000	0.019240	0.022635	0.000174	0.098958	0.869679	-0.817879
Curvature	-0.226340	-0.227002	0.201555	0.111763	0.020321	0.010197	0.006493	0.203890	0.112001	0.175976	0.262137	0.433496	0.565468	0.639624	0.580157	0.410237	0.226739	0.151233	0.019240	1.000000	0.003253	0.000680	-0.135755	-0.259909	-0.007772
10yr_cs_AR5	0.028391	-0.054945	0.023741	-0.015226	0.043226	0.016751	0.044152	0.003329	-0.028007	-0.026560	-0.023599	-0.020891	-0.018724	-0.011099	-0.000472	-0.000739	0.006621	0.006409	0.022635	0.003253	1.000000	0.161808	0.051822	0.020193	-0.045926
10yr_cs_AR20	-0.023470	-0.016395	0.000755	0.010138	0.013265	0.007874	0.005025	0.002126	0.021253	0.023124	0.023905	0.026073	0.024657	0.029319	0.034003	0.023223	0.021310	0.023136	0.000174	0.000680	0.161808	1.000000	0.291519	0.004180	-0.035775
10yr_cs_AR60	0.039927	0.082188	0.075614	-0.119716	0.122277	0.003984	0.119201	0.098525	0.018387	0.022553	0.025948	0.016841	0.012171	0.055820	0.099328	0.142195	0.164378	0.179162	0.098958	0.135755	0.051822	0.291519	1.000000	0.130211	-0.137965
UNRATE	0.546384	0.301425	0.803774	-0.897631	0.281206	0.008605	0.448598	0.068294	0.679395	0.681938	0.667024	0.600195	0.495643	0.186754	0.100555	0.370525	0.606459	0.671544	0.869679	0.259909	0.020193	0.004180	0.130211	1.000000	0.871762
IP	0.608756	-0.017210	-0.797059	0.876780	-0.407241	0.011728	-0.577612	0.119183	0.608430	0.581747	0.541188	0.438686	0.313956	0.017131	0.239297	0.447615	0.643231	0.664217	0.817879	0.007772	0.045926	0.035775	0.137965	0.871762	1.000000

To remove the collinearity in data and reduce overfitting from more noisy variables, PCA technique is applied for 3 month to 30 year treasury rates and the correlation matrix is found to be stable after applying PCA.



	10yr_creditspread	10Yr_Inflation	10Yr_FedFund	SP_500_AdjClose	VIX_AdjClose	VIX_difference	VIX_6M_AdjClose	TEDRATE	Slope	Curvature	10yr_cs_AR5	10yr_cs_AR20	10yr_cs_AR60	UNRATE	IP	PCA_Yield_1	PCA_Yield_2	PCA_Yield_3
10yr_creditspread	1.000000	-0.436098	0.198276	-0.649040	0.657813	-0.010455	0.683958	0.454527	0.324784	-0.315317	0.036164	-0.010104	-0.038942	0.356428	-0.439106	-0.195849	-0.383896	0.245362
10Yr_Inflation	-0.436098	1.000000	0.131839	-0.010415	-0.543500	0.006596	-0.530036	-0.456473	0.257800	-0.364356	-0.049588	0.000897	0.122502	0.283270	0.085545	0.087155	-0.244346	0.384373
10Yr_FedFund	0.198276	0.131839	1.000000	-0.641612	0.327889	-0.017871	0.372394	-0.031665	0.937724	0.090454	0.039295	0.042752	0.087160	0.720877	-0.701205	0.452304	-0.845730	-0.193874
SP_500_AdjClose	-0.649040	-0.010415	-0.641612	1.000000	-0.655150	-0.007901	-0.749209	-0.287170	-0.747158	0.287593	-0.032965	-0.033371	-0.120497	-0.857737	0.817466	-0.236602	0.698862	-0.327395
VIX_AdjClose	0.657813	-0.543500	0.327889	-0.655150	1.000000	0.111487	0.954941	0.669245	0.293174	0.028771	0.024784	-0.016189	0.080606	0.343768	-0.521457	0.249738	-0.220878	0.094300
VIX_difference	-0.010455	0.006596	-0.017871	-0.007901	0.111487	1.000000	0.039849	0.056658	-0.005266	-0.007202	-0.018788	-0.004831	-0.004061	-0.007823	0.013067	0.009819	0.010365	0.013373
VIX_6M_AdjClose	0.683958	-0.530036	0.372394	-0.749209	0.954941	0.039849	1.000000	0.553481	0.351438	-0.017927	0.031733	-0.012709	0.079871	0.476158	-0.655463	0.203547	-0.297802	0.112679
TEDRATE	0.454527	-0.456473	-0.031665	-0.287170	0.669245	0.056658	0.553481	1.000000	-0.048267	0.197455	-0.007203	-0.011267	0.089182	-0.129335	-0.101389	0.376453	0.197223	0.126917
Slope	0.324784	0.257800	0.937724	-0.747158	0.293174	-0.005266	0.351438	-0.048267	1.000000	-0.142826	0.044722	0.054040	0.106720	0.828332	-0.711524	0.317360	-0.943186	0.004767
Curvature	-0.315317	-0.364356	0.090454	0.287593	0.028771	-0.007202	-0.017927	0.197455	-0.142826	1.000000	-0.009814	-0.015726	-0.121369	-0.419559	0.051521	0.504968	0.330540	-0.766053
10yr_cs_AR5	0.036164	-0.049588	0.039295	-0.032965	0.024784	-0.018788	0.031733	-0.007203	0.044722	-0.009814	1.000000	0.172617	0.053945	0.036561	-0.073578	0.000820	-0.043492	-0.008801
10yr_cs_AR20	-0.010104	0.000897	0.042752	-0.033371	-0.016189	-0.004831	-0.012709	-0.011267	0.054040	-0.015726	0.172617	1.000000	0.301966	0.045670	-0.086206	0.039034	-0.038302	0.021376
10yr_cs_AR60	-0.038942	0.122502	0.087160	-0.120497	0.080606	-0.004061	0.079871	0.089182	0.106720	-0.121369	0.053945	0.301966	1.000000	0.131333	-0.148206	0.191558	-0.041256	0.223632
UNRATE	0.356428	0.283270	0.720877	-0.857737	0.343768	0.007823	0.476158	-0.129335	0.828332	-0.419559	0.036561	0.045670	0.131333	1.000000	-0.812590	0.091818	-0.844542	0.272478
IP	-0.439106	0.085545	-0.701205	0.817466	-0.521457	0.013067	-0.655463	-0.101389	-0.711524	0.051521	-0.073578	-0.086206	-0.148206	-0.812590	1.000000	-0.308683	0.636057	-0.069403
PCA_Yield_1	-0.195849	0.087155	0.452304	-0.236602	0.249738	0.009819	0.203547	0.376453	0.317360	0.504968	0.000820	0.039034	0.191558	0.091818	-0.308683	1.000000	-0.000000	-0.000000
PCA_Yield_2	-0.383896	-0.244346	-0.845730	0.698862	-0.220878	0.010365	-0.297802	0.197223	-0.943186	0.330540	-0.043492	-0.038302	-0.041256	-0.844542	0.636057	-0.000000	1.000000	0.000000
PCA_Yield_3	0.245362	0.384373	-0.193874	-0.327395	0.094300	0.013373	0.112679	0.126917	0.004767	-0.766053	-0.008801	0.021376	0.223632	0.272478	-0.069403	-0.000000	0.000000	1.000000

Principal Component Analysis (PCA)

Reducing the dimension of predictors is called dimensionality reduction. This can be achieved in two ways, one by feature elimination and other by feature extraction.

Feature elimination: Reduce the dimensions by eliminating features. Although feature elimination maintains the interpretability of variables, we are running into a risk of losing the explainability / benefits from dropped variables.

Feature extraction: In feature extraction, we create new variables with each new variable as a combination of all old variables. Once we order these variables in terms of their explained variance ratio, we can drop the least important ones as we are not comfortable or not confident about the variables to be dropped through feature elimination. The main disadvantage with this process is independent variables become less interpretable.

One of the techniques is PCA, to reduce the dimensionality of the feature space to few orthogonal (correlation = 0) features explaining most of the variability.

PCA step by step procedure:

1. Normalize the data using $z = (x - \mu)/\sigma$
2. Compute the covariance two variables X and Y using the following formula:$\boldsymbol{Cov}\ (\boldsymbol{X}, \boldsymbol{Y}) = \rho(\boldsymbol{X}, \boldsymbol{Y}) * \boldsymbol{\sigma X} * \boldsymbol{\sigma Y}$ This results in a covariance matrix of A
3. Find the eigenvectors of 'A' which represents the direction of maximum variance and corresponding eigenvalues: ordered in decreasing marginal increase in variance by decreasing corresponding eigenvalues:
   $Av = \lambda v$, $Av - \lambda v = 0$ and $v(A - \lambda I) = 0$, with v not equal to 0 - find the determinant to get λ value lambda is the scalar value - eigenvalue and v is the eigenvector
4. Order in decreasing eigenvalues as they represent decreasing marginal increase in variance. Choose 'k' Eigen values based on the number of dimensions or principal components required.

Eigenvector is used because the transformation doesn't change the direction and is useful to extract information from raw data.

In this model, I have used k = 3 for reducing the dimensions of treasury yield curve features because converting to 3 features itself explains 99% of the variance in the data.

Evaluation metrics: (R square & MSE)

R square: Indicates the proportion of variance of the dependent variable explained or predicted from the independent variables. It ranges from 0 to 1 (i.e. 0 to 100%). Sometimes R square can be negative which implies the chosen model fits worse than a straight line and hence the model doesn't follow the trend of the data

Also can be formulated as Total variance explained by the model / Total variance

To achieve a better model, the target is to minimize or maximize objective function. The group of functions that are minimized are called "loss functions". One of the loss function we consider for regression models is reducing the Mean Squared Error between the actual and predicted values

Mean squared error (MSE): Calculated as the average of the squared terms of difference between predicted and observed values

Results: (Linear Regression)

Before application of PCA: R square: 0.39, MSE: 0.05, Post PCA: R square: 0.35, MSE: 0.05

Performing PCA slightly reduced the model performance, the reason might be because of the transformation of the original treasury yield curve features. To understand the significance of the treasury yield curve features, a decision tree is constructed below.

We got only 35% variance explainability because the major drawback of linear regression model is that it cannot consider the non-linearity in the data to make predictions. To bring the non-linearity, complex models like Random forest and XG Boost should be trained.

5.2. Decision trees

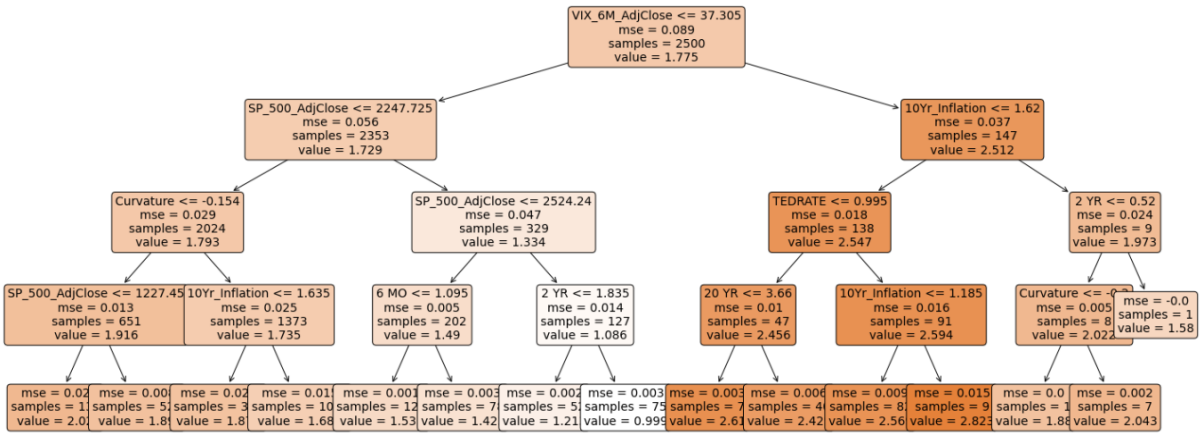
Decision trees build regression models in the form of a tree structure, which are one of the most useful supervised learning algorithms. They examine all the attributes of the dataset to find the ones that give the best possible result by splitting the data into subgroups. They perform this task recursively by splitting subgroups into smaller and smaller units until the Tree is finished (stopped by certain criteria).

As the model accuracy and performance depends on how the splits are made, for that decision, Decision trees use different algorithms that differ in the possible structure of the Tree (e.g. the number of splits per node), the criteria on how to perform the splits, and when to stop splitting.

Algorithm CART

CART is a DT algorithm that produces binary Classification or Regression Trees, depending on whether the dependent (or target) variable is categorical or numeric, respectively. It handles data in its raw form (no pre-processing needed), and can use the same variables more than once in different parts of the same DT, which may uncover complex interdependencies between sets of variables.

In the case of Regression Trees, the CART algorithm looks for splits that minimize the Least Square Deviation (LSD), choosing the partitions that minimize the result over all possible options. The LSD (sometimes referred as "variance reduction") metric minimizes the sum of the squared distances (or deviations) between the observed values and the predicted values. The difference between the predicted and observed values is called "residual", which means that LSD chooses the parameter estimates so that the sum of the squared residuals is minimized.



CART doesn't use an internal performance measure for Tree selection. Instead, Decision trees' performances are always measured through testing or via cross-validation, and the Tree selection proceeds only after this evaluation has been done (which will be discussed later)

Results: (Decision tree) R square: 0.52, MSE: 0.04

Above results show that the single decision tree regressor performs better than the linear regression. To further improve the model performance ensemble decision tree algorithms can be used as explained below.

5.3. Random Forest

A model composed of many models is called an Ensemble model. Two types of ensemble learning are Boosting, Bagging (also called Bootstrap Aggregation)

a. Boosting methods try to reduce the bias of the combined estimator by combining several weak models. Bias is explained as when there's less data, algorithms oversimplify the model by paying little attention to training data which in turn leads to a high difference between average prediction of model and correct value. This results in high error on both training and testing data
b. Bootstrap refers to random sampling with replacement. Bootstrap allows us to better understand the bias and the variance with the dataset. Bootstrap involves random sampling of a small subset of data from the dataset. It is a general procedure that can be used to reduce the variance for those algorithms that have high variance, typically decision trees.

   Bagging makes each model run independently and then aggregates the outputs at the end without preference to any model.

Decision trees are sensitive to the specific data on which they are trained. If the training data is changed the resulting decision tree can be quite different and in turn the predictions can be quite different. Also Decision trees are computationally expensive to train, carry a big risk of overfitting, and tend to find local optima because they can't go back after they have made a split.

To address these weaknesses, we turn to Random Forest which illustrates the power of combining many decision trees into one model.

Random forest is a bagging technique, and a decision tree based model which makes predictions by considering the non-linearity of the data. The trees in random forests are run in parallel with no interaction between these trees while building.

Random forest aggregates many decision trees with a number of features that can be split on at each node limited to some percentage of the total (which is known as the **hyper parameter**).

This ensures that the ensemble model does not rely too heavily on any individual feature, and makes fair use of all potentially predictive features. Each tree draws a random sample from the original data set when generating its splits, adding a further element of randomness that prevents overfitting. Hence, this helps prevent the trees from being too highly correlated.

Advantage is that it's one of the most accurate learning algorithms available which runs efficiently on large databases. It can handle thousands of input variables without variable deletion and generates an internal unbiased estimate of the generalization error as the forest building progresses.

Results: (Random forest Model) Performance of model on Training & Testing data

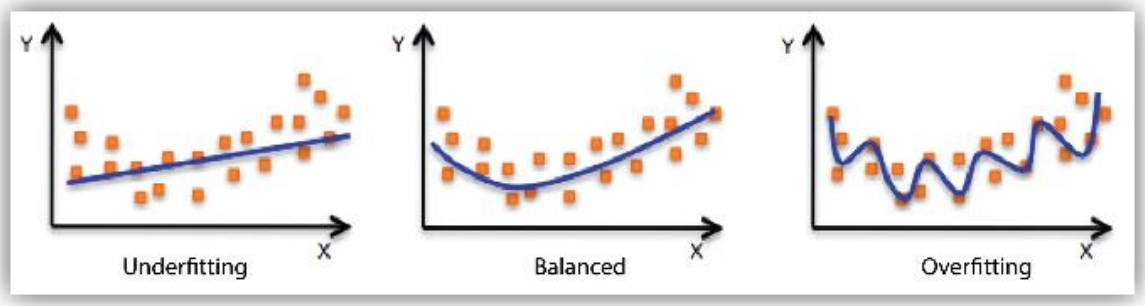
```
[65] # performance metrics for train data

     y_pred_train_rf_pre =  rf.predict(x_train_pre_pca)
     # The mean squared error
     print('Mean squared error: %.2f'
           % mean_squared_error(y_train_pre_pca, y_pred_train_rf_pre))
     # The coefficient of determination: 1 is perfect prediction
     print('Coefficient of determination: %.2f'
           % r2_score(y_train_pre_pca, y_pred_train_rf_pre))
```

```
Mean squared error: 0.00
Coefficient of determination: 1.00
```

```
[66] # performance metrics for test data

     print('Mean squared error: %.2f'
           % mean_squared_error(y_test_pre_pca, y_pred_rf_pre))
     # The coefficient of determination: 1 is perfect prediction
     print('Coefficient of determination: %.2f'
           % r2_score(y_test_pre_pca, y_pred_rf_pre))
```

```
Mean squared error: 0.02
Coefficient of determination: 0.74
```

From the above results, it is clear that the RF model is performing better on the training data than the testing data. One potential reason for this is that the model might be over fitted on the training data (Overfitting/variance is memorizing the training data instead of finding the actual patterns in the data which reduces the models generalization power).



To reduce this overfitting, we can tune the Random forest parameters to make the model

generalizable over the whole distribution of data. Tuning the parameters like number of trees, min samples split, no of features to be used on each tree will help to reduce the model complexity and thus improve the models generalization power. The only way to choose the right parameters is by training the model on the different values for these parameters and then choosing the best ones out of them that perform well on the Testing data. But by choosing the parameters only based on one set of testing data can make the model more sensitive or specific for one set of data and might not generalize over other datasets. To make the hyper parameter tuning more appropriate, we use cross validation for choosing the tuned parameters.

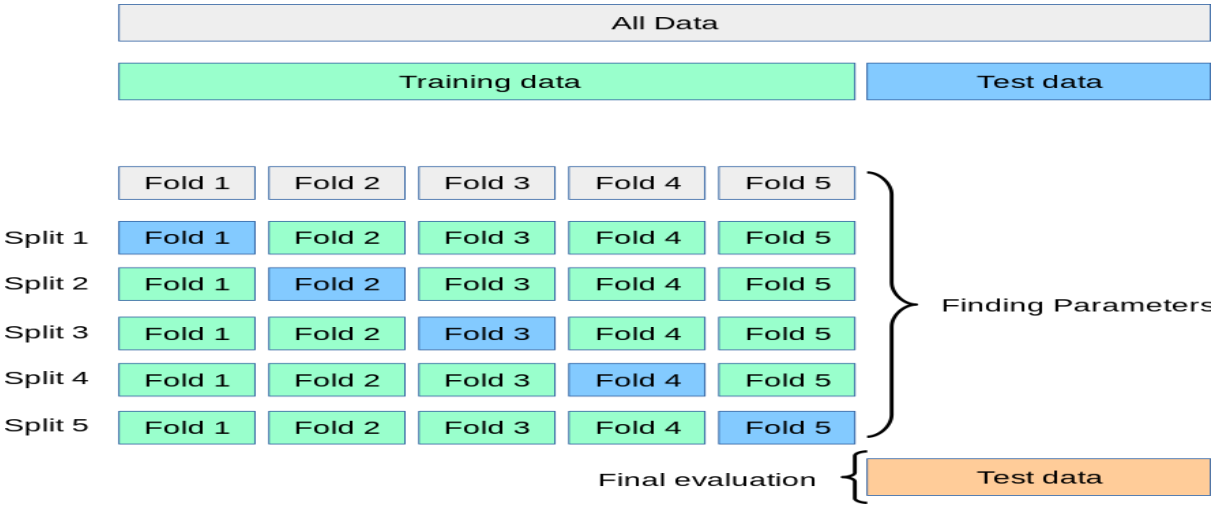
Cross validation

For hyper parameter tuning, we use K-fold cross-validation technique to evaluate predictive models by dividing the original sample into a training set to train the model, and a test set to evaluate it.

Procedure:

1. Split the dataset into k equal partitions
2. Use first fold as testing data and union of other folds as training data and calculate testing accuracy
3. Repeat step 1 and step 2. Use different set as test data different times. That is if we are dividing the dataset into k folds. On the first iteration, 1st fold will be test data and union of rest will be training data. Then we will calculate the testing accuracy. Then on next iteration 2nd fold will be test data and union of rest will be training data. Likewise, we will do for all folds.

Take the average of these test accuracy as the accuracy of the sample.



Note: Cross validation should always be done on the training data, testing data should not be used in the whole tuning process. If we include the testing data, this will make the model to select the best parameters by also considering the test data i.e, future data. So always CV should be performed on the training data.

Performance of the RF model - post tuning:

```
[ ]  # performance metrics for train data

     y_pred_train_rf_hp =  rf_hp.predict(x_train_pre_pca)
     # The mean squared error
     print('Mean squared error: %.2f'
           % mean_squared_error(y_train_pre_pca, y_pred_train_rf_hp))
     # The coefficient of determination: 1 is perfect prediction
     print('Coefficient of determination: %.2f'
           % r2_score(y_train_pre_pca, y_pred_train_rf_hp))

     #increase of estimators, coefficient of determination decreases in negative terms like from -0.65 to -0.57
```

```
Mean squared error: 0.01
Coefficient of determination: 0.90
```

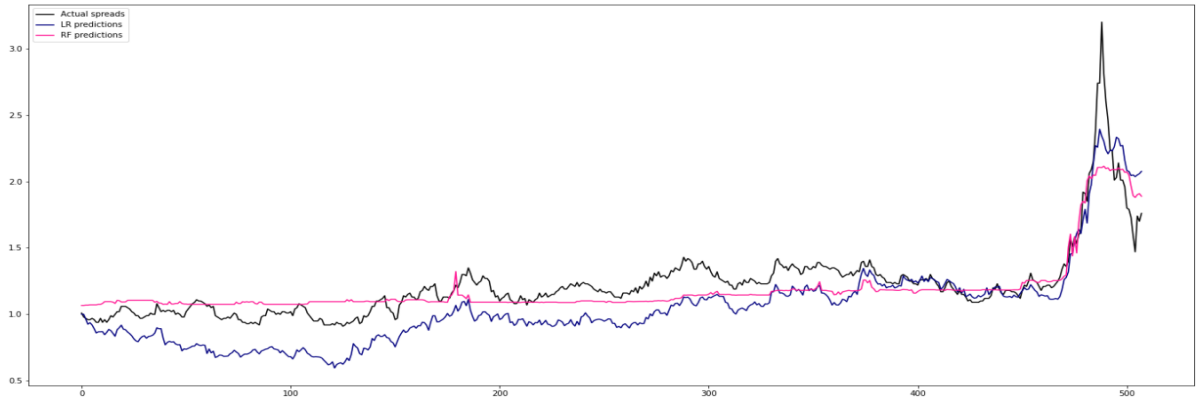
```
[ ]  # performance metrics for test data

     print('Mean squared error: %.2f'
           % mean_squared_error(y_test_pre_pca, y_pred_rf_hp))
     # The coefficient of determination: 1 is perfect prediction
     print('Coefficient of determination: %.2f'
           % r2_score(y_test_pre_pca, y_pred_rf_hp))

     #increase of estimators, coefficient of determination decreases in negative terms like from -0.65 to -0.57
```

```
Mean squared error: 0.02
Coefficient of determination: 0.76
```

After performing hyper parameter tuning, the difference between training and testing RMSE is less, proving that the tuned model is more generalizable compared to the original model.

Although the Random forest model has lower RMSE values, looking at the actual predictions it is clear that linear regression is able to predict the trend of the credit spread in a better way. To further get the better performance on trend and RMSE, we continue to test other models.

PLOT



PYCARET

Before implementing other regression trees, PyCaret can be used to check the different regression models performance. It's an open source, low-code machine learning library in Python to train and deploy machine learning pipelines and models in production.

Results:

	Model	MAE	MSE	RMSE	R2	RMSLE	MAPE
0	Extra Trees Regressor	0.026500	0.001500	0.038300	0.983100	0.013100	0.014700
1	CatBoost Regressor	0.032500	0.002200	0.046400	0.975500	0.015700	0.018000
2	Random Forest	0.032300	0.002400	0.049300	0.972200	0.016900	0.018000
3	Light Gradient Boosting Machine	0.033800	0.002600	0.050900	0.970400	0.017700	0.018900
4	Gradient Boosting Regressor	0.048500	0.004200	0.064900	0.951700	0.022700	0.027100
5	Extreme Gradient Boosting	0.048700	0.004300	0.065400	0.951000	0.022800	0.027200
6	Decision Tree	0.042300	0.005400	0.072400	0.939900	0.024800	0.023500
7	Bayesian Ridge	0.057000	0.005400	0.073600	0.937900	0.025700	0.031800
8	Linear Regression	0.056900	0.005400	0.073700	0.937800	0.025600	0.031800
9	Random Sample Consensus	0.056300	0.005700	0.075700	0.934400	0.026200	0.031400
10	Ridge Regression	0.060500	0.006100	0.077900	0.930500	0.027200	0.033900
11	AdaBoost Regressor	0.082600	0.010200	0.100800	0.883700	0.036000	0.047100
12	K Neighbors Regressor	0.081300	0.013700	0.117000	0.843600	0.041100	0.045700
13	Orthogonal Matching Pursuit	0.121200	0.024700	0.156900	0.718400	0.055900	0.070200
14	Huber Regressor	0.123300	0.025000	0.158000	0.713300	0.056000	0.070800
15	Support Vector Machine	0.145600	0.034300	0.184800	0.610500	0.063500	0.080600
16	Elastic Net	0.172700	0.047800	0.218300	0.456100	0.078500	0.100800
17	Lasso Regression	0.176500	0.050900	0.225400	0.420300	0.080200	0.102200
18	Passive Aggressive Regressor	0.230700	0.085800	0.282100	0.000800	0.106900	0.138600
19	Lasso Least Angle Regression	0.210200	0.088300	0.296800	-0.005100	0.107500	0.126300
20	Least Angle Regression	28.094200	60574.005200	154.013700	-762104.281100	1.583200	14.393600

One of the main disadvantages of the PyCaret package is that all the results for the algorithms shown are for the default parameters of that model. So these models might be over fitted or under fitted with the data. So, each model's performance can be further improved by hyper parameter tuning. These models serve as the best baseline to compare most of the regression ML models and further we can fine tune the top performing models from the above.

From the PyCaret results we can further tune the top performing models like Extra Trees Regressors XG Boost, BART etc.,

Extra Tree Regressors

The main difference between random forests and extra trees (usually called extreme random forests) lies in the fact that, instead of computing the locally optimal feature/split combination

(for the random forest), for each feature under consideration, a random value is selected for the split (for the extra trees). This leads to more diversified trees and less splitters to evaluate when training an extremely random forest.

Extra trees seem much faster (about three times) than the random forest method (at, least, in scikit-learn implementation). They seem to keep a higher performance in presence of noisy features,

Results: (Extra Tree Regressors) R square: 0.73, MSE: 0.02

We observe that when all the variables are relevant, both methods seem to achieve the same performance.

XG Boost

XG Boost is a boosting ensemble technique as explained above. In this method trees are built sequentially by giving more weightage to the errors in the previous tree. Boosting techniques usually try to address the bias and they might be more prone to overfitting. Initial results without hyper parameter tuning shows that Random forest (Bagging) is performing better than the boosting method for our case.

Results: (XG Boost) R square: 0.70, MSE: 0.02

5.4. Bayesian Additive Regression Trees (BART)

Since the random forest model has poor performance on prediction, we want to improve our tree model with better ensemble methods. Thus, we introduce a boosting tree model called: Bayesian Additive Regression Trees (BART) Model. In contrast with the random forest model, BART uses a boosting method to train its model. BART is a stronger learner for each tree in the model. Additionally, boosting is a great method for dealing with under fitting.

An essential fact about decision trees is that a given dataset can typically be described well by many different tree structures. This causes greedily-constructed trees to be very unstable, with small perturbations of trees leading to vastly different tree topologies. This motivates methods

based on ensembles of decision trees, such as bagging and random forests which increase estimation stability by averaging predictions over the bootstrap distribution of the decision tree.

Bayesian approaches fundamentally differ from these algorithmic approaches in that they posit a full probability model for the data, combining a prior distribution $\pi(T)$ for T with a likelihood $m(D \mid T)$ for the data $D = \{(Xi, Yi): 1 \leq i \leq N\}$.

This approach falls in framework of Bayesian nonparametric/semiparametric. This viewpoint allows for natural uncertainty quantification using the posterior distribution $\pi(T \mid D)$.

Algorithm

Defining priors on tree structures

We consider priors on (T, Θ_T) of the form: $\pi(T, \Theta_T) = \pi_T(T) \cdot \prod_{\eta \in L_T} \pi_\theta(\theta_\eta)$, That is, the θ_η's are conditionally independent given the tree structure T .

We require a prior distribution $\pi_T(T)$ for the tree structure. A common choice is to use a branching process prior. For each node η in the tree, we split η with prior probability $q(\eta) = \gamma(1 + D(\eta))^{-\beta}$, where $D(\eta)$ denotes the depth of η (where the root of the tree has depth $D(\eta) = 0$). The parameters (γ, β) are parameters which control the shape of the trees.

The parameter $\gamma > 0$ controls the prior probability that the tree is empty (and hence is typically large), while $\beta > 0$ penalizes trees which are too deep.

Once the tree topology is generated, we associate to each branch $\eta \in I_T$ a splitting rule of the form $[x_{j(\eta)} \leq C_\eta]$. We set $j(\eta) \overset{\text{indep}}{\sim} \text{Categorical}(s)$ where $s = (s_1, \ldots, s_P)$ is a probability vector and P is the dimension of the prediction X_i; most implementations set $s_j = P^{-1}$, although this choice is not ideal when the predictors do not have equal importance. Conditional on $j(\eta) = j$, we take $C_\eta \sim G_{T,\eta}$ where $G_{T,\eta}$ is a distribution G_j restricted to the set of values which do not lead to logically empty nodes when variable j is split on (if no such split exists, another predictor is selected according to s).

Furthermore, with no additional effort, the Bayesian approach naturally has many of the features of state-of-the-art tree ensembling methods. For example, the process of averaging

predictions across samples from the posterior distribution is similar to the averaging which occurs in the random forests algorithm, naturally leading to stability in inferences.

Framework

The BART framework replaces the single T with a sum of trees $\mu(x) = \sum_{t=1 \text{ to } T} g(x; T_t, \Theta_t)$, where $g(x; T_t, \Theta_t) = \mu_{\eta(t,x)}$, $\eta(t, x) = \eta$ if x is associated to leaf node η in tree T_t, and $\Theta_t = \{\mu_\eta : \eta \in LT_t\}$.

The tree structures Tt are then given independent priors as described above. The leaf parameters $\mu\eta$ are given iid Normal$(0, \sigma^2 \mu /T)$ priors, with the scaling factor T chosen so that $Var\{\mu(x)\} = \sigma^2 \mu$ irrespective of T. The BART model was initially motivated by analogy with boosting algorithms, which combine many so-called "weak learners" to construct predictions; the weak learners of choice in many boosting applications are shallow decision trees. By taking β large in the branching process prior, the BART model represents a Bayesian version of this idea with $\mu(x)$ represented as a sum of shallow decision trees.

Results: (BART) Mean squared error: 0.03, Coefficient of determination: 0.55

We observe that the results are better than single Decision tree but the disadvantage is BART implementation is very time consuming because of MCMC (Markov Chain Monte Carlo) sampling, hence it's tedious to tune the parameters and run the model again. So, we prefer to test models based on Neural Networks

6. ML models - Long Short Term Memory (LSTM)

LSTM is one of the deep learning methods, a special kind of Recurrent Neural Networks (RNN). These deep learning methods also help to save time from feature engineering which is not required.

Training a neural network has three major steps. First, it does a forward pass and makes a prediction. Here for operation of neuron non-linear activation function like Sigmoid or logistic function: $\sigma(z) = 1/(1+e^{-z})$ is used.

The assumption of a dependent variable to follow a sigmoid function inherently assumes a Gaussian distribution for the independent variable which is a general distribution we see for a lot of randomly occurring events and this is a good generic distribution to start with.

Second, it compares the prediction to the ground truth using a loss function. The loss function outputs an error value which is an estimate of how poorly the network is performing. Last, it uses that error value to do back propagation which calculates the gradients for each node in the network. In these conventional feed-forward neural networks, all test cases are considered to be independent. That is, when fitting the model for a particular day, there is no consideration for the data on previous days.

In order to capture the time-dependency, we can use Recurrent Neural Networks (RNN). However, RNN only remembers for small durations of time which is due to the vanishing gradient approach. LSTM layer architecture is instead built in such a way the network "decides" whether to modify its "internal memory" at each step. Doing so, and if properly trained, the layer can keep track of important events from further in the past, allowing for much richer inference.

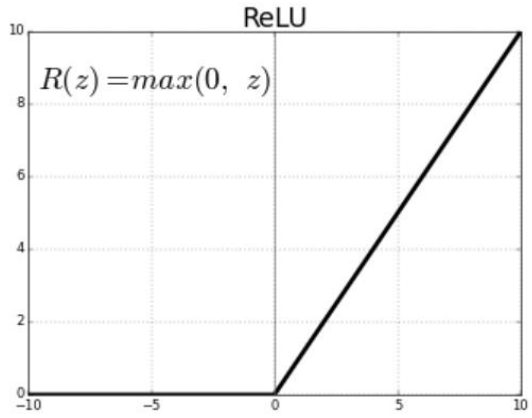
Vanishing gradient problem: As gradient is the value used to adjust the networks internal weights, allowing the network to learn; the bigger the gradient, the bigger the adjustments and vice versa. Here is where the problem lies. When doing back propagation, each node in a layer calculates its gradient with respect to the effects of the gradients, in the layer before it. So if the adjustments that cause gradients to exponentially shrink as it back propagates down. The earlier layers fail to do any learning as the internal weights are barely being adjusted due to extremely small gradients. And that's the vanishing gradient problem.

LSTM model, uses the sigmoid function and tanh functions in input and forget gates in a manner that it learns long-term and short term dependencies, proving to be an effective solution to this shortcoming.

Sigmoid function: sigmoid function squashes it's input into a very small output range [0,1] and has very steep gradients. Thus, there remain large regions of input space, where even a large change produces a very small change in the output. This problem increases with an increase in the number of layers and thus stagnates the learning of a neural network at a certain level.

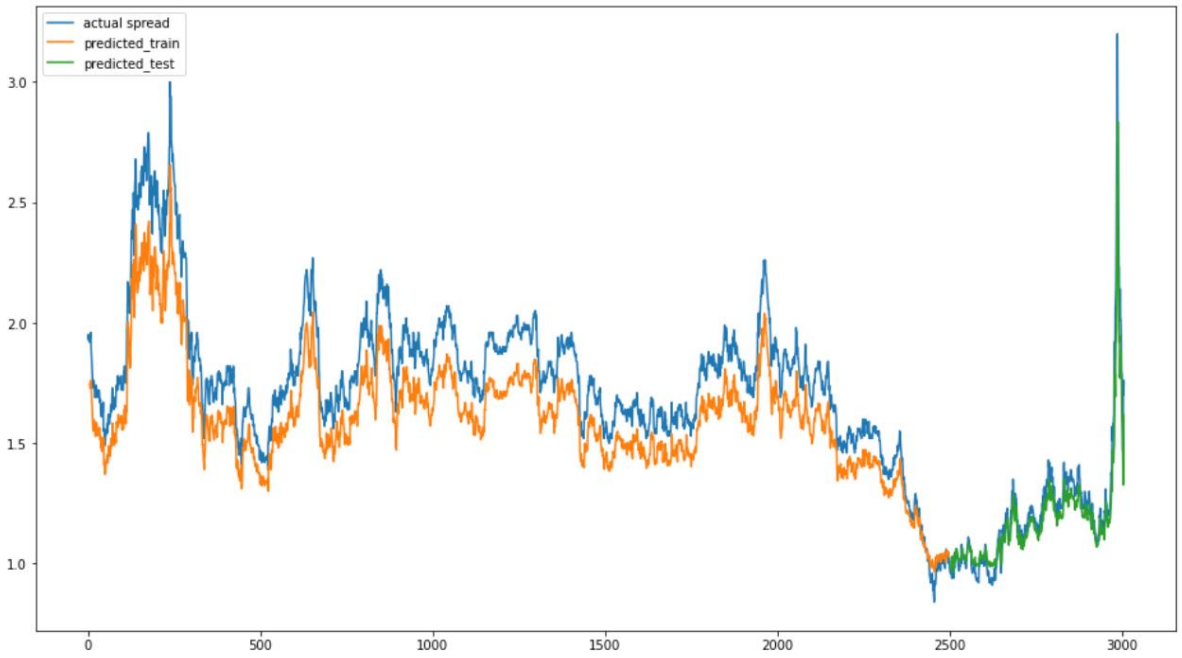
Tanh function: tanh is a suitable function whose second derivative can sustain for a long range before going to zero. tanh(z) function is a rescaled version of the sigmoid, and its output range is [− 1,1] instead of [0,1]. A higher gradient here helps in a better learning rate. However, the problem of vanishing gradients still persists in Tanh function.

ReLU function: The Rectified Linear Unit is the most commonly used activation function in deep learning models. The function returns 0 if it receives any negative input, but for any positive value x, it returns that value back. So, it can be written as f(x) = max (0, x).



The Leaky ReLU is one of the most well-known. It is the same as ReLU for positive numbers. But instead of being 0 for all negative values, it has a constant slope (less than 1.). In general practice, ReLU has found to be performing better than sigmoid or tanh functions.

Results: LSTM Train Score: 0.18 RMSE**,** Test Score: 0.08 RMSE



Above results show that the LSTM model performs better than other non-linear Machine learning models. One potential reason could be because of the lack of more feature engineering required for the tradional ML models. In the case of LSTM, it acts as an approximator function to capture all the nonlinearity given the required amount of data.

7. Conclusion

Credit spreads play a crucial role in the dynamic interaction of financial conditions with the real economy. As the movements in credit spreads may also reflect shifts in the effective supply of funds offered by financial intermediaries, which, in the presence of financial market frictions,have important implications for the usefulness of credit spreads as predictors of economic activity. So, it's very important to predict them, and can be extended to price illiquid corporate bonds. From the different machine learning methodologies that we built, it is observed that ensemble methods like random forest and sequential neural networks like LSTM are performing better for future credit spread predictions. These models can be further improved by creating more financial features or by tuning the model architecture.

References

http://www.csam.or.kr/journal/view.html?doi=10.29220/CSAM.2017.24.6.543

https://www.ml.cmu.edu/research/dap-papers/f18/dap-martin-daniel.pdf

https://medium.com/@amitkumar.singh/i-recently-completed-a-program-in-data-science-74edadd2c835

https://heartbeat.fritz.ai/understanding-the-mathematics-behind-principal-component-analysis-efd7c9ff0bb3

https://towardsdatascience.com/a-one-stop-shop-for-principal-component-analysis-5582fb7e0a9c

https://www.thirdway.org/report/covering-the-spread-credit-spreads-as-leading-indicators

https://www.scielo.br/scielo.php?script=sci_arttext&pid=S0103-65132018000100206

https://machinelearningmastery.com/how-to-calculate-precision-recall-f1-and-more-for-deep-learning-models/

https://medium.com/@shekharshashank1/2-words-code-to-compare-20-ml-regression-models-with-pycaret-8ed70c62a6b7

https://jakevdp.github.io/PythonDataScienceHandbook/05.03-hyperparameters-and-model-validation.html

https://towardsdatascience.com/logistic-regression-model-tuning-with-scikit-learn-part-1-425142e01af5

https://towardsdatascience.com/the-mathematics-of-decision-trees-random-forest-and-feature-importance-in-scikit-learn-and-spark-f2861df67e3

https://towardsdatascience.com/illustrated-guide-to-lstms-and-gru-s-a-step-by-step-explanation-44e9eb85bf21

https://towardsdatascience.com/illustrated-guide-to-lstms-and-gru-s-a-step-by-step-explanation-44e9eb85bf21