Machine Learning for Trading
Learning Algorithms Report

Table of Contents

Section I: Linear Regression Learner Analysis

For this part of the project, I created a data-generating algorithm called 'best4linreg.py'. The goal of this algorithm was to generate a dataset that performs significantly better with LinRegLearner than with KNNLearner. I utilized Python's csv library to write the generated data to a csv file, which was then used for testing and analysis on the learners.

My approach was to randomly space data points in a linear fashion [x1, x2, y]. In theory, this would allow for the linear regression learner to perform significantly better than the nearest neighbor approach due to the random nature of spacing between data points. I thought that the KNN would be thrown off by the random spacing between data points, while the linear regression learner would see improved performance with the strictly linear nature of the data. To do this I set x1 and x2 equal to the value of the counter squared. Squaring would allow for seemingly 'random' spacing between data points. I calculated the Y values according to the following relationship of X1 and X2: $y = x1 + 2(x2)$.

1000 data points were generated and stored in best4linreg.csv. A snapshot of the data has been provided below, and statistics and analysis follow on the next page.

Test Data Snapshot:

X1	X2	Y
1	1	3
4	4	12
9	9	27
16	16	48
25	25	75
36	36	108
49	49	147
64	64	192
81	81	243
100	100	300
121	121	363
144	144	432
169	169	507
196	196	588
225	225	675
256	256	768
289	289	867
324	324	972
361	361	1083
400	400	1200

…. Full data set includes 1000 data points in total

Test Data Statistics:

Linear Regression Learner using best4linreg.csv dataset

In sample results

- RMSE: 1.33338054743e-10
- corr: 1.0

Out of sample results

- RMSE: 6.25508959964e-10
- corr: 1.0

KNN Learner using best4linreg.csv dataset

In sample results

- RMSE: 1039.23013813
- corr: 0.999999918842

Out of sample results

- RMSE: 1044351.48738
- corr: nan

Note: the Linear Regression Learner has perfect correlation (strictly linear relationship) and also has near zero RMSE. Although the KNN learner does achieve high correlation, it does not perform nearly as well when it comes to RMSE. The output statistics shown above support the claim that best4linreg.csv achieves the goal of its generating algorithm.

Section II: KNN Learner Analysis

For this part of the project, I created a data-generating algorithm called 'best4KNN.py'. The goal of this algorithm was to generate a dataset that performs significantly better with KNNLearner than with LinRegLearner. I utilized Python's csv library to write the generated data to a csv file, which was then used for testing and analysis on the learners.

At first my approach here was to make the most non-linear dataset I could think of, as that would hamper the linear regression learner more than the KNN. As I tested various datasets out, I realized that the KNN learner almost always outperformed the linear regression learner, except for the case described in section I. For my best4KNN dataset, I used the following approach: I set X1 and X2 each equal to random integers (0,15). Then, I calculated the Y values as X1^2 + X2^3 with added noise to increase non-linearity.

In theory, this would allow for the KNN approach to perform significantly better than the linear regression approach due to the non-linear relationship (in all dimension) between the data points. 1000 data points were generated and stored in best4KNN.csv. A snapshot of the associated dataset has been provided below, and details and analysis follow on the next page.

Test Data Snapshot:

X1	X2	Y
9	3	115
5	10	1032
10	10	1102
3	5	143
4	14	2761
13	2	182
4	2	24
2	9	736
12	12	1880
9	5	211
2	11	1336
2	13	2204
14	12	1927
4	5	147
3	1	19
3	14	2754
9	5	211
3	3	41
10	7	445
14	8	715

…. Full data set includes 1000 data points in total

Test Data Statistics:

Linear Regression Learner + best4KNN.csv

In sample results

- RMSE: 365.935802885
- corr: 0.911562380551

Out of sample results

- RMSE: 348.49926686
- corr: 0.905809386317

KNN Learner + best4KNN.csv

In sample results

- RMSE: 25.052644571
- corr: 0.999606872997

Out of sample results

- RMSE: 43.2990617681
- corr: 0.998646660881

Note: the KNN Learner achieves significantly lower RMSE values in both in-sample and out-of-sample results. The KNN Leaner also achieves higher correlation values (.999, .998) than those achieved by the linear regression learner (.911, .905). The output statistics shown above support the claim that best4KNN.csv achieves the goal of its generating algorithm.

Section III: Bootstrap Aggregating Analysis

Dataset = ripple.csv
Algorithm = KNN

For which values of K does overfitting occur? (Don't use bagging).

Using ripple.csv for analysis, it can be shown that lower k values can cause over fitting.

# Bags	Correlation In sample	Correlation Out of sample	RMSE In sample	RMSE Out of sample
1	1.0	0.944111176194	0.0	0.237716222234
2	0.985971678512	0.952952269598	0.117863434564	0.213547134947
3	0.981360326901	0.955537498166	0.136590187312	0.207762150054
4	0.975383222588	0.954609910672	0.158319703229	0.212486985302
5	0.973427600347	0.951772965431	0.166240346459	0.221620653532
6	0.97161540002	0.946508234451	0.17594069633	0.237689111246
7	0.967528179895	0.944268967869	0.190372419945	0.247239572658
8	0.964988623322	0.938383482968	0.201496444354	0.259628727957
9	0.959418573483	0.93599266378	0.218001664854	0.268723178393
10	0.954402050655	0.934828370822	0.233914971909	0.275123388016
11	0.953123373291	0.930530334316	0.242855583286	0.287798930224
12	0.947323689615	0.927021610032	0.259662102279	0.300660721025
13	0.943783409718	0.920098605516	0.271635176783	0.316680688995
14	0.93856412722	0.914788474044	0.287449698456	0.32960095803
15	0.932949866613	0.911635612216	0.301414891142	0.340362526482
99	-0.202412678616	-0.19426360487	0.732705532827	0.724201479553

https://en.wikipedia.org/wiki/Overfitting. A good indicator of over fitting is where out of sample RMSE has a global minimum.

In this case: K=3

Now use bagging in conjunction with KNN with the ripple dataset. How does performance vary as you increase the number of bags? Does overfitting occur with respect to the number of bags?

Increasing the number of bags will improve performance at first, but there exists a threshold where further increases to the number of bags will incur negative affects of over fitting.

As the number of bags exceeds 20, the performance remains fairly constant.

# of Bags	Correlation In sample	Correlation Out of sample	RMSE In sample	RMSE Out of sample
1	0.960783993092	0.910775660141	0.197012335589	0.291759022171
10	0.983071896201	0.961405385833	0.134567170282	0.198413614062
20	0.984931474581	0.963632019079	0.128125363927	0.194305731247
30	0.985799218291	0.964009201094	0.12508226275	0.193120667938
40	0.985891900212	0.961516470139	0.12450416856	0.198734888362
50	0.985717474111	0.963229370898	0.124821666043	0.194860524783

Can bagging reduce or eliminate overfitting with respect to K for the ripple dataset?

Yes. The proper use of bagging and sample size can significantly improve overfitting with respect to K for the ripple dataset. By helping to aggregate multiple models, bagging can provide better performance than a KNN (K=1) approach. Increasing K results in the averaging or more 'votes' in each prediction, and produces a smoother curve.