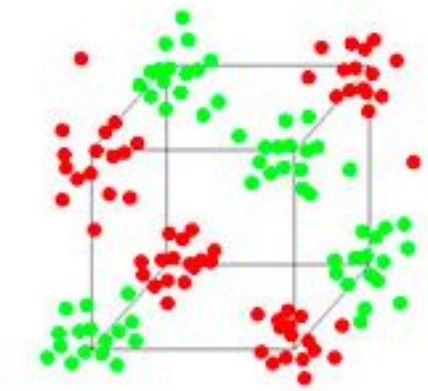
General Assembly - Data Science Immersive

Madelon Dataset Project

identify relevant features using machine learning



Introduction

When working with real data you may have hundreds or thousands of features and the patterns will be much more nuanced. In that real world case, the importance of feature engineering is to use domain specific knowledge and human insight to ensure that the data contains relevant indicators for the prediction task. By identify relevant features in the early stage in research will not only increase the effectiveness of the predictive model but also save amount of the time that needed to be spent for computations and resources that required by such process. In this report, we will work on artificially produced dataset that contains multivariate and highly non linear data points. Through various feature selection techniques, we will examine the variance in the effectiveness of the machine learning classification models.

Agenda

1. Introduce the Data
2. Basic Data Exploration
    1. Basic statistics of data
    2. Plotting distributions
3. Sampling

4. Benchmark Models
    1. Logistic Regression
    2. Decision Tree
    3. K Nearest Neighbors
    4. Support Vector Classifier
5. Feature Engineering/Selection
    1. Select K Best feature selection with f_classif scoring
    2. Logistic Regression CV with Lasso penalty (l1)
    3. Select from model using Logistic Regression
    4. Select From Model using Decision Tree Classifier
    5. Select from Model using Support Vector Classifier
    6. Select from Model using K Nearest Neighbor Classifier
    7. Recursive Feature Elimination using Logistic Regression
    8. Dimension reduction through Principal Component Analysis
6. Feature Selection and Model Building
7. Repeat step 5 & 6 for Instructor's data

1. Introducing The Data

The Data

We are going to utilize two dataset to demonstrate the effectiveness of feature engineering. Both of them are artificially generated to contain informative, redundant, repeating ,and noisy features. The first dataset is called madelon, containing 500 features for two classifications 1 and -1.

Here is brief information on it.

```
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 di
mensions constitute 5 informative features. 15 linear combinations of those features wer
e 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 or
der of the features and patterns were randomized.
```

Another data is also artificially created with similar goal in mind and contains 1000 features with 200,000 data points total. I will create subsets of these dataset to explain and demonstrate the effective way to feature engineer and derive better predictions.

We will use python and its libraries to conduct this report.

Python libraries:

- Numpy
- Pandas
- Sci-kit learn
- Matplotlib
- Almost entire workflow is covered by these four libraries

2. Basic Data Exploration

Exploratory Data Analysis is conducted in the initial stage to provide better understanding of the structure of the data and to spot visually any patterns or distinctive outliers in the raw data. Basic stats or graphs showing distribution of data or correlation is usually a good tool.

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 2000 entries, 0 to 1999
Columns: 500 entries, 0 to 499
dtypes: int64(500)
memory usage: 7.6 MB
```

	0	1	2	3	4	5	6	7	8	9	...	490	491	492	493	494	495	496	497	498	499
0	485	477	537	479	452	471	491	476	475	473	...	477	481	477	485	511	485	481	479	475	496
1	483	458	460	487	587	475	526	479	485	469	...	463	478	487	338	513	486	483	492	510	517
2	487	542	499	468	448	471	442	478	480	477	...	487	481	492	650	506	501	480	489	499	498
3	480	491	510	485	495	472	417	474	502	476	...	491	480	474	572	454	469	475	482	494	461
4	484	502	528	489	466	481	402	478	487	468	...	488	479	452	435	486	508	481	504	495	511

Fig 1.0 = Data type and the first five rows of our 500 feature data set.

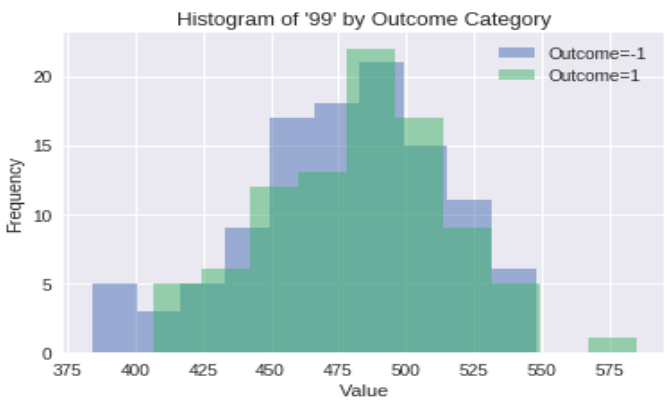


Fig 2.0 - Distribution of points in feature "99"

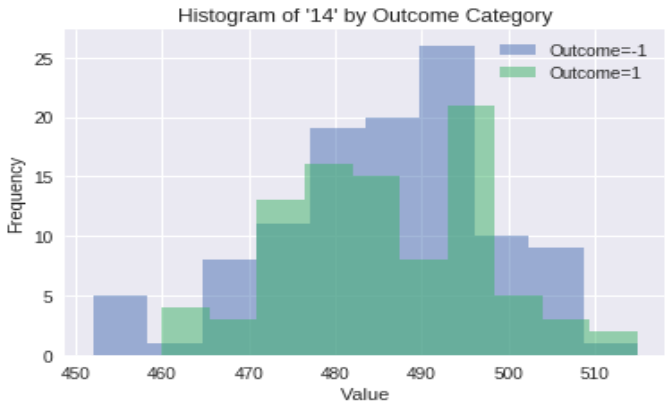


Fig 2.1 - Distribution of points in feature "14"

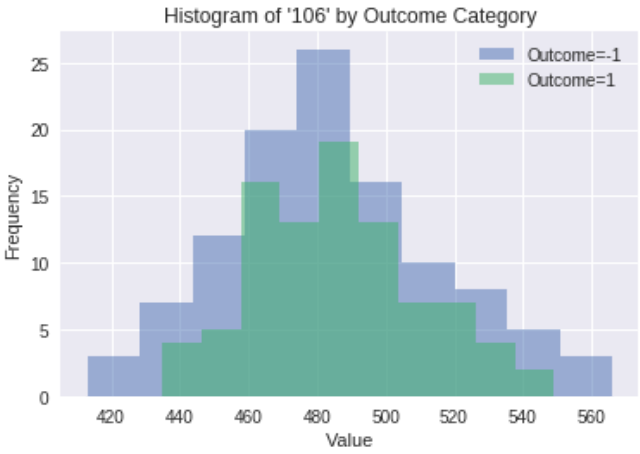


Fig 2.2 - Distribution of points in feature "106"

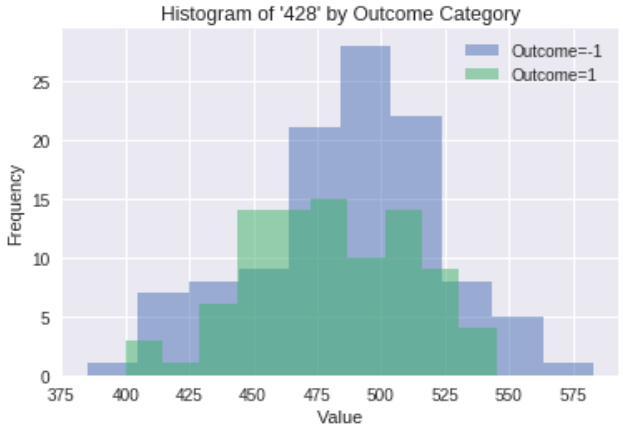


Fig 2.3 - Distribution of points in feature "428"

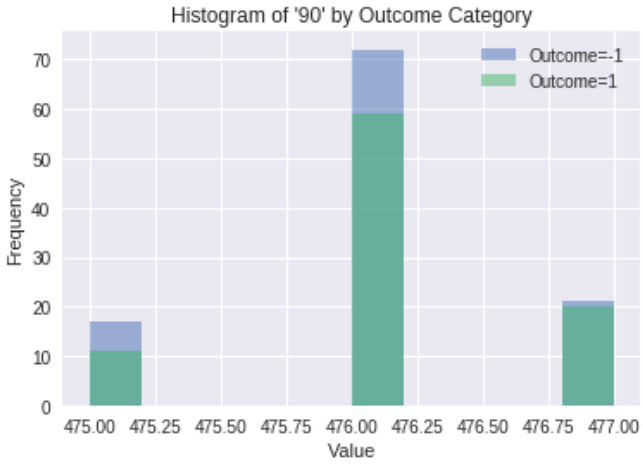


Fig 2.4 - Distribution of points in feature "90"

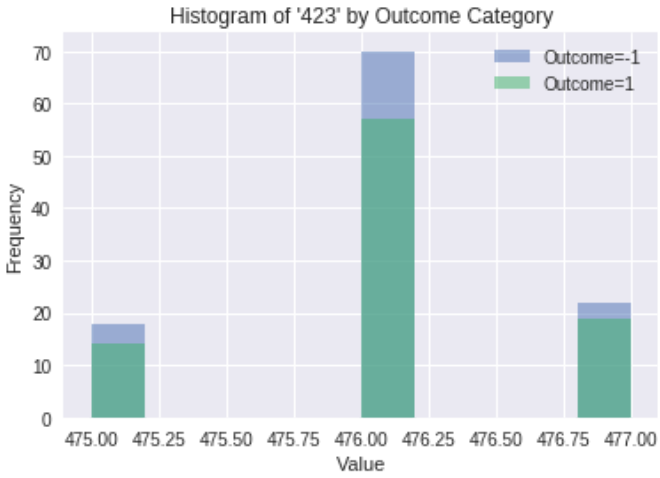


Fig 2.5- Distribution of points in feature "423"

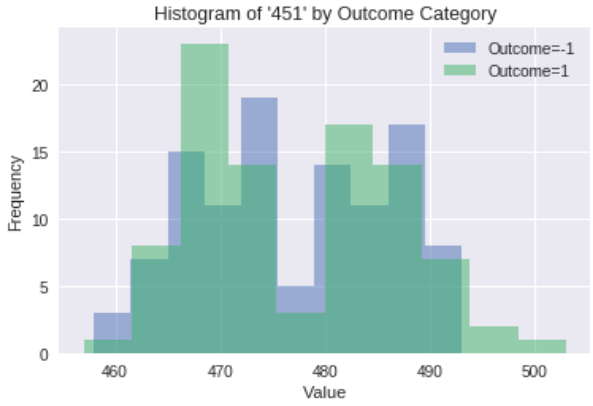


Fig 2.6 - Distribution of points in feature "451"

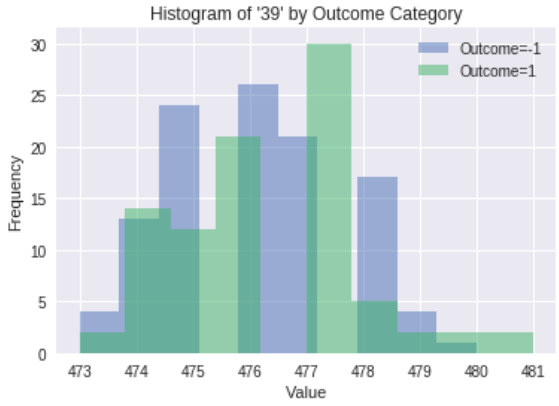


Fig 2.7 - Distribution of points in feature "39"

	count	mean	std	min	25%	50%	75%	max
203	2000.0	508.1945	42.703132	332.0	481.0	509.0	537.0	647.0
26	2000.0	484.7835	21.573278	421.0	471.0	485.0	499.0	573.0
296	2000.0	475.1750	24.965957	394.0	459.0	475.0	491.0	561.0
198	2000.0	476.5120	1.404223	472.0	476.0	477.0	477.0	481.0
9	2000.0	478.7890	7.190920	455.0	474.0	479.0	484.0	503.0
170	2000.0	494.6405	42.958701	350.0	467.0	495.0	522.0	639.0
485	2000.0	482.7790	11.275936	444.0	475.0	483.0	490.0	523.0
363	2000.0	486.1715	24.397596	410.0	469.0	486.0	502.0	573.0
92	2000.0	479.3155	20.979061	399.0	465.0	479.0	493.0	551.0
73	2000.0	484.1835	36.814080	354.0	460.0	484.0	508.0	629.0
126	2000.0	483.3870	12.921018	440.0	475.0	483.0	492.0	530.0
97	2000.0	478.9505	23.181345	406.0	463.0	479.0	494.0	554.0
316	2000.0	477.3155	10.116478	448.0	471.0	477.0	484.0	510.0
93	2000.0	492.5830	18.754332	438.0	480.0	492.0	505.0	557.0
271	2000.0	498.9620	41.093624	369.0	471.0	499.0	526.0	639.0
21	2000.0	494.2390	22.344295	412.0	479.0	494.0	509.0	571.0
307	2000.0	480.8795	8.985062	450.0	475.0	481.0	487.0	512.0
163	2000.0	479.9680	6.126316	458.0	476.0	480.0	484.0	500.0
192	2000.0	495.4240	19.671291	428.0	483.0	496.0	508.0	559.0
109	2000.0	480.8710	10.096154	442.0	474.0	481.0	487.0	518.0

Fig 3.0 - Basic Statistics of the 20 sampled feature from the whole dataset.

Figure 1 shows how the data is structured with no missing value and only integers. Thus concern of dealing with null value or categorical features is not an issue with this dataset.

Figure 2 shows examples of how the data is spread by the features. It is inefficient to plot all of five hundred histograms so I went through and posted some interesting ones. FIgure 2.2 and 2.3 both shows the normal distribution and this applies to most of the features. We can infer that our data follows more of normal distribution from features to features. It's minimal in its impact, we do find similar patterns like figures 2.0 and 2.1 where figures do show small points of outliers but nothing extreme. Not all features followed same distribution. Although very balanced, figures 2.4 and 2.5 showed clusters of points in distribution. Although nothing inferential, it's important to note that some features followed exact same pattern. Figures 2.6 and 2.7 does show that some features had a lot of overlaps and some didn't.

These findings does seem to align with the fig 3.0. Where we find the mean of the data points were not far off from each other while other factors being very varying from features to features.

3. Sampling

Since we are dealing with a large dataset, we are going to utilize sampling. As we go through the demonstration of feature selection process, the effectiveness of the sampling will also be tested and demonstrated.

For Madelon dataset, I will use “train.data” and its label file as train data and validation datas will be used as the test data while for instructor created dataset, we will conduct train test split. Three samples of 10% of dataset will be drawn from madelon and 5% of whole data set from instructor.

	0	1	2	3	4	5	6	7	8	9	...	491	492	493	494	495	496	497	498	499	label
674	487	470	536	473	506	465	543	477	478	474	...	475	507	438	523	442	487	485	489	523	-1
1699	467	446	518	494	491	483	497	478	471	488	...	483	471	294	477	497	474	478	459	454	1
1282	486	496	463	473	488	482	432	474	500	476	...	482	520	535	517	461	479	513	552	478	1
1315	480	444	514	489	536	473	470	477	486	486	...	482	445	657	481	557	478	484	468	490	1
1210	486	549	594	480	439	479	370	475	496	483	...	469	478	437	476	460	477	519	495	483	1

5 rows × 501 columns

	0	1	2	3	4	5	6	7	8	9	...	491	492	493	494	495	496	497	498	499	label
278	481	497	489	489	475	488	573	474	510	486	...	479	515	614	524	514	470	465	489	481	1
492	482	461	517	490	485	476	484	475	493	480	...	479	498	456	435	553	489	495	523	492	-1
1266	491	503	500	485	565	479	588	477	468	485	...	479	487	721	480	548	482	479	519	482	1
557	481	498	491	475	489	472	456	476	457	488	...	481	509	335	440	517	478	482	541	524	1
871	494	490	494	483	467	474	461	478	494	477	...	485	533	495	496	510	480	463	446	487	1

5 rows × 501 columns

	0	1	2	3	4	5	6	7	8	9	...	491	492	493	494	495	496	497	498	499	label
933	469	488	558	477	531	477	482	476	493	482	...	482	484	388	496	457	490	483	540	504	-1
1459	484	517	562	489	526	477	444	476	517	481	...	482	458	382	516	533	479	510	478	510	-1
1338	479	489	434	487	388	481	455	476	495	483	...	482	492	592	467	529	474	483	545	466	-1
1802	485	430	435	488	420	493	517	478	461	474	...	483	464	648	499	476	475	484	510	500	1
263	479	444	537	505	614	481	531	478	502	480	...	478	465	173	465	468	474	473	548	502	-1

5 rows × 501 columns

Fig 4.0 - First five rows of randomly sampled three subsets of the train data.

Through the random sampling, I created three subsets that are not identical. Along with the whole dataset, they will be used in feature selection to efficiently go through different techniques.

4.Benchmark Models

It is entirely possible to use these data as it is and have machine learn them with existing predictive models with default assumptions. We do not consider any findings or inferences and naively fit the datas to the model so the effectiveness is not at all optimal. Before go through data engineering, let's conduct it and have it as a benchmark where we can measure the effectiveness of our data engineering. Four benchmark models that we are going to use is Logistic Regression, Decision Tree Classifier, K Nearest Neighbors Classifier, and Support Vector Classifier. I will give a brief explanation of each and code snippets first and aggregate the accuracy score of each models.

Logistic Regression

Here is brief explanation of what Logistic Regression is:

Logistic regression is a statistical method for analyzing a dataset in which there are one or more independent variables that determine an outcome. The outcome is measured with a dichotomous variable (in which there are only two possible outcomes).

> **In logistic regression, the dependent variable is binary or dichotomous, i.e. it only contains data coded as 1 (TRUE, success, pregnant, etc.) or 0 (FALSE, failure, non-pregnant, etc.).**

The goal of logistic regression is to find the best fitting (yet biologically reasonable) model to describe the relationship between the dichotomous characteristic of interest (dependent variable = response or outcome variable) and a set of independent (predictor or explanatory) variables. Logistic regression generates the coefficients (and its standard errors and significance levels) of a formula to predict a *logit transformation* of the probability of presence of the characteristic of interest:

$$logit(p) = b_0 + b_1X_1 + b_2X_2 + b_3X_3 + \ldots + b_kX_k$$

It is power model to predict for binary outcomes linear relationship. Obviously, our dataset is not linear in nature, it has 500 features. However, it is not impossible to utilize it in solving this type of problem. There is one parameter that we will fix. It's called C and it's inversely correlated with a figure called alpha in regularization. Alpha is the number that affects beta vectors you see in the equation. High it is, more controlled the beta vectors are. We do not want any weights or control to be in effect in our benchmark model. Therefore, we will use high number of C, so the alpha is low. After grid searching on logistic regression

with raw data, I selected 10000 as the C value in our parameter.

```
lr_steps = (
    ("scaler", StandardScaler()),
    ("logreg", LogisticRegression(C=10000)),
)

lr_pipe = Pipeline(lr_steps)

sample_1_lr_fit = lr_pipe.fit(sample_1_X, sample_1_y)
sample_2_lr_fit = lr_pipe.fit(sample_2_X, sample_2_y)
sample_3_lr_fit = lr_pipe.fit(sample_3_X, sample_3_y)
```

Decision Tree Classifier

Decision Tree Classifier is a simple and widely used classification technique. It applies a straightforward idea to solve the classification problem. Decision Tree Classifier poses a series of carefully crafted questions about the attributes of the test record. Each time time it receive an answer, a follow-up question is asked until a conclusion about the class label of the record is reached.

The simple way to describe the steps:

1. Place the best attribute of the dataset at the root of the tree.
2. Split the training set into subsets. Subsets should be made in such a way that each subset contains data with the same value for an attribute.
3. Repeat step 1 and step 2 on each subset until you find leaf nodes in all the branches of the tree.

```
tree_steps = (
    ("scaler", StandardScaler()),
    ("tree", DecisionTreeClassifier()),
)

tree_pipe = Pipeline(tree_steps)

sample_1_tree_fit = tree_pipe.fit(sample_1_X, sample_1_y)
sample_2_tree_fit = tree_pipe.fit(sample_2_X, sample_2_y)
sample_3_tree_fit = tree_pipe.fit(sample_3_X, sample_3_y)
```

K Nearest Neighbors Classifier

As the name infers, this model looks for k number of neighboring points for newly unseen points to classify. In the classification setting, the K-nearest neighbor algorithm essentially boils down to forming a majority vote between the K most similar instances to a given

"unseen" observation. Similarity is defined according to a distance metric between two data points. A popular choice is the Euclidean distance given by:

$$d(x, x') = \sqrt{(x_1 - x_1')^2 + (x_2 - x_2')^2 + \ldots + (x_n - x_n')^2}$$

Just as K nearest neighbor is easy to understand and implement, it works just as easily with multiclass data sets whereas other algorithms are hard coded for the binary setting. Moreover the non-parametric nature of KNN gives it an edge in certain settings where the data may be highly "unusual." This might be a good reason to predict with KNN for this dataset. However, KNN can be less powerful when dealing with multidimensional datas. The accuracy of KNN can be severely degraded with high-dimension data because there is little difference between the nearest and farthest neighbor.

```
knn_steps = (
    ("scaler", StandardScaler()),
    ("knn", KNeighborsClassifier())
)

knn_pipe = Pipeline(knn_steps)

sample_1_knn_fit = knn_pipe.fit(sample_1_X, sample_1_y)
sample_2_knn_fit = knn_pipe.fit(sample_2_X, sample_2_y)
sample_3_knn_fit = knn_pipe.fit(sample_3_X, sample_3_y)
```

Support Vector Classifier

"Support Vector Machine" (SVM) is a supervised machine learning algorithm which can be used for both classification or regression challenges. However, it is mostly used in classification problems. In this algorithm, we plot each data item as a point in n-dimensional space (where n is number of features you have) with the value of each feature being the value of a particular coordinate. Then, we perform classification by finding the hyperplane that differentiate the two classes very well. Support Vectors are simply the coordinates of individual observation. Support Vector Machine is a frontier which best segregates the two classes (hyperplane/ line).

Support Vector Classifier has a parameter called "kernel." Here, there are "rbf" and "poly" parameter that is useful for data with a high dimensions. While this lets SVM be a strong model for high dimensions, but could also be a weak model when we are dealing a lot of

noise in data. Support Vector Classifier also utilizes alpha, with the same reason of minimal regularization, we will set a high C of 10000 here as well.

```
svc_steps = (
    ("scaler", StandardScaler()),
    ("svc", SVC(C=10000))
)

svc_pipe = Pipeline(svc_steps)

sample_1_svc_fit = svc_pipe.fit(sample_1_X, sample_1_y)
sample_2_svc_fit = svc_pipe.fit(sample_2_X, sample_2_y)
sample_3_svc_fit = svc_pipe.fit(sample_3_X, sample_3_y)
```

Scores

Here are the accuracy scores for each benchmark model.

	Train Data	Test Data
Logistic Regression	0.7425	1.0
Decision Tree	1.0	1.0
K Nearest Neighbors	0.7289	0.729
Support Vector Classifier	1.0	1.0

As it was noted, SVC performs well when it's dealing with higher dimension. Knn and Logistic Regression showed poor performance with sampled datas as it was expected in their linear nature. As decision tree's processing algorithm does contain a bit of feature selecting, it did perform the second best. Here, we do see many test score and train score of 1.0. This potentially mean that our model is overfitting. Meaning that our model is designed to only follow the plotting pattern but not exacting classifying with information from features. Testing data can have better performance than train data by coincidence but that margin cannot be too large. Like the logistic regression score we got, we can't conclude that the testing score is good since the model can't perform better when dealing with the unseen data since their prediction is based on the train data. We need to approach the

testing score with a grain of salt when it's too high. We will keep the benchmark scores for comparison later.

5. Feature Engineering/Selecting

We will conduct various feature selection methods and through multi-level process, we will pick out features that deemed to be important and informative for our predictions.

Select K Best feature selection with f_classif scoring

In simple words, this method eliminated all features but K number of features. This is from sklearn, "Univariate feature selection works by selecting the best features based on univariate statistical tests. It can be seen as a preprocessing step to an estimator. Scikit-learn exposes feature selection routines as objects that implement the transform method. SelectKBest removes all but the k highest scoring features. The methods based on F-test estimate the degree of linear dependency between two random variables."

This method will generate a "pvalue" for each features. In general the p-value indicates how probable a given outcome or a more extreme outcome is under the null hypothesis. In this case of feature selection, the null hypothesis is something like this feature contains no information about the prediction target, where no information is to be interpreted in the sense of the scoring method. In short, the p-value of a feature selection score indicates the probability that this score or a higher score would be obtained if this variable showed no interaction with the target.

I will only be collecting pvalue less than 0.05 to create list of column that is deemed to be indicative through this process.

	0	1	2	3	4	5	6	7	8	9	...	16	17	18	19	20	21	22	23	24	25
sample1	35	49	53	55	64	129	141	152	155	168	...	302	316	317	336	352	354	365	373	380	475
sample2	35	49	53	55	64	129	141	152	155	168	...	302	316	317	336	352	354	365	373	380	475
sample3	35	49	53	55	64	129	141	152	155	168	...	302	316	317	336	352	354	365	373	380	475

Fig 5.0 - Features selected through Select K Best

The results are 26 features that are listed above. For spacing reason, they are not all shown but all across the sample, same features were selected to be informative. I will keep them and use to tally up the results at the end of this process.

Lasso penalty (l1) with Logistic Regression CV

Parameters to be used:

- l1 for Lasso penalty
- 100 for number of different regularization strenghts.
- 10 for number of cross-validation folds
- 'liblinear' for solver as it is required for the Lasso penalty

Lasso penalty is a metric that is used to regularize the features. They are used in large data set with many features to reduce the weight of unimportant features. Lasso Regression performs L1 regularization, i.e. adds penalty equivalent to absolute value of the magnitude of coefficients.

I will be using top 30 of the features with the high coefficients by ranking. What does a large coefficient signify? It means that we're putting a lot of emphasis on that feature, i.e. the particular feature is a good predictor for the outcome. When it becomes too large, the algorithm starts modelling intricate relations to estimate the output and ends up overfitting to the particular training data. Since the data is organized by the ranks, it is not sorted.

	0	1	2	3	4	5	6	7	8	9	...	20	21	22	23	24	25	26	27	28	29
sample1	49	302	373	264	80	61	243	278	295	421	...	6	415	172	129	133	55	102	383	333	434
sample2	49	302	373	264	80	61	243	278	295	421	...	6	415	172	129	133	55	102	383	333	434
sample3	49	302	373	264	80	61	243	278	295	421	...	6	415	172	129	133	55	102	383	333	434

3 rows × 30 columns

Figure 5.1 - Features selected through Lasso penalty method

Select from model using Logistic Regression

Select from model method in sklearn utilizes a model as an estimator. With the estimator, it creates a threshold that is innate in the model. In this step, it will examine the coefficients of each feature to the target and eliminate all the features that is below the mean.

	0	1	2	3	4	5	6	7	8	9	...	206	207	208	209	210	211	212	213	214	215
sample1	4	5	6	10	12	13	16	18	21	25	...	483	486	487	492	494	495	496	497	498	499
sample2	4	5	6	10	12	13	16	18	21	25	...	483	486	487	492	494	495	496	497	498	499
sample3	4	5	6	10	12	13	16	18	21	25	...	483	486	487	492	494	495	496	497	498	499

3 rows × 216 columns

Figure 5.2 - Features selected through Select from model with Logistic Regression

We are ended with 216 features. Although quite large in output, we are going to narrow it down with comparison with other methods. It can signify that the ones that are not in this list is really nothing indicative.

Select From Model using Decision Tree Classifier

With Decision Tree as an estimator, it will conduct feature selection using mean as a default threshold of feature importance for each features.

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
sample1	22	55	64	83	102	146	159	170	184	193	197	223	256	302	306	355	365	398	408	415
sample2	22	55	64	83	102	146	159	170	184	193	197	223	256	302	306	355	365	398	408	415
sample3	22	55	64	83	102	146	159	170	184	193	197	223	256	302	306	355	365	398	408	415

Fig 5.3 - Features selected through Select from model with Decision Tree

Select From Model using Support Vector Classifier

Select from model is useful feature that allows us to pick out the feature according to its coefficients or feature importance that is built in in the estimator of choice. Support Vector Classifier will have to be a linear in order for us to set a threshold and eliminated features.

	0	1	2	3	4	5	6	7	8	9	...	204	205	206	207	208	209	210	211	212	213
sample1	0	3	4	5	6	10	12	16	17	18	...	475	477	481	486	491	492	493	495	496	499
sample2	0	3	4	5	6	10	12	16	17	18	...	475	477	481	486	491	492	493	495	496	499
sample3	0	3	4	5	6	10	12	16	17	18	...	475	477	481	486	491	492	493	495	496	499

3 rows × 214 columns

Fig 5.4 - Features selected through Select from model with Support Vector Classifier

Select From Model using K Nearest Neighbor Classifier

	0	1	2	3	4	5	6	7	8	9	...	204	205	206	207	208	209	210	211	212	213
sample1	0	3	4	5	6	10	12	16	17	18	...	475	477	481	486	491	492	493	495	496	499
sample2	0	3	4	5	6	10	12	16	17	18	...	475	477	481	486	491	492	493	495	496	499
sample3	0	3	4	5	6	10	12	16	17	18	...	475	477	481	486	491	492	493	495	496	499

3 rows × 214 columns

Fig 5.5 - Features selected through Select from model with K Nearest Neighbor Classifier

Recursive Feature Elimination using Logistic Regression

Although it is very computationally heavy and takes a long time to run, recursive feature elimination is a powerful feature elimination tool. In the process, we try to use a subset of features and train a model using them. Based on the inferences that we draw from the previous model, we decide to add or remove features from your subset. The problem is essentially reduced to a search problem. These methods are usually computationally very expensive.

	0	1	2	3	4	5	6	7	8	9	...	20	21	22	23	24	25	26	27	28	29
sample1	31	34	49	55	61	64	85	129	132	133	...	333	348	351	365	373	380	390	467	487	492
sample2	31	34	49	55	61	64	85	129	132	133	...	333	348	351	365	373	380	390	467	487	492
sample3	31	34	49	55	61	64	85	129	132	133	...	333	348	351	365	373	380	390	467	487	492

3 rows × 30 columns

Fig 5.6 - Features selected through Recursive Feature Elimination using Logistic Regression

Final Count

We have conducted seven different methods of feature selection. Through seven different methods on these samples, we were able to obtain features with more indicative importance to the target. Each methods gave different kinds and number of features. I will go ahead and count the frequency of feature that were selected in this step. One thing to take note is that although not all samples have same data points, the feature selections

came out identical all across the samples. Let's add selected features together and count them. We will only use features that got selected at least 5 times out of 7 methods.

	365	302	55	49	373	64	380	170	129	316	...	61	184	133	155	295	415	34	31	333	264
count	7	7	7	6	6	6	6	6	6	6	...	5	5	5	5	5	5	5	5	5	5

1 rows × 23 columns

```
Int64Index([365, 302,  55,  49, 373,  64, 380, 170, 129, 316, 141, 102, 138,
             61, 184, 133, 155, 295, 415,  34,  31, 333, 264],
           dtype='int64')
```

Fig 5.7 - Final Features that were selected at least five times

Dimension Reduction Through Principal Components Analysis

Principal component analysis (PCA) is a technique that transforms a dataset of many features into principal components that "summarize" the variance that underlies the data. Each principal component is calculated by finding the linear combination of features that maximizes variance, while also ensuring zero correlation with the previously calculated principal components. It is one of the most common dimensionality reduction techniques. Turn our features into principal components and let's examine how much of each components explains the whole variance of the data. PCA is used if there are too many features or if observation/feature ratio is poor. Also, PCA is a potentially good option if there are a lot of highly correlated variables in your dataset. Unfortunately, PCA makes models a lot harder to interpret since our feature tells us what it is that impacts our target but principal component is principal component, it is hard to interpret in our everyday concept.

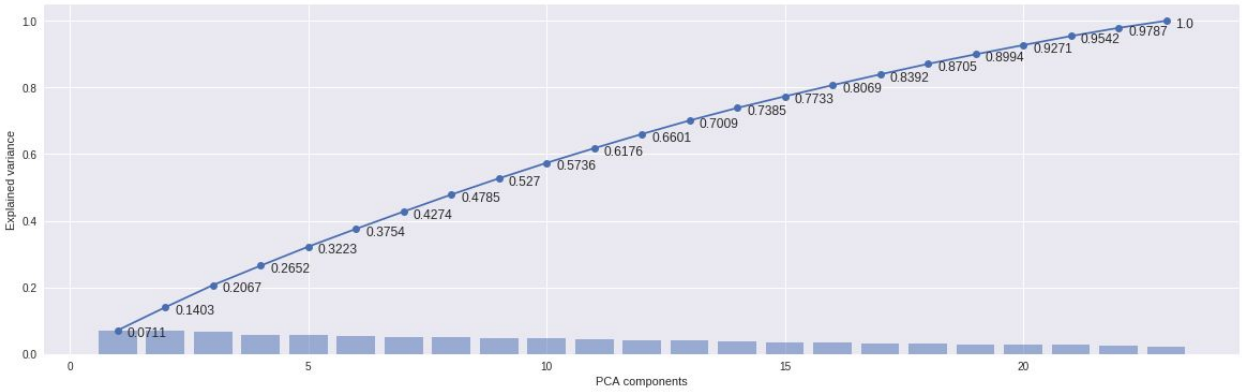


Fig 6.0 - Principal Components Analysis variance ratio for each components when turned into 23 components.

Fig 6.0 shows the graph of each component's ratio of variance relative to the whole data's variance. It is clear that there is very minimal difference in the ratio. There are some different in the first three components (them being slightly higher than rest). This makes sense since our feature selection showed us that three of the features contains strong indications. However, even in that the difference is negligible. Meaning that each component in our data is telling about the same information on our data. Without setting fixed number of components as the parameter, we will utilize PCA in our pipeline for model building.

6. Model Building

With feature derived from previous step, we will build a model using pipeline and gridsearch CV. Grid Search will find the optimal parameters that go into our pipeline. We will again use the models that we tested in the benchmark step and compare the results.

Model Fit: Logistic Regression

Unlike the Benchmark, we will look for C that will give us the best score.

```
lr_grid_steps = (
    ("scaler", StandardScaler()),
    ("pca", PCA()),
    ("logreg", LogisticRegression()),
)

lr_grid_pipe = Pipeline(lr_grid_steps)

lr_param_grid = {
    "pca__n_components" : range(1,11),
    "logreg__penalty": ["l1", "l2"],
    "logreg__C": np.logspace(-4, 4, 10)
}


lr_grid = GridSearchCV(lr_grid_pipe, param_grid = lr_param_grid)


sample_1_lr_grid_fit = lr_grid.fit(X_train_sample_1, sample_1_y)
sample_2_lr_grid_fit = lr_grid.fit(X_train_sample_2, sample_2_y)
sample_3_lr_grid_fit = lr_grid.fit(X_train_sample_3, sample_3_y)
```

This is the best parameter that we found

```
{'logreg__C': 2.7825594022071258,
 'logreg__penalty': 'l1',
 'pca__n_components': 10}
```

Model Fit: Decision Tree

```
tree_steps = (
    ("scaler", StandardScaler()),
    ("pca", PCA()),
    ("tree", DecisionTreeClassifier()),
)

tree_pipe = Pipeline(tree_steps)


tree_param_grid = {
    "pca__n_components" : range(1,11),
    "tree__criterion": ["gini", "entropy"],
}

tree_grid = GridSearchCV(tree_pipe, param_grid = tree_param_grid)




sample_1_tree_grid_fit = tree_grid.fit(X_train_sample_1, sample_1_y)
sample_2_tree_grid_fit = tree_grid.fit(X_train_sample_2, sample_2_y)
sample_3_tree_grid_fit = tree_grid.fit(X_train_sample_3, sample_3_y)
```

The best parameters are:

```
{'pca__n_components': 9, 'tree__criterion': 'gini'}
```

Model fit: K Nearest Neighbor Classifier

```
knn_steps = (
    ("scaler", StandardScaler()),
    ('pca', PCA()),
    ("knn", KNeighborsClassifier())
)

knn_pipe = Pipeline(knn_steps)

knn_param_grid ={
    "pca__n_components" : range(1,11),
    "knn__n_neighbors" : range(3,6),
    "knn__weights": ['uniform','distance']
}

knn_grid = GridSearchCV(knn_pipe, param_grid = knn_param_grid)


sample_1_knn_grid_fit = knn_grid.fit(X_train_sample_1, sample_1_y)
sample_2_knn_grid_fit = knn_grid.fit(X_train_sample_2, sample_2_y)
sample_3_knn_grid_fit = knn_grid.fit(X_train_sample_3, sample_3_y)
```

```
{'knn__n_neighbors': 4, 'knn__weights': 'uniform', 'pca__n_components': 10}
```

Model Fit: Support Vector Classifier

```
svc_steps = (
    ("scaler", StandardScaler()),
    ('pca', PCA()),
    ("svc", SVC())
)

svc_pipe = Pipeline(svc_steps)

svc_param_grid ={
    "pca__n_components":range(5,11),
    "svc__C" : np.logspace(-4, 4, 5),
    "svc__kernel" : ['linear','rbf','poly']
}

svc_grid = GridSearchCV(svc_pipe, param_grid = svc_param_grid)

sample_1_svc_grid_fit = svc_grid.fit(X_train_sample_1, sample_1_y)
sample_2_svc_grid_fit = svc_grid.fit(X_train_sample_2, sample_2_y)
sample_3_svc_grid_fit = svc_grid.fit(X_train_sample_3, sample_3_y)
```

```
{'pca__n_components': 10, 'svc__C': 1.0, 'svc__kernel': 'linear'}
```

Final Scores

	Processed Train Data	Processed Test Data	Unprocess Train Data	Unprocessed Test Data
Logistic Regression	0.558	0.6016	0.7425	1.0
Decision Tree	0.8629	0.875	1.0	1.0
K Near Neighbor Classifier	0.9769	0.9983	0.7289	0.729
Support Vector Classifier	0.62	0.6483	1.0	1.0

Let's start with Logistic Regression. Although through our process the score decreased substantially, this might indicate that Logistic Regression's unprocessed test data score should not be understood as it stands. As I mentioned briefly, logistic regression is a great tool for linearly correlated data and it's also labeled as a linear model. With understanding the data prior to building out a model, we saw that with 500 features, the data is not linear

by nature. If we were to jump into model fitting without such knowledge, one could have inferred that Logistic Regression would yield them a great prediction. However, a person with familiarity with the data and obtain some understanding by exploring it beforehand, would understand right off the bat that just naive Logistic Regression will not a good job by the nature. By processing the data and have machine understand the data without noisy, it was clear by the result that this can only result in a slight better than random guessing.

Decision tree on the other hand, still gave a reasonably good score for both train and test datas after preprocessing. Same concept applies here as well. When a model is over-fitting, meaning that this model is following patterns of train data and drawing a function that would cover each and every data point that might contain error or outlier. With data processing Decision Tree is no longer building a model just to have the data points fit into the function. Rather it is now working as a model that understands the whole data's classification indicatives and drawing reasonable predictions.

KNN Classifier did improve considerably after preprocessing. Considering its relative high performance in the unprocessed data could mean that the KNN classifier could be a good model to work with data with such a structure. However, it is understood that KNN isn't optimal for data with high dimension. That could also be dependant on data structure since in the 3d space like madelon, the distance difference still could be large between near and further data points. Although the score is high and there's positive potential, test data did score a bit better than train data and its test data score is near 1.0. With this scores, this model should be utilize with some caution.

Support Vector Classifier didn't perform very well in this dataset. Although we were able to use other kernel option to optimize at the higher dimension, processed data showed us that we could avoid over fitting by feature selections but not a good predictor for this type of problem.

In summation, it was showed that KNN Classifier is the better model to predict out of the four models we demonstrated. It is more important to note that, our sampling gave us consistent results when we were conducting feature selection. Also, the score that one can obtain by learning unprocessed data can create a tremendous misleading interpretation.

Data engineering not only improved the model when applied appropriately but also help sorting out inapproriate model with unreasonably and unrealistically high accuracy scores.

7. Repeat step 5 & 6 on larger dataset

For demonstration of same process with larger dataset, we will use instructor's data set that I mentioned above. We will conduct similar steps to eliminate noisy features and fit them samples into the model. Unlike the Madelon data, here we are dealing with thousand features. Not only we will utilize sampled datas for feature selection, we will use them to score our models as well. We will take the mean of three samples' scores. It is way too heavy duty to use the whole data to go through all the computations.

Here are basic information on the samples:

```
Int64Index: 10001 entries, 9994 to 19994
Columns: 1001 entries, feat_000 to target
dtypes: float64(1000), int64(1)
memory usage: 76.5 MB
<class 'pandas.core.frame.DataFrame'>
Int64Index: 10001 entries, 7446 to 17446
Columns: 1001 entries, feat_000 to target
dtypes: float64(1000), int64(1)
memory usage: 76.5 MB
<class 'pandas.core.frame.DataFrame'>
Int64Index: 10001 entries, 9024 to 19024
Columns: 1001 entries, feat_000 to target
dtypes: float64(1000), int64(1)
memory usage: 76.5 MB
```

I have selected 10001 entries per sample, which is about 5% of the whole dataset. Even at this amount, the computation load is quite heavy. Therefore, we need to tweak way we compute in our process in order to derive meaningful data processing.

Let's dive right into benchmarks

Benchmark Scores

We are going to use the same four models.

	Mean sampled Train Data	Mean sampled Test Data
Logistic Regression	1.0	1.0
Decision Tree	1.0	1.0
K Nearest Neighbors Classifier	0.714	0.8259
Support Vector Classifier	1.0	1.0

Feature Selection with Select K Best

This time, we set the pvalue limit to 0.03 instead for stricter selection.

	0	1	2	3	4	5	6	7	8	9	...	31	32	33	34	35	36	37	38	39	40
sample1	20	175	177	257	269	308	315	336	339	341	...	808	809	815	825	829	867	907	916	920	958
sample2	20	175	177	257	269	308	315	336	339	341	...	808	809	815	825	829	867	907	916	920	958
sample3	20	175	177	257	269	308	315	336	339	341	...	808	809	815	825	829	867	907	916	920	958

3 rows × 41 columns

Logistic Regression CV with Lasso penalty (l1)

Due to heavy computation, we will use 10 for number of Cs to look for, and 5 fold cross validation.

	0	1	2	3	4	5	6	7	8	9	...	20	21	22	23	24	25	26	27	28	29
sample1	920	681	269	808	666	671	670	669	668	667	...	674	675	676	677	678	679	680	682	683	684
sample2	920	681	269	808	666	671	670	669	668	667	...	674	675	676	677	678	679	680	682	683	684
sample3	920	681	269	808	666	671	670	669	668	667	...	674	675	676	677	678	679	680	682	683	684

Select from model using Logistic Regression

Threshold is now 2.5 times the mean for stricter selection.

	0	1	2	3	4	5	6	7	8	9	...	45	46	47	48	49	50	51	52	53	54
sample1	12	20	34	94	117	135	175	177	257	268	...	891	897	907	920	927	949	951	958	966	987
sample2	12	20	34	94	117	135	175	177	257	268	...	891	897	907	920	927	949	951	958	966	987
sample3	12	20	34	94	117	135	175	177	257	268	...	891	897	907	920	927	949	951	958	966	987

3 rows × 55 columns

Select From Model using Decision Tree Classifier

Threshold is now 3 times the mean for stricter selection.

	0	1	2	3	4	5	6	7	8	9	...	39	40	41	42	43	44	45	46	47	48
sample1	26	36	40	58	88	129	208	220	231	257	...	808	824	829	847	854	867	904	920	954	956
sample2	26	36	40	58	88	129	208	220	231	257	...	808	824	829	847	854	867	904	920	954	956
sample3	26	36	40	58	88	129	208	220	231	257	...	808	824	829	847	854	867	904	920	954	956

3 rows × 49 columns

Select from Model using Support Vector Classifier

Threshold is now 2.5 times the mean for stricter selection.

	0	1	2	3	4	5	6	7	8	9	...	40	41	42	43	44	45	46	47	48	49
sample1	12	66	76	117	135	177	195	257	268	269	...	829	867	875	891	897	916	920	951	958	987
sample2	12	66	76	117	135	177	195	257	268	269	...	829	867	875	891	897	916	920	951	958	987
sample3	12	66	76	117	135	177	195	257	268	269	...	829	867	875	891	897	916	920	951	958	987

3 rows × 50 columns

Select from Model using K Nearest Neighbor Classifier

Threshold is now 2 times the mean

	0	1	2	3	4	5	6	7	8	9	...	40	41	42	43	44	45	46	47	48	49
sample1	12	66	76	117	135	177	195	257	268	269	...	829	867	875	891	897	916	920	951	958	987
sample2	12	66	76	117	135	177	195	257	268	269	...	829	867	875	891	897	916	920	951	958	987
sample3	12	66	76	117	135	177	195	257	268	269	...	829	867	875	891	897	916	920	951	958	987

Recursive Feature Elimination using Logistic Regression

We would loved to run Recursive feature elimination method where we eliminate one feature at a time, but it would simply take too long to run. Therefore, we will eliminate in increments of 5 instead to faster computation. We are only left with one column (feat_681).

Final Count

	257	526	808	504	308	341	681	269	920	829	...	177	809	958	396	445	867	793	728	897	891
count	5	5	5	5	5	5	5	5	5	5	...	4	4	4	4	4	4	4	4	3	3

1 rows × 25 columns

We are down to these 23 features.

Dimension reduction through Principal Component Analysis

Again, we are going to utilize PCA to see if there's any number of components that explains the whole variance well. This time we got to eliminate a lot more features. Let's graph the variance ration explained by each components by setting the data dimension to 23.

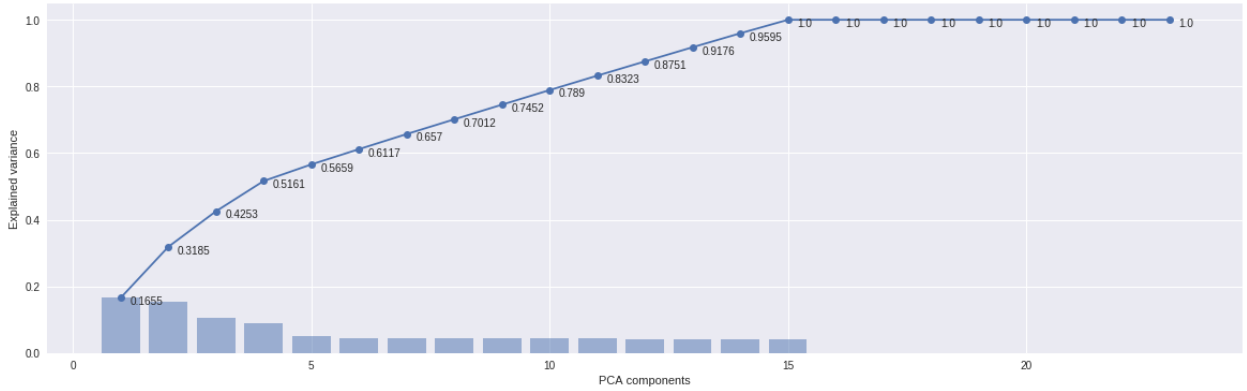


Fig 7.0 - PCA variance ratio on sample 1

Fig 7.1- PCA variance ratio on sample 2

Fig 7.0 - PCA variance ratio on sample 3

In this dataset, it seems that there are about five components that are most indicative of the whole features. The amount of variance explained diminishes after the fifth components and stays more of the same until 15th components. This information is not distinctive enough for us to conclude that there are five features we would want to use. Instead, the data's dimension can be reduced to five components to explain the data well enough. Its variance is around 60% of the whole dataset. Since this is very computationally heavy work, having a resource that can lead us to create a efficient range for our grid search can save a lot of time. Let's start Grid Search and get the scores.

Model Building Logistic Regression

```
lr_grid_steps = (
    ("scaler", StandardScaler()),
    ("pca", PCA()),
    ("logreg", LogisticRegression()),
)

lr_grid_pipe = Pipeline(lr_grid_steps)

lr_param_grid = {
    "pca__n_components" : range(1,7),
    "logreg__penalty": ["l1", "l2"],
    "logreg__C": np.logspace(-4, 4, 10)
}


lr_grid = GridSearchCV(lr_grid_pipe, param_grid = lr_param_grid)
```

Best Parameters:

```
{'logreg__C': 0.046415888336127774, 'logreg__penalty': 'l1'}
```

Model Building Decision Tree

```
tree_steps = (
    ("scaler", StandardScaler()),
    ("pca", PCA(n_components=5)),
    ("tree", DecisionTreeClassifier()),
)

tree_pipe = Pipeline(tree_steps)


tree_param_grid = {
    "tree__criterion": ["gini", "entropy"],
    "tree__max_depth": [None, 3, 5]

}

tree_grid = GridSearchCV(tree_pipe, param_grid = tree_param_grid)
```

Best Parameters:

```
{'tree__criterion': 'gini', 'tree__max_depth': 5}
```

Model Building K Nearest Neighbor Classifier

```
knn_steps = (
    ("scaler", StandardScaler()),
    ('pca', PCA(n_components=5)),
    ("knn", KNeighborsClassifier())
)

knn_pipe = Pipeline(knn_steps)

knn_param_grid ={
    "knn__n_neighbors" : range(3,6),
    "knn__weights": ['uniform','distance']
}

knn_grid = GridSearchCV(knn_pipe, param_grid = knn_param_grid)
```

Best parameters:

```
{'knn__n_neighbors': 5, 'knn__weights': 'distance'}
```

Model Building Support Vector Classifier

```
svc_steps = (
    ("scaler", StandardScaler()),
    ('pca', PCA(n_components=5)),
    ("svc", SVC())
)

svc_pipe = Pipeline(svc_steps)

svc_param_grid ={
    "svc__C" : np.logspace(-4, 4, 3)
}

svc_grid = GridSearchCV(svc_pipe, param_grid = svc_param_grid)
```

Best parameter: {'svc__C': 1.0}

Final Scores

	Mean sampled processed Train Data	Mean sampled processed Test Data	Mean sampled unprocessed Train Data	Mean sampled unprocessed Test Data
Logistic Regression	0.61	0.61	1.0	1.0
Decision Tree	0.63	0.65	1.0	1.0
K Nearest Neighbor Classifier	0.756	0.85	0.714	0.8259
Support Vector Classifier	0.7435	0.77	1.0	1.0

This time, our preprocessing yielded better performance for KNN model again. However, we found that while Decision Tree had suffered quite a loss in performance metric, Support Vector Classifier did better than in our previous dataset. Obviously, the data's structure is

quite different and amount of noisy feature also increased substantially in this dataset. In this process of data engineering, we found that the KNN classifier does still perform better than relative to three other models. We have selected and used only five components in dealing with this dataset. It might have erased other meaningful features but with limited resources and memory, We were able to derive better scoring for accuracy for KNN while eliminating over fitting and dubious perfect scores for other models. The findings we demonstrate is the same. It is important to run through data engineering since the raw data will give us misleading information if we don't process them properly.