Executive Summary

For our upcoming bid to Company X, our submission must include machine learning in order to achieve a high level of competitiveness against other bidders.

To demonstrate the power of machine learning, I have utilized the Madelon dataset. This dataset is highly non-linear, contains 500 features, and predicts a binary classification target. Based on these characteristics, I have selected two models that traditionally perform well: Logistic Regression (LR) and K Nearest Neighbors (KNN).

The work contained within this report outlines three different approaches (see summary below), each with selected parameters based on machine learning.

1. **LR using Standard Scaler** with little regularization, utilizing l2, known as the ridge model
2. **LR using Standard Scaler** with regularization (default gamma of 1), utilizing l1, known as the lasso model
3. **LR and KNN using SelectKBest** with regularization, LR utilizing l2, and a grid search to cross validate results

The conclusion of the work herein has confirmed the assertion that machine learning is necessary in order to present a competitive bid. Our model KNN using SelectKBest has selected 5 salient features (out of 500) to predict our target with an acceptable test score (>85%). The ability to select features through machine learning will allow our bid to seriously compete on price and allow for a shortened timeline, leaving a positive impression on our potential client and the ability to sell additional work in the future.

```python
from sqlalchemy import create_engine
import pandas as pd
from sklearn.model_selection import train_test_split


def load_data_from_database(user, password, url, port, database, table):
    """
    Read arguments provided to pass to create_engine (using sqlalchemy) to return the
    sql table in a pandas DataFrame.
    """

    engine = create_engine('postgresql://{}:{}@{}:{}/{}'.format(user, password, url, port,
                                                               database))

    return pd.read_sql_table(table, con = engine)



def add_processes(process, data_dict):

    if 'processes' in data_dict:
        data_dict['processes'].append(process)
    else:
        data_dict['processes'] = [process]

def make_data_dict(data, random_state = None, test_size = 0.25):
    """
    Performs a test train split, where 'X' features contain the string 'feat' and 'y'
    target contains the string 'label' to return a dictionary of the train and test sets.
    """

    data_dict = {}

    X = data[[col for col in data.columns if 'feat' in col]]
    y = data['label']

    X_train, X_test, y_train, y_test = train_test_split(X, y, random_state = random_state,
                                                        test_size = test_size)

```

```
In [1]: from lib.project_5_ADH import load_data_from_database, make_data_dict, general_model, general_transformer
        from sklearn.preprocessing import StandardScaler
        from sklearn.linear_model import LogisticRegression
```

Step 1 - Benchmarking

NOTE: EACH OF THESE SHOULD BE WRITTEN SOLELY WITH REGARD TO STEP 1 - BENCHMARKING

Domain and Data

The data, referred to as Madelon, is 2000 rows and 500 features (1 index, 1 target). The dataset is artificial with a two-class target (-1, 1) with continuous input (parameter) variables.

Problem Statement

Implement a machine learning pipeline designed to perform a naive logistic regression as a baseline model.

Solution Statement ¶

Provide a jupyter notebook that will control the baseline model pipeline with little (or no) regularization.

Metric

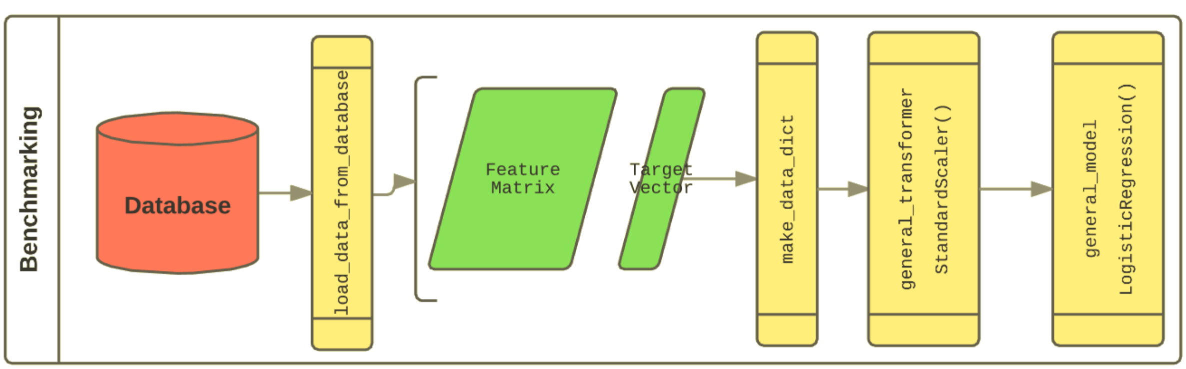
The metric that I will use with my logistic regression is accuracy rate.

Benchmark

The benchmark I will judge the baseline model against will be the baseline accuracy benchmark, which is the percentage of the database that would be guessed correctly if the majority target were applied to the entire dataset.

Implementation

Implement the following code pipeline using the functions you write in `lib/project_5_ADH.py`.



In [2]:
```
madelon_df = load_data_from_database('dsi_student', 'correct horse battery stapl
e', 'joshuacook.me',
                                     '5432', 'dsi', 'madelon')
```

In [3]:
```
madelon_df.head()
```

Out[3]:

	index	feat_000	feat_001	feat_002	feat_003	feat_004	feat_005	feat_006	feat_007	feat_008	...
0	0	485	477	537	479	452	471	491	476	475	...
1	1	483	458	460	487	587	475	526	479	485	...
2	2	487	542	499	468	448	471	442	478	480	...
3	3	480	491	510	485	495	472	417	474	502	...
4	4	484	502	528	489	466	481	402	478	487	...

5 rows × 502 columns

Accuracy Score

In [4]:
```
madelon_df['label'].describe()
```

Out[4]:
```
count    2000.00000
mean        0.00000
std         1.00025
min        -1.00000
25%        -1.00000
50%         0.00000
75%         1.00000
max         1.00000
Name: label, dtype: float64
```

Modeling

In [5]:
```
data_dict = make_data_dict(madelon_df, random_state = 40, test_size = 0.20)
```

In [6]:
```
data_dict = general_transformer(StandardScaler(), data_dict)
```

In [7]:
```
lg_c_small = general_model(LogisticRegression(C = 1E10), data_dict)
```

In [8]:
```
data_dict['train score'], data_dict['test score']
```

Out[8]:
```
(0.78125, 0.55249999999999999)
```

Results

I beat our accuracy score of 50% with a test score of ~55%.

Note, our accuracy score is 50% given the mean of the target ('label' column) is 0. Because the target is binary, either -1 or 1, we know that a perfect mean of 0.0 indicates that there is an equal amount of -1 and 1 in our target.

In [1]:
```
from lib.project_5_ADH import load_data_from_database, make_data_dict, general_mo
del, general_transformer
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LogisticRegression
```

Step 2 - Identify Salient Features Using $\ell 1$-penalty

Domain and Data

The data, referred to as Madelon, is 2000 rows and 500 features (1 index, 1 target). The dataset is artificial with a two-class target (-1, 1) with continuous input (parameter) variables.

Problem Statement

Implement a machine learning pipeline using Logisitic Regression and the l1 penalty to select salient features programatically.

Solution Statement

Provide a jupyter notebook with a pipeline (with regularization) that will provide the number of salient features used in the model.

Metric

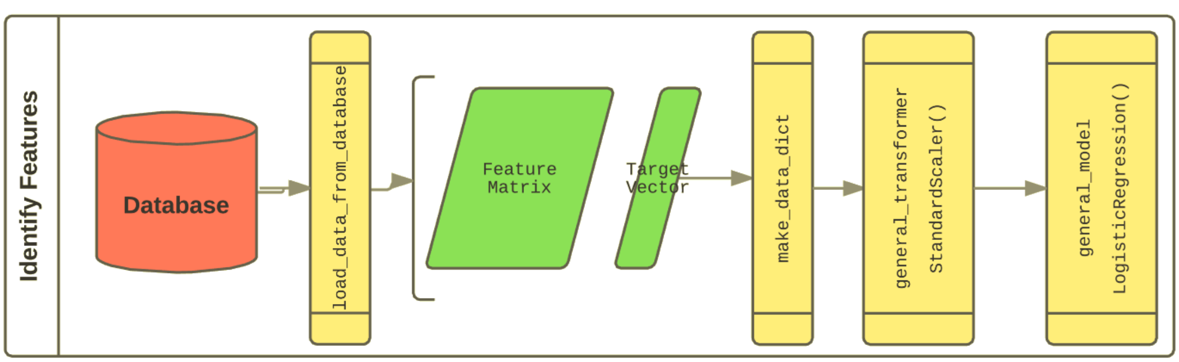
I would like to identify the number of salient features.

Benchmark

I'd like to get as little salient features as possible.

Implementation

Implement the following code pipeline using the functions you write in `lib/project_5_ADH.py`.



In [2]:
```
madelon_df = load_data_from_database('dsi_student', 'correct horse battery stapl
e', 'joshuacook.me',
                                     '5432', 'dsi', 'madelon')
```

In [3]:
```
madelon_df.head()
```

Out[3]:

	index	feat_000	feat_001	feat_002	feat_003	feat_004	feat_005	feat_006	feat_007	feat_008	...
0	0	485	477	537	479	452	471	491	476	475	...
1	1	483	458	460	487	587	475	526	479	485	...
2	2	487	542	499	468	448	471	442	478	480	...
3	3	480	491	510	485	495	472	417	474	502	...
4	4	484	502	528	489	466	481	402	478	487	...

5 rows × 502 columns

In [4]:
```
data_dict = make_data_dict(madelon_df, random_state = 40, test_size = 0.20)
```

In [5]:
```
data_dict = general_transformer(StandardScaler(), data_dict)
```

In [6]:
```
lg_l1 = general_model(LogisticRegression(penalty = 'l1'), data_dict)
```

In [7]:
```
lg_l1['train score'],lg_l1['test score']
```

Out[7]:
```
(0.77562500000000001, 0.54249999999999998)
```

In [8]:
```
coefs = data_dict['processes'][1].coef_.flatten()
```

In [9]:
```
coefs_abs = [abs(coef) for coef in coefs]
```

In [10]:
```
len([num for num in coefs_abs if num > 0.0001])
```

Out[10]:
```
472
```

Results

Using the l1 penalty with logistic regression, the model identified 472 salient features. If we compare our train and test score, we can see that even these features are not such a great predictor of the entire dataset.

In [1]:
```
from lib.project_5_ADH import load_data_from_database, make_data_dict, general_mo
del, general_transformer
from numpy import arange
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LogisticRegression
from sklearn.feature_selection import SelectKBest
from sklearn.neighbors import KNeighborsClassifier
from sklearn.model_selection import GridSearchCV
import pandas as pd
import numpy as np
```

Step 3 - Build Model

Domain and Data

The data, referred to as Madelon, is 2000 rows and 500 features (1 index, 1 target). The dataset is artificial with a two-class target (-1, 1) with continuous input (parameter) variables.

Problem Statement

Implement a machine learning pipeline using Logisitic Regression and KNeighborsClassifier while transforming the data using SelectKBest.

Solution Statement

Provide a jupyter notebook with a pipeline (with regularization) that will show how the Logistic Regression and KNeighbors models work. Check how many salient features each use.

Metric

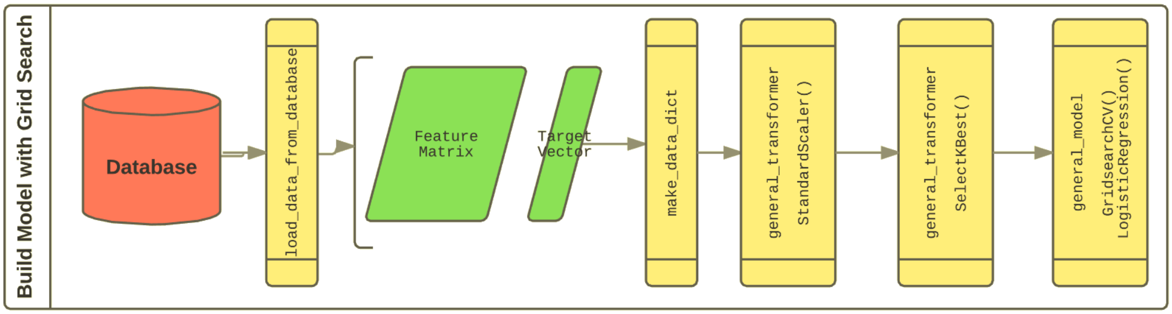
I would like to reduce the amount of salient features that I determined in the prior workbook (step 2).

Benchmark

I would like to beat my test score from step 1 of ~55%.

Implementation

Implement the following code pipeline using the functions you write in `lib/project_5.py`.



In [2]:
```
madelon_df = load_data_from_database('dsi_student', 'correct horse battery stapl
e', 'joshuacook.me',
                                     '5432', 'dsi', 'madelon')
```

In [3]:
```
madelon_df.head()
```

Out[3]:

	index	feat_000	feat_001	feat_002	feat_003	feat_004	feat_005	feat_006	feat_007	feat_008	...
0	0	485	477	537	479	452	471	491	476	475	...
1	1	483	458	460	487	587	475	526	479	485	...
2	2	487	542	499	468	448	471	442	478	480	...
3	3	480	491	510	485	495	472	417	474	502	...
4	4	484	502	528	489	466	481	402	478	487	...

5 rows × 502 columns

In [4]:
```
data_dict = make_data_dict(madelon_df, random_state = 40, test_size = 0.20)
```

In [5]:
```
data_dict = general_transformer(StandardScaler(), data_dict)
```

In [6]:
```
data_dict = general_transformer(SelectKBest(), data_dict)
```

In [7]:
```
K_best_selection = data_dict['processes'][1]
```

In [8]:
```
np.where(K_best_selection.get_support())
```

Out[8]:
```
(array([ 48,  64, 105, 128, 241, 336, 338, 442, 472, 475]),)
```

In [9]:
```
LR_scores = general_model(LogisticRegression(), data_dict)
LR_scores['train score'], LR_scores['test score']
```

Out[9]:
```
(0.61499999999999999, 0.59750000000000003)
```

In [10]:
```
data_dict['processes'][2].coef_.flatten()
# SelectKBest only chose 10!
```

Out[10]:
```
array([ 0.13579799, -0.11706957,  0.05055887,  0.15962245,  0.16447294,
        0.21886389,  0.20087084, -0.31649912,  0.2762322 ,  0.40462379])
```

In [11]:
```
KNN_scores = general_model(KNeighborsClassifier(), data_dict)
KNN_scores['train score'], KNN_scores['test score']
```

Out[11]: (0.9106250000000002, 0.8625000000000004)

In [12]:
```
lg_param_grid = {'C': [0.0001, 0.001, 0.01, 0.1, 1.0, 10.0, 100.0, 1000.0]}
GSCV_LR_scores = general_model(GridSearchCV(LogisticRegression(),lg_param_grid),
data_dict)
GSCV_LR_scores['train score'], GSCV_LR_scores['test score']
```

Out[12]: (0.6131250000000003, 0.5999999999999998)

In [13]:
```
GSCV_LR = data_dict['processes'][4]
```

In [14]:
```
GSCV_LR_df = pd.DataFrame(GSCV_LR.cv_results_)
GSCV_LR_df[['mean_test_score','mean_train_score', 'param_C', 'rank_test_score']]
# best estimator is c = 0.1
```

Out[14]:

	mean_test_score	mean_train_score	param_C	rank_test_score
0	0.593750	0.590936	0.0001	7
1	0.591875	0.596875	0.001	8
2	0.600625	0.608437	0.01	2
3	0.603750	0.615623	0.1	1
4	0.600625	0.616248	1	2
5	0.596250	0.610938	10	4
6	0.595625	0.610313	100	5
7	0.595625	0.610313	1000	5

In [15]:
```
knn_param_grid = {'n_neighbors': [x for x in arange(3, 22, 2)]}
GSCV_knn_scores = general_model(GridSearchCV(KNeighborsClassifier(),knn_param_gri
d), data_dict)
GSCV_knn_scores['train score'], GSCV_knn_scores['test score']
```

Out[15]: (0.9106250000000002, 0.8625000000000004)

In [16]:
```
GSCV_KNN = data_dict['processes'][5]
```

In [17]:
```
GSCV_KNN_df = pd.DataFrame(GSCV_KNN.cv_results_)
GSCV_KNN_df[['mean_test_score','mean_train_score', 'param_n_neighbors', 'rank_tes
t_score']]
# best estimator is neighbors = 5
```

Out[17]:

	mean_test_score	mean_train_score	param_n_neighbors	rank_test_score
0	0.848750	0.931246	3	2
1	0.850000	0.903437	5	1
2	0.844375	0.890624	7	3
3	0.839375	0.874374	9	4
4	0.826250	0.871249	11	5
5	0.821875	0.863436	13	6
6	0.815625	0.856873	15	8
7	0.816875	0.848749	17	7
8	0.811250	0.844061	19	9
9	0.808750	0.834998	21	10

Results

KNearestNeighbors (KNN) immensely outperformed Logisitic Regression (LR). Train and test scores for KNN and LR are (0.91, 0.86) and (0.61, 0.60), respectively, when a grid search is performed.