Project 3:

Feature Selection and Classification

Purpose:

The purpose of this exploration is to demonstrate a capacity to identify relevant features in a dataset, using machine learning. This will be done by analyzing Madelon, a synthetic dataset.

Data Description:

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). A number of distractor features, called 'probes', were added, having no predictive power. The order of the features and patterns were randomized.
Madelon has a total of: 500 features, of which 5 are informative, 15 are redundant and 480 are probes.

Problem Statement:

To demonstrate the ability to identify relevant features in a dataset, using machine learning, I will analyze two versions of the Madelon dataset. One dataset will be downloaded from the UCI website and the other one will be retrieved from a database, housed on AWS.

Process and methodology:

This will be done by analyzing three subsets of data from the UCI Madelon dataset and three subsets of data from a database Madelon dataset.

The process will involve:

- Step 1: Data acquisition and benchmarking
- Step 2: Identifying salient features
- Step 3: Determining feature importance
- Step 4: Building the final model

Data acquisition:

For this exploration, two types of Madelon datasets were retrieved. One dataset came from the UCI repository and the other one came from a database on AWS.

- UCI Madelon Data description:
  - Contains 500 features and 2000 rows data.
  - The train data has 2000 rows, which I sampled into 3 subsets of 200 rows.
  - The validate data has 600 rows.

- Database Madelon Data description:
  - Contains 1000 features and 200000 rows of data.
  - 15000 rows of data were successfully downloaded from the database.
  - The data was then divided into:
    - 3 train data subsets of 4000 rows.
    - 1 test data subset of 3000.
    - The target y was separated from the data.

Benchmarking:

- Four pipelines were built to perform a naïve fit for each of the following base model classes:
  - logistic regression
  - decision tree classifier
  - support vector classifier
  - k nearest neighbors classifier

To validate our models, I primarily looked at 3 scores.

- Accuracy: is a ratio of correctly predicted observation to the total observations. (TP+TN)/(TP+FP+FN+TN)
- Precision: is the correctly predicted positive observations to the total predicted positive observations. TP/(TP+FP)
- Recall: is the ratio of correctly predicted positive observations to all the observations in the actual class. TP/(TP+FN)

The following dataframe shows the benchmark results for UCI data. As it can be seen, the test accuracy scores are not great. The accuracy scores of all models, for raw and scaled data, are roughly around 50%.

	dataset	model	raw_tr_acc_score	raw_tst_acc_score	raw_precision	raw_recall	sc_tr_acc_score	sc_tst_acc_score	sc_precision	sc_recall
0	Set_1	LogisticRegression	1.000000	0.500000	0.548387	0.472222	1.000000	0.454545	0.500000	0.416667
1	Set_1	SVC	1.000000	0.454545	0.000000	0.000000	1.000000	0.500000	0.565217	0.361111
2	Set_1	DecisionTree	1.000000	0.454545	0.500000	0.472222	1.000000	0.454545	0.500000	0.472222
3	Set_1	KNN	0.783582	0.560606	0.629630	0.472222	0.753731	0.545455	0.615385	0.444444
4	Set_2	LogisticRegression	1.000000	0.560606	0.448276	0.500000	1.000000	0.545455	0.423077	0.423077
5	Set_2	SVC	1.000000	0.606061	0.000000	0.000000	1.000000	0.590909	0.473684	0.346154
6	Set_2	DecisionTree	1.000000	0.575758	0.473684	0.692308	1.000000	0.575758	0.473684	0.692308
7	Set_2	KNN	0.776119	0.606061	0.500000	0.576923	0.686567	0.575758	0.466667	0.538462
8	Set_3	LogisticRegression	1.000000	0.560606	0.562500	0.545455	1.000000	0.484848	0.484848	0.484848
9	Set_3	SVC	1.000000	0.500000	0.500000	1.000000	1.000000	0.530303	0.527778	0.575758
10	Set_3	DecisionTree	1.000000	0.454545	0.457143	0.484848	1.000000	0.454545	0.457143	0.484848
11	Set_3	KNN	0.813433	0.651515	0.656250	0.636364	0.708955	0.515152	0.514286	0.545455

The following dataframe shows the benchmark results for the Database data. As it can be seen, the test accuracy scores are not great. The accuracy scores of all models, for raw and scaled data, is also roughly around 50%.

	dataset	model	raw_tr_acc_score	raw_tst_acc_score	raw_precision	raw_recall	sc_tr_acc_score	sc_tst_acc_score	sc_precision	sc_recall
0	Test_Set_1	LogisticRegression	0.801866	0.543182	0.537984	0.532209	0.801866	0.543182	0.537984	0.532209
1	Test_Set_1	SVC	1.000000	0.623485	0.614815	0.636503	1.000000	0.579545	0.573374	0.581288
2	Test_Set_1	DecisionTree	1.000000	0.595455	0.592187	0.581288	1.000000	0.595455	0.592187	0.581288
3	Test_Set_1	KNN	0.754104	0.599242	0.584131	0.654908	0.718284	0.517424	0.510067	0.582822
4	Test_Set_2	LogisticRegression	0.803731	0.538636	0.570533	0.520744	0.803731	0.538636	0.570533	0.520744
5	Test_Set_2	SVC	1.000000	0.621970	0.648810	0.623748	1.000000	0.582576	0.615265	0.565093
6	Test_Set_2	DecisionTree	1.000000	0.628030	0.647727	0.652361	1.000000	0.628030	0.647727	0.652361
7	Test_Set_2	KNN	0.760448	0.593182	0.616379	0.613734	0.717910	0.563636	0.586742	0.595136
8	Test_Set_3	LogisticRegression	0.795149	0.496970	0.519461	0.502899	0.795149	0.496970	0.519461	0.502899
9	Test_Set_3	SVC	0.998134	0.504545	0.527864	0.494203	0.998134	0.500758	0.524181	0.486957
10	Test_Set_3	DecisionTree	0.998134	0.492424	0.514881	0.501449	0.998134	0.492424	0.514881	0.501449
11	Test_Set_3	KNN	0.684328	0.500758	0.524181	0.486957	0.694030	0.525000	0.548686	0.514493

Goal:

Our goal is to demonstrate that by using principles and tools of machine learning and data science, we can extract the most meaningful features from the dataset and use them to train our final model. This we conjecture will reduce dimensionality, improve processing time and most importantly, improve the accuracy of our predictions.

Feature Selection:

UCI data

To extract the 20 good features from this noisy Madelon dataset, I used slightly different methods, for the UCI and the database data. In comparison, the UCI data was small, 600 rows of data between the 3 subsets. I ran the training data, for each dataset through the following 4 methods of feature selection:

- SelectKBest, with K=20
- SelectPercentile, with percentile=4% (which is 20 for 500 features)
- SelectFromModel: Lasso, with alpha =0.13
- Noise reduction using a DecisionTreeRegressor

After processing the datasets, I ended up with a set of features selected by each dataset. This was determined by taking the union of the features selected by all for methods, for each data set.

Dataset1- selected 42 features:

```
[17, 28, 32, 48, 55, 64, 66, 79, 99, 105, 110, 120, 128,
 138, 153, 178, 213, 236, 241, 281, 286, 296, 318, 326, 335, 336,
 338, 347, 357, 378, 379, 390, 430, 433, 442, 451, 453, 455, 472,
 475, 493, 497]
```

Dataset2 - selected 35 features:

```
[28, 48, 64, 71, 105, 108, 128, 153, 226, 241, 250, 253, 255,
 281, 318, 336, 338, 378, 386, 390, 413, 420, 433, 434, 435, 442,
 448, 451, 453, 455, 472, 475, 486, 493, 496]
```

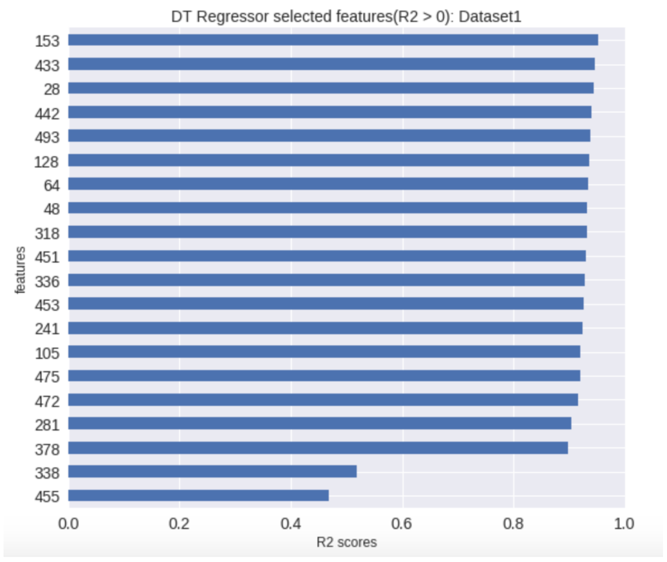
Dataset3 - selected 39 features:

```
[0,   21,  28,  48,  64, 105, 128, 135, 151, 153, 196, 211, 241,
 243, 249, 263, 281, 318, 336, 338, 340, 353, 378, 381, 422, 425,
 429, 433, 436, 442, 443, 447, 451, 453, 455, 472, 475, 478, 493
```

I then found the common features selected from all datasets, by taking their intersection.
The selected 20 good features found were:

```
[28,  48,  64, 105, 128, 153, 241, 281, 318, 336, 338, 378, 433,
 442, 451, 453, 455, 472, 475, 493]
```

Interestingly, these were the exact 20 features selected by the noise reduction method, proposed by Joshua. For this data set, most of its features are noise and uncorrelated to the other features. It makes sense that a method looking at how well a feature can be predicted by the rest of the features the dataset, would easily eliminate the uncorrelated ones.



Database data

To extract the 20 good features from this noisy Madelon dataset, I used a slightly different method for dealing with the database data. In comparison, this data was large, 12,000 rows of data, between the 3 subsets. The dataset has 1000 features, with 20 being good and 5 being informative.

I would like to note that the data in the database, has 200,000 rows of observation. However, I was not successfully able to retrieve more than 3000 rows of data at a time.

- Any data chunk request, greater than 3000 rows, would kill the kernel.
- There were also issues of the database being unresponsive during certain periods of time.
- I understand that that there are engineering constraints to be dealt with, when handling data acquisition and processing.
- However, due to time constraints, I used the 15000 rows of data I was able to retrieve, to make 3 train data subsets of 4000 rows each and 3000 rows of data for the validate set.

I ran these training data subsets, of 4000 rows each, through the following 4 methods of feature selection:

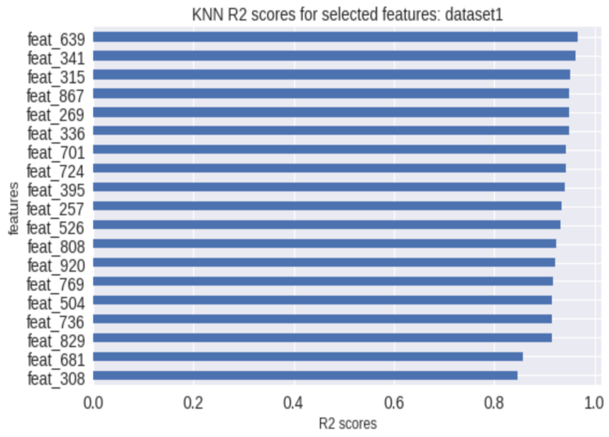
- SelectKBest, with K=20
- SelectPercentile, with percentile=2% (which is 20 for 1000 features)
- SelectFromModel: LassoCV, with threshold='mean'
- SelectFromModel using DecisionTreeRegressor

This is where I did something different. Since the data has 1000 features, I wanted to avoid using the noise reduction, proposed by Joshua just yet. So, I experimentally took the following approach.

Like with UCI data, after processing the datasets, I ended up with a set of features selected by each dataset. This was determined by taking the union of the features selected by all for methods, for each data set. I then took the intersection of these features, to find the common ones selected by all datasets. There were 32 features selected.

```
[67, 134, 257, 269, 274, 308, 315, 330, 336, 341, 360, 395, 431,
 454, 492, 504, 526, 639, 681, 685, 701, 724, 736, 743, 769, 808,
 811, 825, 829, 841, 867, 920]
```

Since 32 features were selected, I wanted to then reduce them to 20 good features. So here is where I used the noise reduction method, proposed by Joshua. Since I had used the DecisionTreeRegressor as a SelectFromModel feature selector, I used KNeighborsRegressor for the noise reduction method this time. From this noise reduction, I was able to extract 19 good features.



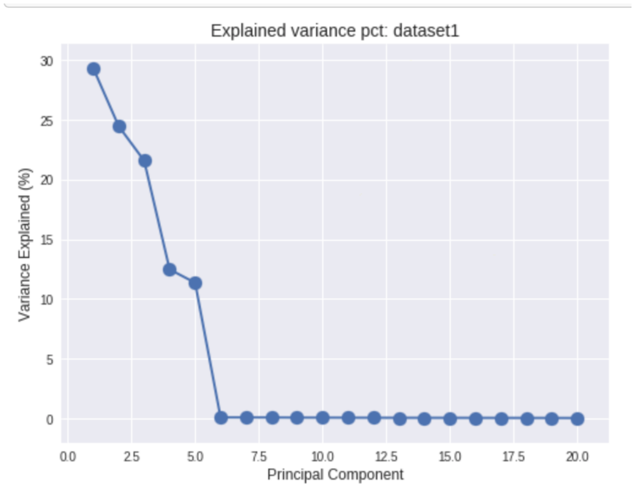
The following final 19 good features were extracted:

[feat_341, feat_639, feat_269, feat_315, feat_867, feat_336, feat_395, feat_701, feat_724
 feat_526, feat_257, feat_769, feat_736, feat_808, feat_920, feat_828, feat_504, feat_681
 feat_308]

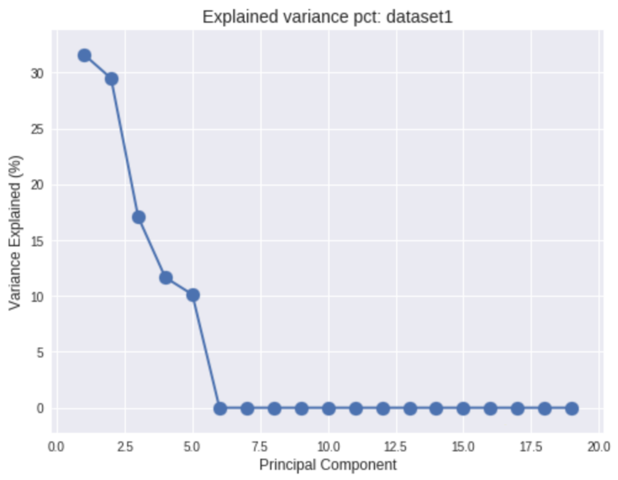
Feature Importance:

After determining the good features, I used principal component analysis (PCA) to transform them into a space, where most of the variance in the data could be captured within a fewer number of components.

PCA on UCI data's 20 good features I extracted, from feature analysis:



PCA on Database data's 19 good features I extracted, from feature analysis:

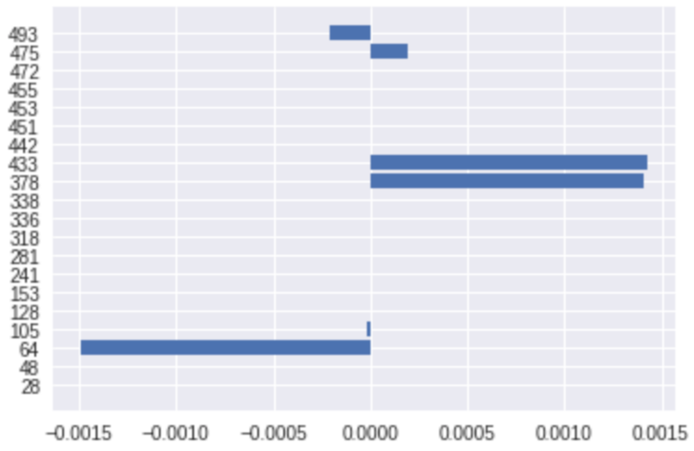


As seen in the PCA graphs above, when the data is transformed into new principal components, there are still the same number of components, as features. However, only a few are informative.

- The UCI PCA graph shows that the 20 features extracted were transformed, but 5 components explain the variability in the data.
- Similarly, we see that the Database PCA graph shows that the 19 features extracted were transformed, but 5 components explain the variability in the data.

This is exactly what I expected, since the Madelon dataset is known to have only 5 informative features.

Since Lasso's regularization method, naturally zeroes out certain low importance features, I was curious to see what would happen when I ran my data through it. The results are shown below.

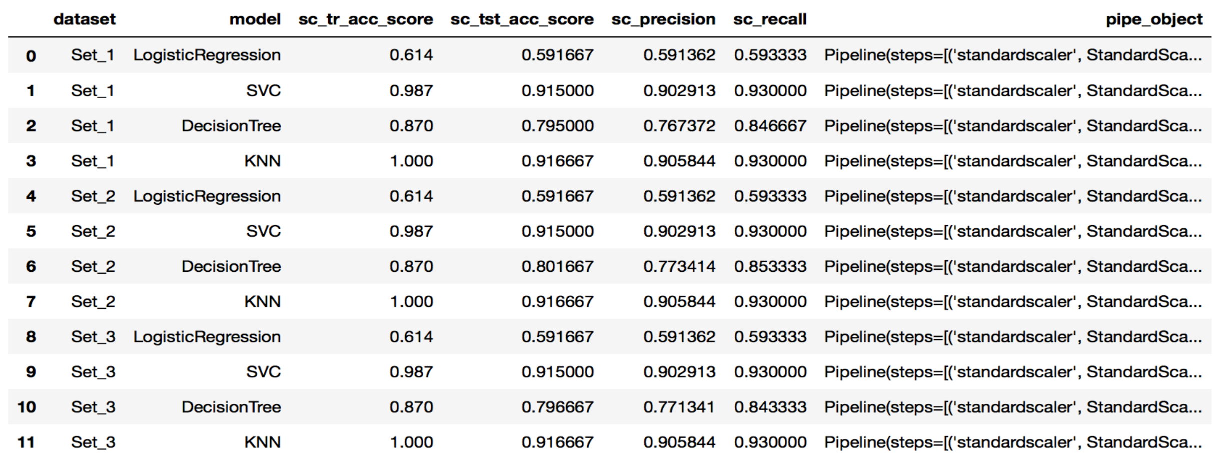


As it can be seen, Lasso picked up on the following 5 most informative features:
[64, 378, 433, 475, 493]. However, when I ran the gridsearch on data with only these 5 features selected, I got lower results than with the PCA data. So, for the tuning of the hyper parameters, described in the following section, I used PCA to transform the data before doing a gridsearch.

Model Selection:

After doing PCA, I did a 'gridsearch' on the 4 models I am using, to tune their hyper parameters, to find the best model to predict our data.

To train the models for the UCI dataset, I used its entire training data of 2000 rows and its validation data of 600 rows. See the results from the gridsearch below.



	dataset	model	sc_tr_acc_score	sc_tst_acc_score	sc_precision	sc_recall	pipe_object
0	Set_1	LogisticRegression	0.614	0.591667	0.591362	0.593333	Pipeline(steps=[('standardscaler', StandardSca...
1	Set_1	SVC	0.987	0.915000	0.902913	0.930000	Pipeline(steps=[('standardscaler', StandardSca...
2	Set_1	DecisionTree	0.870	0.795000	0.767372	0.846667	Pipeline(steps=[('standardscaler', StandardSca...
3	Set_1	KNN	1.000	0.916667	0.905844	0.930000	Pipeline(steps=[('standardscaler', StandardSca...
4	Set_2	LogisticRegression	0.614	0.591667	0.591362	0.593333	Pipeline(steps=[('standardscaler', StandardSca...
5	Set_2	SVC	0.987	0.915000	0.902913	0.930000	Pipeline(steps=[('standardscaler', StandardSca...
6	Set_2	DecisionTree	0.870	0.801667	0.773414	0.853333	Pipeline(steps=[('standardscaler', StandardSca...
7	Set_2	KNN	1.000	0.916667	0.905844	0.930000	Pipeline(steps=[('standardscaler', StandardSca...
8	Set_3	LogisticRegression	0.614	0.591667	0.591362	0.593333	Pipeline(steps=[('standardscaler', StandardSca...
9	Set_3	SVC	0.987	0.915000	0.902913	0.930000	Pipeline(steps=[('standardscaler', StandardSca...
10	Set_3	DecisionTree	0.870	0.796667	0.771341	0.843333	Pipeline(steps=[('standardscaler', StandardSca...
11	Set_3	KNN	1.000	0.916667	0.905844	0.930000	Pipeline(steps=[('standardscaler', StandardSca...

As seen from the table above, KNN and SVC performed the best for the UCI data.

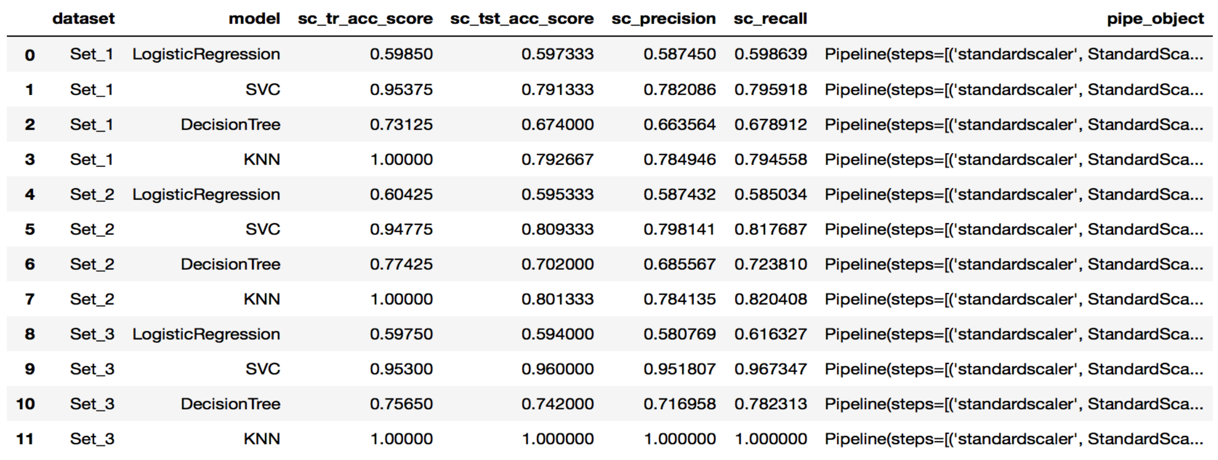
KNN 'gridsearch' results for validation data:

- Accuracy: 0.916667
- Precision: 0.905844
- Recall: 0.93

SVC 'gridsearch' results for validation data:

- Accuracy: 0.9150
- Precision: 0.9029
- Recall: 0.93

To train the models for the Database data, I used the 3 data subsets of 400 rows each. I used a validation data of 1500 rows. See the results from the gridsearch below.



	dataset	model	sc_tr_acc_score	sc_tst_acc_score	sc_precision	sc_recall	pipe_object
0	Set_1	LogisticRegression	0.59850	0.597333	0.587450	0.598639	Pipeline(steps=[('standardscaler', StandardSca...
1	Set_1	SVC	0.95375	0.791333	0.782086	0.795918	Pipeline(steps=[('standardscaler', StandardSca...
2	Set_1	DecisionTree	0.73125	0.674000	0.663564	0.678912	Pipeline(steps=[('standardscaler', StandardSca...
3	Set_1	KNN	1.00000	0.792667	0.784946	0.794558	Pipeline(steps=[('standardscaler', StandardSca...
4	Set_2	LogisticRegression	0.60425	0.595333	0.587432	0.585034	Pipeline(steps=[('standardscaler', StandardSca...
5	Set_2	SVC	0.94775	0.809333	0.798141	0.817687	Pipeline(steps=[('standardscaler', StandardSca...
6	Set_2	DecisionTree	0.77425	0.702000	0.685567	0.723810	Pipeline(steps=[('standardscaler', StandardSca...
7	Set_2	KNN	1.00000	0.801333	0.784135	0.820408	Pipeline(steps=[('standardscaler', StandardSca...
8	Set_3	LogisticRegression	0.59750	0.594000	0.580769	0.616327	Pipeline(steps=[('standardscaler', StandardSca...
9	Set_3	SVC	0.95300	0.960000	0.951807	0.967347	Pipeline(steps=[('standardscaler', StandardSca...
10	Set_3	DecisionTree	0.75650	0.742000	0.716958	0.782313	Pipeline(steps=[('standardscaler', StandardSca...
11	Set_3	KNN	1.00000	1.000000	1.000000	1.000000	Pipeline(steps=[('standardscaler', StandardSca...

As seen from the table above, KNN and SVC performed the best again, for the Database data.

KNN ‘gridsearch’ results for validation data:

- Accuracy: 1.0
- Precision: 1.0
- Recall: 1.0

SVC ‘gridsearch’ results for validation data:

- Accuracy: 0.96
- Precision: 0.9518
- Recall: 0.9673

Building a Final Model:

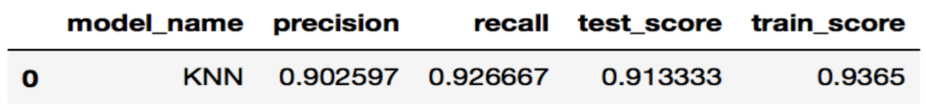
Based on the results above, I ran the entire training data I have for both models KNeighborsClassifier and SVC. Since KNN performed better, I selected it to build my final model.

I built my final pipeline to:

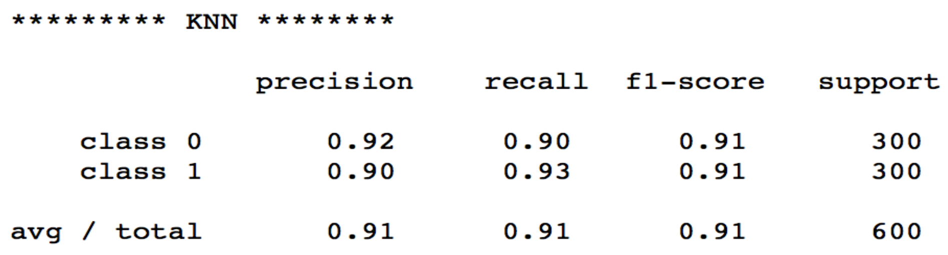
- Scale the data
- PCA the data
- Fit and score the data, using KNeighborsClassifier

To train and fit my final model, for the UCI data I used all 2000 rows of the training data and 600 rows of provided as validation data. For the Database, I used 12000 rows of training data, and 3000 rows of validation data. The results for the UCI data are displayed below.

UCI final model prediction results:



	model_name	precision	recall	test_score	train_score
0	KNN	0.902597	0.926667	0.913333	0.9365



```
********* KNN ********
                 precision    recall  f1-score   support

    class 0         0.92      0.90      0.91       300
    class 1         0.90      0.93      0.91       300

avg / total         0.91      0.91      0.91       600
```

Benchmark results:

	dataset	model	raw_tr_acc_score	raw_tst_acc_score	raw_precision	raw_recall	sc_tr_acc_score	sc_tst_acc_score	sc_precision	sc_recall
0	Set_1	LogisticRegression	1.000000	0.500000	0.548387	0.472222	1.000000	0.454545	0.500000	0.416667
1	Set_1	SVC	1.000000	0.454545	0.000000	0.000000	1.000000	0.500000	0.565217	0.361111
2	Set_1	DecisionTree	1.000000	0.454545	0.500000	0.472222	1.000000	0.454545	0.500000	0.472222
3	Set_1	KNN	0.783582	0.560606	0.629630	0.472222	0.753731	0.545455	0.615385	0.444444
4	Set_2	LogisticRegression	1.000000	0.560606	0.448276	0.500000	1.000000	0.545455	0.423077	0.423077
5	Set_2	SVC	1.000000	0.606061	0.000000	0.000000	1.000000	0.590909	0.473684	0.346154
6	Set_2	DecisionTree	1.000000	0.575758	0.473684	0.692308	1.000000	0.575758	0.473684	0.692308
7	Set_2	KNN	0.776119	0.606061	0.500000	0.576923	0.686567	0.575758	0.466667	0.538462
8	Set_3	LogisticRegression	1.000000	0.560606	0.562500	0.545455	1.000000	0.484848	0.484848	0.484848
9	Set_3	SVC	1.000000	0.500000	0.500000	1.000000	1.000000	0.530303	0.527778	0.575758
10	Set_3	DecisionTree	1.000000	0.454545	0.457143	0.484848	1.000000	0.454545	0.457143	0.484848
11	Set_3	KNN	0.813433	0.651515	0.656250	0.636364	0.708955	0.515152	0.514286	0.545455

As see from tables above the final model results are amazingly better than the benchmark results.

The KNN benchmark results, for scaled data were around:

- Accuracy: 0.5152
- Precision: 0.5143
- Recall: 0.5455

The final results are:

- Accuracy: 0.9133
- Precision: 0.9026
- Recall: 0.9267

Now let's look at the Database data final model prediction results:

	model_name	precision	recall	test_score	train_score
0	KNN	0.850727	0.87619	0.864	0.873167

```
********* KNN ********

             precision    recall  f1-score   support

    class 0       0.88      0.85      0.86       765
    class 1       0.85      0.88      0.86       735

avg / total       0.86      0.86      0.86      1500
```

Benchmark results:

	dataset	model	raw_tr_acc_score	raw_tst_acc_score	raw_precision	raw_recall	sc_tr_acc_score	sc_tst_acc_score	sc_precision	sc_recall
0	Test_Set_1	LogisticRegression	0.801866	0.543182	0.537984	0.532209	0.801866	0.543182	0.537984	0.532209
1	Test_Set_1	SVC	1.000000	0.623485	0.614815	0.636503	1.000000	0.579545	0.573374	0.581288
2	Test_Set_1	DecisionTree	1.000000	0.595455	0.592187	0.581288	1.000000	0.595455	0.592187	0.581288
3	Test_Set_1	KNN	0.754104	0.599242	0.584131	0.654908	0.718284	0.517424	0.510067	0.582822
4	Test_Set_2	LogisticRegression	0.803731	0.538636	0.570533	0.520744	0.803731	0.538636	0.570533	0.520744
5	Test_Set_2	SVC	1.000000	0.621970	0.648810	0.623748	1.000000	0.582576	0.615265	0.565093
6	Test_Set_2	DecisionTree	1.000000	0.628030	0.647727	0.652361	1.000000	0.628030	0.647727	0.652361
7	Test_Set_2	KNN	0.760448	0.593182	0.616379	0.613734	0.717910	0.563636	0.586742	0.595136
8	Test_Set_3	LogisticRegression	0.795149	0.496970	0.519461	0.502899	0.795149	0.496970	0.519461	0.502899
9	Test_Set_3	SVC	0.998134	0.504545	0.527864	0.494203	0.998134	0.500758	0.524181	0.486957
10	Test_Set_3	DecisionTree	0.998134	0.492424	0.514881	0.501449	0.998134	0.492424	0.514881	0.501449
11	Test_Set_3	KNN	0.684328	0.500758	0.524181	0.486957	0.694030	0.525000	0.548686	0.514493

As see from tables above the final model results are much better than the benchmark results. The KNN benchmark results, for scaled data were around:

- Accuracy: 0.525
- Precision: 0.5487
- Recall: 0.5145

The final results:

- Accuracy: 0.864
- Precision: 0.8507
- Recall: 0.8762

Conclusion:

The purpose of this project was to demonstrate a capacity to identify relevant features, in a dataset, using machine learning. This was done by analyzing Madelon, a synthetic dataset. As we can see from our exploration results, by applying various techniques of machine learning, we were able to take a large dataset, comprised of many features, and effectively reduce its dimensionality, by extracting only the most relevant ones.

By only using the most informative features to train our model, we can significantly improve its prediction power and effectively reduce its processing time. Clearly, this exploration of the Madelon dataset, demonstrates the capacity to identify relevant features in a big dataset, using machine learning.