PROJECT 3: Madelon Feature Selection + Classification

Author: William Buck

CLIENT INTRODUCTION:

Company XYZ.com has a complex dataset with potentially thousands of features. If our data science research firm wins the bid for this project, we will have to extract the meaningful features using a variety of feature selection techniques.

To demonstrate our firms' abilities to manage such complex problems and identify relevant features using machine learning models, I've analyzed the Madelon synthetic dataset, which exemplifies the issues related to complex feature selection with big data. Below is a description from the UCI Website.

> "Madelon is a two-class classification problem with continuous input variables. The difficulty is that the problem is multivariate and highly non-linear... Madelon is an artificial dataset containing data points grouped in 32 clusters placed on the vertices of a five dimensional hypercube and randomly labeled +1 or -1. The five dimensions constitute 5 informative features. 15 linear combinations of those features were added to form a set of 20 (redundant) informative features. Based on those 20 features one must separate the examples into the 2 classes (corresponding to the +-1 labels). We added a number of distractor feature called 'probes' having no predictive power. The order of the features and patterns were randomized."

This report will be split into two parts. Part 1 will consist of analysis on the standard Madelon dataset with 500 features, 4400 observations, 5 informative features, 15 linear combinations, and 480 noise features. Part 2 will be the same techniques that were refined during Part 1, but the data will be expanded to demonstrate the broader application of the techniques used for the smaller dataset. The Madelon data in Part 2 will be 1000 features and 200000 observations with the same number of informative features and linear combinations of those features.

PART 1: UCI DATA

EDA AND NOISE REMOVAL:

Since the artificially created Madelon data set has 5 important features and 15 linear combinations of those features, I decided a good place to start would be to use a regressor to compare every feature (column) against the rest of the dataset and find linear relationships between those features. I used a naive KNeighborsRegressor as well as a DecisionTreeRegressor for this, and was able to identify exactly 20 features that have mean model scores (r^2) that are positive (I ran the regression model multiple times and took the mean score). Additionally, most of the models scores are very high in the 0.8-0.9 range for the unpruned DecisionTree, for all three of the random data subsets. This means that each random sample set had exactly 20 correlated features that were universally identified by all samples and regression models that I tested.

	r2_score	feature
433	0.950393	feat_433
318	0.948537	feat_318
281	0.946645	feat_281
153	0.941721	feat_153
28	0.938253	feat_28
442	0.93649	feat_442
378	0.935396	feat_378
472	0.933099	feat_472
241	0.932783	feat_241
475	0.931758	feat_475
48	0.929233	feat_48
451	0.92462	feat_451
493	0.922693	feat_493
105	0.920188	feat_105
64	0.908813	feat_64
336	0.906259	feat_336
453	0.903833	feat_453
128	0.873082	feat_128
455	0.41356	feat_455
338	0.381006	feat_338
251	-0.626586	feat_251
454	-0.704861	feat_454
224	-0.730493	feat_224
45	-0.771957	feat_45
73	-0.772401	feat_73

	r2_score	feature
281	0.953785	feat_281
48	0.944713	feat_48
153	0.938098	feat_153
451	0.937697	feat_451
28	0.93592	feat_28
318	0.932708	feat_318
378	0.932335	feat_378
475	0.930833	feat_475
336	0.928195	feat_336
105	0.928092	feat_105
128	0.921867	feat_128
241	0.921	feat_241
433	0.919549	feat_433
453	0.918804	feat_453
64	0.900241	feat_64
442	0.895737	feat_442
493	0.894377	feat_493
472	0.89072	feat_472
338	0.45955	feat_338
455	0.409738	feat_455
60	-0.635815	feat_60
113	-0.646052	feat_113
269	-0.646775	feat_269
367	-0.683983	feat_367
469	-0.747116	feat_469

	r2_score	feature
493	0.940601	feat_493
128	0.936577	feat_128
318	0.93505	feat_318
451	0.93491	feat_451
48	0.933219	feat_48
241	0.932901	feat_241
28	0.932648	feat_28
153	0.922717	feat_153
105	0.921241	feat_105
442	0.918617	feat_442
378	0.916518	feat_378
433	0.914914	feat_433
336	0.911518	feat_336
281	0.910063	feat_281
64	0.908639	feat_64
453	0.901863	feat_453
472	0.895126	feat_472
475	0.890257	feat_475
455	0.516872	feat_455
338	0.360076	feat_338
220	-0.668756	feat_220
477	-0.67568	feat_477
130	-0.702042	feat_130
199	-0.713046	feat_199

Figure 1: Above is the mean model scores in DataFrames from the 3 different random 10% samples using a DecisionTreeRegressor. Each model was run a total of 10 times on each data sample and the average score is what is shown.

I believe that these results are a good indication that I have found the 5 informative features and the 15 linear combinations because 480/500 of the features in the Madelon dataset are noise, and all three of the 10% samples of the training dataset returned identical lists of features when compared to the other features.

BENCHMARK:

The UCI dataset has two perfectly balanced classes, so the benchmark accuracy score is 0.5. When I perform a train_test_split on the full dataset and fit a LogisticRegression, I was able to score marginally better than benchmark on the validation set with a train score of 0.74 and a validation score of 0.58. Using all of the features on the full dataset, I was able to score a 0.755 roc_auc_score, only tuning the n_neighbors hyperparameter for the KNeighborsClassifier model.

Further, I fit a naive, cross validated KNeighborsClassifer using the 20 features I identified above in Figure 1, which resulted in a validation score of 0.91333, showing a strong predictive ability of the naïve KNC model with the Madelon data using the 20 features. The roc_auc_score using the 20 features was .89333.

FEATURE SELECTION:

Separating the 15 linear combinations of the 5 informative features proved to be less straight forward than removing noise. Using SelectKBest, SelectFromModel, and Recursive Features Elimination, I identified a variety of feature combinations that were not universal across the different selection models and sample sets.

Summary of feature selection:

- SelectKBest():
  - Features that show in all three of the SelectKBest fits: 128, 241, 475
  - Features that show twice: 378
  - Features that show once: 64, 48, 338, 336

- SelectFromModel():
  - Using Lasso regularization:
    - Features that show in all three of the SelectFromModel fits: 475
    - Features that show twice: 378
    - Features that show once: 64, 105, 318, 336, 338, 442, 453
  - Using ElasticNet regularization:
    - 14 unique features identified: 64, 105, 153, 241, 318, 336, 338, 378, 442, 453, 455, 472, 475, 493

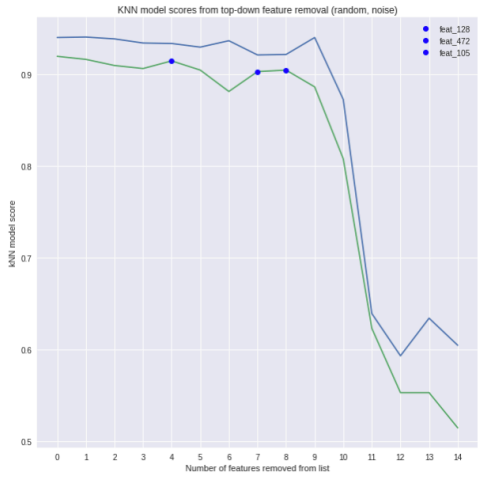
- 5 features that appear in both SFM and SKB with Lasso: 64, 336, 338, 378, 475
- 5 features identified from Recursive Feature Elimination: 28, 48, 105, 338, 475

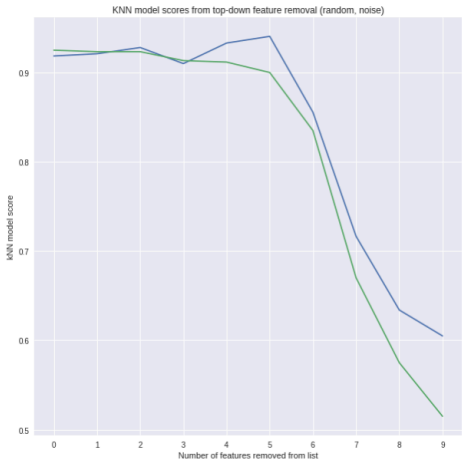
Feature list model scores:

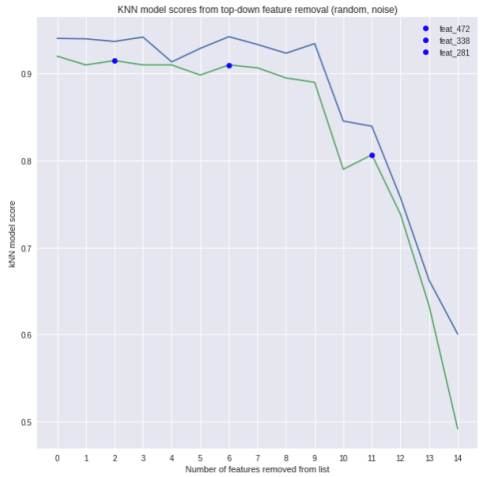
With the features that each feature selection model identified, I tested the entire UCI dataset with only the selected features. All of the lists of features that were identified from all three of the selection models scored lower when I used GridSearchCV with both KNeighborsClassifier and RandomForest than just using all 20 of the features I identified during the initial noise removal. In order to understand how feature removal was affecting the validation scores, I graphed the scores as I removed features and tried to identify which features where causing the score to increase when removed. The details of this method are contained in the madelon_analyzer.py file that is included with this write up.

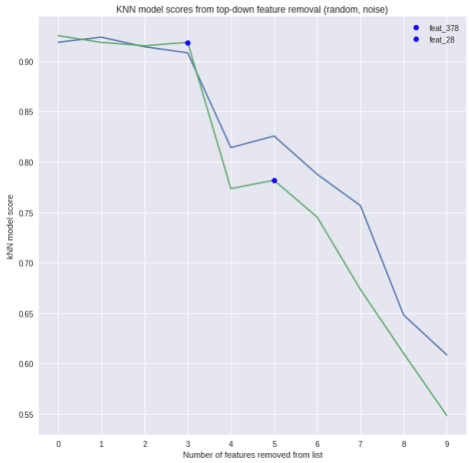
Below are graphs that are output using the list_top_dipped_feats function from the madelon_analyzer, with parameters ‘random’ and ‘noise’ set to True. The graphs show the features that were removed from the feature list marked in blue. By repeating this process several times, I was

able to improve my validation score to 0.931667 with 8 features. Below is Figure 2.1 and 2.2, showing the final two iterations of this process that resulted in the highest validation score of this analysis.









TOP RANDOMIZED NOISE FEATURES: [105, 128, 281, 338, 472]

[9]:

	noise_removal_1
train_score	0.9185
val_score	0.9250

TOP RANDOMIZED NOISE FEATURES: [28, 378]

]:

	noise_removal_1
train_score	0.922500
val_score	0.931667

Figure 2.1 (left) and 2.2 (right): The list of features is randomized and then one feature is removed at each iteration, and the model of choice (in this case KNeighborsClassifier) is fit and scored to the list of reduced features. The blue points on the green line represent the validation score of the model and what feature was removed when the model was fit.

RESULTS FOR PART 1:

I was able to narrow down the list to 8 features, but beyond that I was unable to raise r^2 or roc_auc scores with any less features. I tried all 56 unique combinations of 5 features from the 8 features and scored a 0.91333 validation score with GridSearched KNeighborsClassifier, and a .883 roc_auc score using RandomForest as a classifier. These top 5 features were 48, 153, 451, 455, 475.

PART 2: EXPANDED DATASET

EDA AND NOISE REMOVAL:

I used an identical approach as the smaller dataset on the 1000 feature dataset and once again all three random samples returned the same 20 features. However, the r^2 scores were not as high because I had a significantly smaller percentage of the total data. In order to have a higher certainty, I reduced the number of features using SelectKBest and retested with a 10%, 20000 row sample of the data. This resulted in extremely high r^2 scores for the features, and the exact same 20 were identified.

	r2_score	feature
956	0.876759	feat_956
639	0.843671	feat_639
315	0.601526	feat_315
269	0.513198	feat_269
341	0.509998	feat_341
724	0.483858	feat_724
867	0.477938	feat_867
701	0.473653	feat_701
336	0.465809	feat_336
736	0.458448	feat_736
395	0.448664	feat_395
769	0.43618	feat_769
526	0.30044	feat_526
920	0.27338	feat_920
504	0.229805	feat_504
681	0.220966	feat_681
257	0.167354	feat_257
829	0.141994	feat_829
308	0.121009	feat_308
808	-0.242516	feat_808

	r2_score	feature
639	0.845707	feat_639
956	0.808271	feat_956
315	0.575084	feat_315
867	0.485985	feat_867
341	0.462097	feat_341
701	0.417449	feat_701
724	0.409274	feat_724
526	0.347879	feat_526
769	0.327253	feat_769
829	0.314855	feat_829
269	0.311286	feat_269
336	0.293929	feat_336
920	0.290956	feat_920
395	0.284413	feat_395
257	0.279773	feat_257
808	0.211217	feat_808
736	0.200698	feat_736
504	0.198587	feat_504
681	0.188266	feat_681
308	0.0590268	feat_308

	r2_score	feature
639	0.806627	feat_639
956	0.754375	feat_956
315	0.573237	feat_315
724	0.519356	feat_724
867	0.475584	feat_867
336	0.472721	feat_336
269	0.467021	feat_269
701	0.390734	feat_701
920	0.375489	feat_920
681	0.315068	feat_681
395	0.291377	feat_395
736	0.255419	feat_736
257	0.250702	feat_257
829	0.23143	feat_829
341	0.182425	feat_341
769	0.176203	feat_769
504	0.172482	feat_504
808	0.124137	feat_808
308	-0.0130461	feat_308
526	-0.329475	feat_526

	r2_score	feature
67	0.990509	feat_639
97	0.988641	feat_956
23	0.985173	feat_269
87	0.980179	feat_867
36	0.979375	feat_341
43	0.976142	feat_395
93	0.970617	feat_920
73	0.967101	feat_724
71	0.966585	feat_701
31	0.964912	feat_315
34	0.963006	feat_336
74	0.960983	feat_736
21	0.95288	feat_257
80	0.950582	feat_808
58	0.949982	feat_504
76	0.949542	feat_769
83	0.945805	feat_829
28	0.944273	feat_308
60	0.941839	feat_526
70	0.930325	feat_681

Figure 3: Above is the mean model scores in DataFrames from the 3 different random small samples using a DecisionTreeRegressor, and the test again with reduced features (DataFrame on the left). Each model was run a total of 10 times on each data sample and the average score is what is shown.

BENCHMARK:

The dataset has two perfectly balanced classes, so the benchmark accuracy score is 0.5. When I fit a naive, cross validated KNeighborsClassifer using the 20 features I identified above in Figure 3, I got

a validation score of 0.83033, showing a strong predictive ability of the naïve KNC model with the Madelon data using the 20 features. The roc_auc_score using the 20 features was 0.8096.

While this score is lower than the analysis done in Part 1, it still shows that the model has learn the data using these 20 features and make accurate predictions.

FEATURE SELECTION:

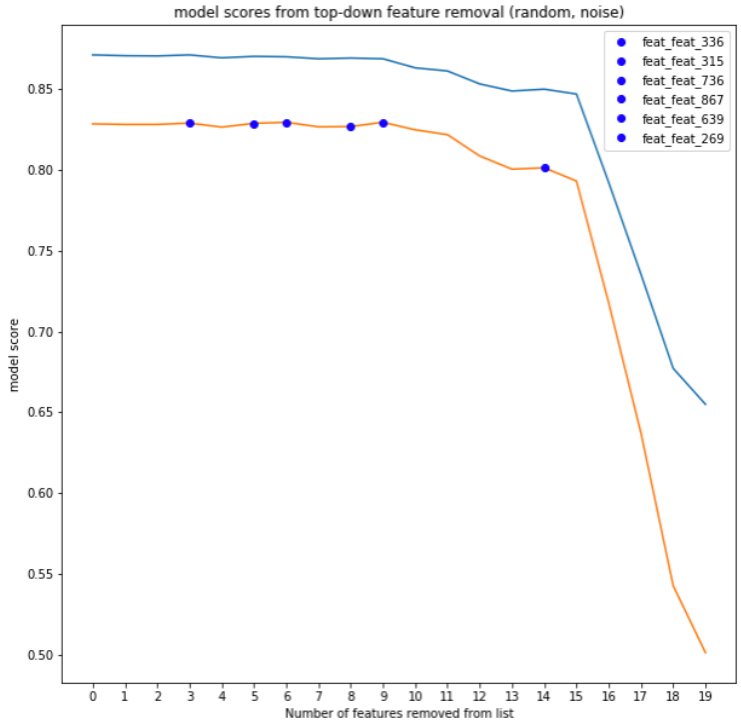
Using a naive KNeighborsClassifier, I was able to score a 0.762284 for my validation set using 5 features chosen by SelectFromModel with Lasso regularization on a subset that included 6506 observations. The reason that I chose to use 6506 features is because it is a representative sample of the population with a 90% confidence level and a 1% margin of error.

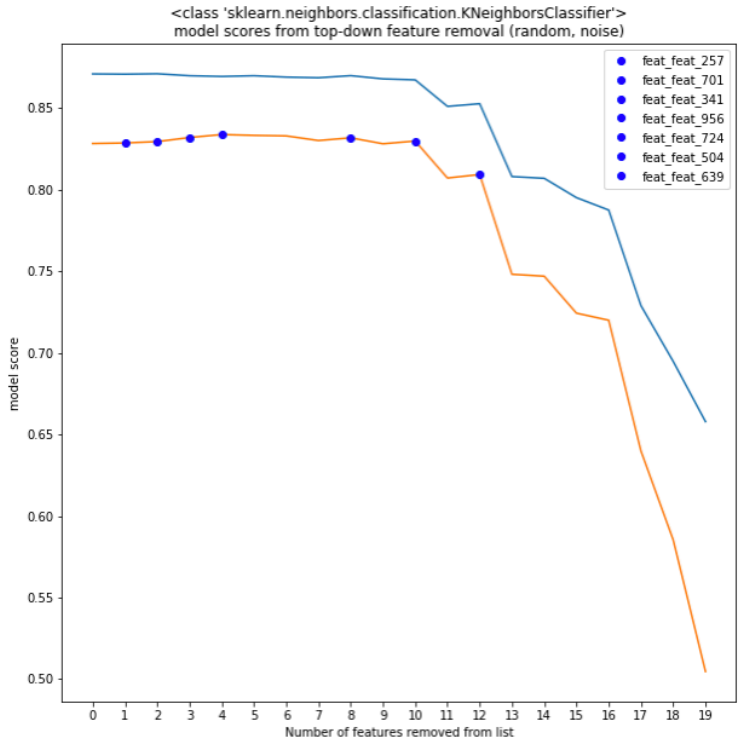
However, just like with the smaller UCI dataset, I found that the different feature selection methods failed to outperform the 20 features I identified using the regression technique detailed in the EDA and Noise Removal section.

	train_score	val_score
my_features	0.8802	0.830333
sfm_feats	0.857	0.762284
RFE_cols	0.785736	0.736
skb_100_feats	0.7088	0.552457
benchmark_plus_skbtop100	0.7122	0.538513
first_3_bd	0.6855	0.523333
first_3_bd_6506	0.6786	0.518592
sfm2_feats	0.699	0.508632
skb_20_feats	0.6782	0.49668

Figure 4: Below is a description of each score and where it came from:
Row 0 - 20 columns identified in EDA and Noise Removal
Row 1 - 5 features identified by SFM.
Row 2 - 20 features from RFE.
Row 3 - SKB top 100.
Row 4 - 20 columns identified using regression, plus the SKB top 100.
Row 5 - 20 columns identified using regression tested on 3000 sample.
Row 6 - Columns identified using regression tested on a different 6506 row sample.
Row 7 - Used SFM on only the top 20 features identified using regression.
Row 8 - SKB 20 features

Using the madelon_analyzer in the exact same way as Part 1, I removed noise features to increase the model score to 0.83450 with 14 features, which is only marginally better than using all 20 features. After removing 6 features, I was unable to raise the score any further from top down feature elimination.





	with_noise_removal
train_score	0.87095
val_score	0.83450

Figure 5: The above graphs were the output of the final iteration of the madelon_analyzer top-down feature elimination function. The process was less successful at removing features than the smaller dataset, but was still successful at raising the score.

RESULTS FOR PART 2:

After reducing the features to 14, I then fit and scored models using GridSearchCV with KNeighborsClassifier on all combinations of 5 features with a 10% sample. The highest model score I could achieve was 0.8365 with features 920, 681, 736, 808, and 526, which is slightly better than the score I achieved with 14 features. The roc_auc score for these features came out to 0.8140, which is slightly better than the 20 features I identified using the madelon_analyzer.

CONCLUSION:

The Madelon dataset's complexity makes it particularly hard to identify relevant features for classification. However, this analysis has proved that our research firm is capable of reducing the complexity of the problem by finding the most important features for predictions using a variety of techniques and models. As we showed in Part 2, we can still use these same techniques applied to smaller datasets on random samples of a larger dataset with success.

Even though I made sure to compare the scores of multiple different models, more could be done given time and resources. If a deeper analysis were to be performed on the Madelon data, several things could be done to improve the predictive power of our models. With more computational power, we could analyze larger, more representative sample of the overall data. Also, we could continue to tune parameters of all the models to maximize predictive power.