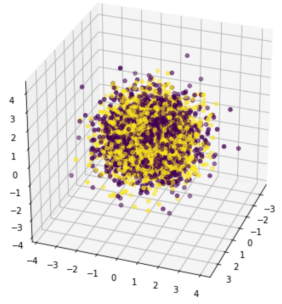
Project 3
Madelon Madness Feature Selection

Introduction

MADELON is an artificial dataset, which was part of the N IPS 2003 feature selection challenge. This is a two-class classification problem with continuous input variables. The difficulty is that the problem is multivariate and highly non-linear. The dataset contains data points grouped in thirty-two clusters placed on the vertices of a five dimensional hypercubes and randomly labeled +1 or -1. The five dimensions constitute five informative features. Fifteen linear combinations of those features were added to form a set of 10 (redundant) informative features. Based on those twenty features, one must separate the examples into the two classes corresponding to the +-1 labels. A number of distractor features called 'probes' having no predictive power were added, and the order of the features and patterns are randomized.[1] This makes dealing with the modeling difficult with linear models.

```
# Madelon dataset generation code
X, y = make_classification(n_samples=4400,
                           n_features=2,
                           n_redundant=0,
                           n_informative=2,
                           n_clusters_per_class=2,
                           class_sep=3.0)

# Visualize hypercube
from mpl_toolkits.mplot3d import axes3d
fig = plt.figure(figsize=(7,7))
ax = fig.add_subplot(111,projection='3d')
data_1 = X[:,0]
data_2 = X[:,1]
data_3 = X[:,2]
_ = ax.scatter(data_1, data_2, data_3, c=y)
cmaps = 'magma'
ax.view_init(30,20)
```



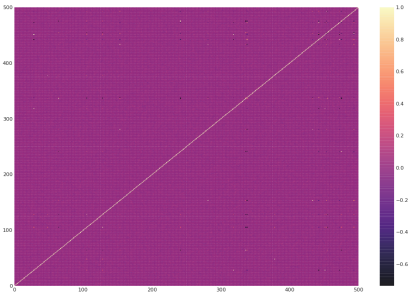
The purpose of this project is to develop a series of models for two purposes: 1.) identifying relevant features, and 2.) generating predictions from the model.

[1] https://git.generalassemb.ly/wblakecannon/project_3/blob/master/_notes/office-hours-2017-10-23-explore-make-classification.ipynb

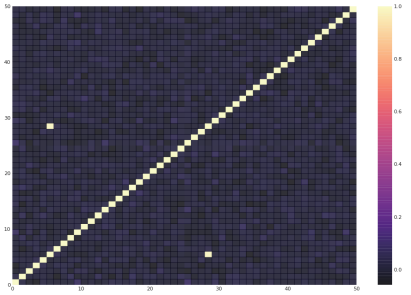
Exploratory Data Analysis

The Madelon dataset[2] is available at UCI's Machine Learning Repository[3] and was imported into a Python Jupyter Notebook[4] environment as NumPy[5] arrays. NunPy arrays were chosen over Pandas DataFrames as an effort to reduce memory footprint. The training data was (2000, 500) and the validation data was (600, 500) in matrix form. A Pearson product-moment correlation coefficient plot was created to visualize correlations with the variables.[6]

```
# Plotting Pearson product-moment correlation coefficients (np.corrcoef)
# by using a pseudo color plot of a 2D array

# Full dataset:
plt.figure(figsize=(15,10), dpi=80)
plt.pcolor(np.corrcoef(X_train, rowvar=False),
        cmap=mpl.cm.magma,
        alpha=0.8)
plt.colorbar()
plt.savefig('./_assets/1-1-pcolor-full.png')
plt.show()
```



```
# Plotting Pearson product-moment correlation coefficients (np.corrcoef)
# by using a pseudo color plot of a 2D array
# Sample  dataset:
plt.figure(figsize=(15,10), dpi=80)
plt.pcolor(np.corrcoef(X_train[:,100:150], rowvar=False),
        cmap=mpl.cm.magma,
        alpha=0.8)
plt.colorbar()
plt.savefig('./_assets/1-2-pcolor-sample.png')
plt.show()
```



It can be seen with these visualizations the extent of the dataset's features as to how only a few very of them have significant correlation. Thus, the dataset is essentially a large amount of noise.

---

[2] http://archive.ics.uci.edu/ml/datasets/madelon

[3] http://archive.ics.uci.edu/ml/index.php

[4] http://jupyter.org

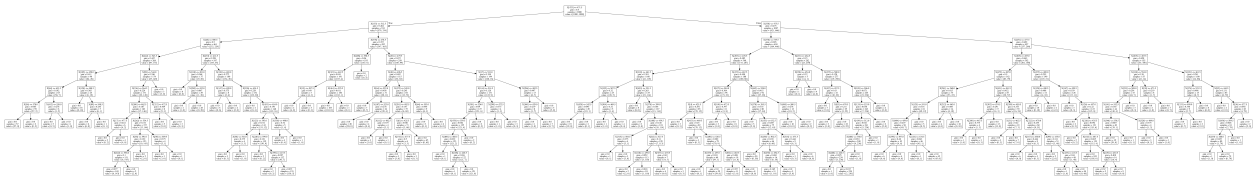
[5] http://www.numpy.org

[6] https://git.generalassemb.ly/wblakecannon/project_3/blob/master/01-eda.ipynb

Decision Tree

A decision tree is a tool that uses a tree-like graph or model of decisions and their possible consequences, including the chance event outcomes, as a display for algorithms that contain conditional control statements. The benefit of decision trees is thatchy are good at isolating and building on important features, highly interpretable, scaling is not needed (decisions are made individually), they work well with a large number of classes, work well with either categorical or numerical variables, and are highly customizable. The weaknesses of decision trees are they do not do well with noisy data, prone to overfitting if left to grow too deep, and potentially computationally expensive. A decision tree classifier was run on the Madelon dataset, and resulted with a score of 0.765.[7] A weakness of decision trees in this particular dataset is that it is extremely difficult to comprehend. Thus, DecisionTreeClassifier may not be the best model to solve the Madelon problem.

```
tree_clf = DecisionTreeClassifier(max_depth=10)
tree_clf.fit(X_train, y_train)

def show_tree(decisionTree, file_path):
    dotfile = io.StringIO()
    export_graphviz(decisionTree, out_file=dotfile)
    pydot.graph_from_dot_data(dotfile.getvalue()).write_png(file_path)
    i = misc.imread(file_path)
    plt.imshow(i)

show_tree(tree_clf, './_assets/2-1-tree-class.png')

tree_clf.score(X_val, y_val)
```



Baseline

A logistic regression model was run to score the error measured using area under the curve as a metric.[8] To create a benchmark, roc_auc_score from sklearn.metrics was used to measure the

[7] https://git.generalassemb.ly/wblakecannon/project_3/blob/master/02-decision-tree.ipynb

[8]https://git.generalassemb.ly/wblakecannon/project_3/blob/master/03-auc-fclassif.ipynb

rate of positives vs negatives. Note that a completely random selection would be a baseline of 0.5.

```
# Import packages
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import roc_auc_score
# Instatiate and fit the logistic regression model
logr = LogisticRegression()
logr.fit(X_train,y_train)
logr.predict_proba(X_train)

# Find the ROC/AUC on one column of the logistic regression predict_proba on the training dataset
roc_auc_score(y_train,logr.predict_proba(X_train)[:,1])

# Find the AUC on one column of the logistic regression predict_proba on the validation dataset
roc_auc_score(y_val,logr.predict_proba(X_val)[:,1])
```

The baseline AUC score of 0.602 on the validation set is not very good considering random guessing should give a score of 0.5, 1 would be the maximum score possible. The score of 0.824 on the training set is overfit. The extent of the unimportant features in the Madelon dataset is now known. This is due to the difference between the ROC-AUC score differences of 0.602 on the validation set and 0.824 on the training set. Thus, the unimportant features must be removed to create a better model. This is known as feature selection.

Feature Selection using f_classif

In order to find the optimal features in the Madelon dataset, a f_classif test from the sklearn package was run. Other testing methods are available, but this is a classification problem and f_classif is specifically designed for this type of problem.

What are F-statistics and the F-test?

F-tests are named after its test statistic, F, which was named in honor of Sir Ronald Fisher. The F-statistic is simply a ratio of two variances. Variances are a measure of dispersion, or how far the data are scattered from the mean. Larger values represent greater dispersion.

F is for F-test Variance is the square of the standard deviation. For us humans, standard deviations are easier to understand than variances because they're in the same units as the data rather than squared units. However, many analyses actually use variances in the calculations.

F-statistics are based on the ratio of mean squares. The term "mean squares" may sound confusing but it is simply an estimate of population variance that accounts for the degrees of freedom (DF) used to calculate that estimate.

Despite being a ratio of variances, you can use F-tests in a wide variety of situations. Unsurprisingly, the F-test can assess the equality of variances. However, by changing the variances that are included in the ratio, the F-test becomes a very flexible test. For example, you can use F-statistics and F-tests to test the overall significance for a regression model, to compare the fits of different models, to test specific regression terms, and to test the equality of means.

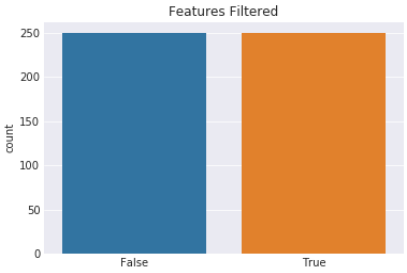
To use the F-test to determine whether group means are equal, it's just a matter of including the correct variances in the ratio. In one-way ANOVA, the F-statistic is this ratio:

F = variation between sample means / variation within the samples

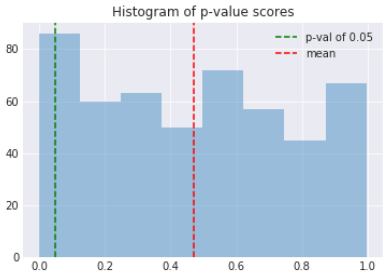
-The Minitab Blog[9]

This test will return a p-value. As with p-values, a small p-value ($p < 0.05$) would indicate whether or not a feature is relative or irrelative. The next step is combining sklearn's f_classif and SelectPercentile to compute the ANOVA f-values, p-values, and choose a certain percentage of relevant features.

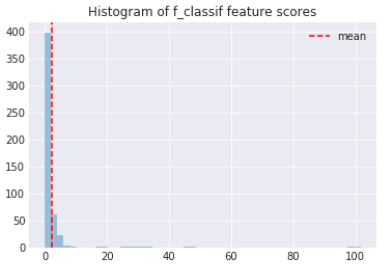
```
# Import packages
from sklearn.feature_selection import f_classif, SelectPercentile
from sklearn.preprocessing import PolynomialFeatures

# Set up feature selection by selecting most relative 50 percent of features
select_feature = SelectPercentile(f_classif, percentile=50)
# Fit the feature selection model
select_feature.fit(X_train,y_train)

# Now we'll make an array of booleans for those upper 50 percent of features
feature_filter = select_feature.get_support()

# Plot the filter results
sns.countplot(feature_filter)
plt.title('Features Filtered')
plt.savefig('./_assets/3-1-feature-filter')
plt.show()
```




```
# Here is the mean and array of f_classif p-values of the features
print('f_classif p-values mean:', select_feature.pvalues_.mean())
        >>>f_classif p-values mean: 0.471989720132


# Plot histogram of p-value scores
sns.distplot(select_feature.pvalues_, kde=False)
plt.axvline(.05, linestyle='dashed', label='p-val of 0.05', color='g')
plt.axvline(select_feature.pvalues_.mean(), label='mean', linestyle='dashed', color='r')
plt.title('Histogram of p-value scores')
plt.legend()
plt.savefig('./_assets/3-2-hist-pvals')
plt.show()
```




```
# Here is the mean and array of f_classif scores of the features
print('f_classif scores mean:', select_feature.scores_.mean())
        >>>f_classif scores mean: 2.10801794996

# Plot histogram of f_classif scores
sns.distplot(select_feature.scores_, kde=False)
plt.axvline(select_feature.scores_.mean(), label='mean', linestyle='dashed', color='r')
plt.title('Histogram of f_classif feature scores')
plt.legend()
plt.savefig('./_assets/3-3-hist-f_classif-scores')
plt.show()
```



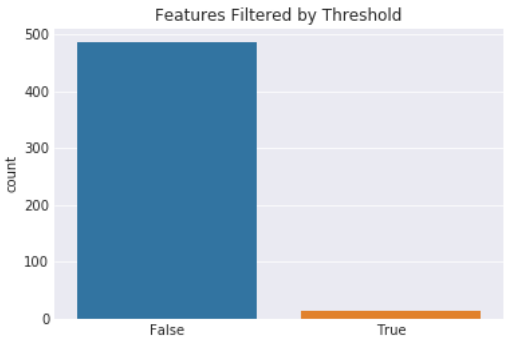
[9] http://blog.minitab.com/blog/adventures-in-statistics-2/understanding-analysis-of-variance-anova-and-the-f-test

The mean of the select_feature.scores_ is 2.11 and the mean of the select_feature.pvalues_ is 0.47. This once again shows us that there is a high amount of unimportant features. This same information can clearly be seen in the two histograms.

The histogram of select_feature.pvalues_ also shows how many unimportant features there are. The green dashed line on the histogram represents the typical p-value cutoff of 0.05, and the red dashed line is the mean. We can see with this high mean and distribution of p-scores greater than 0.05. A small p-value (less than 0.05) would indicate that a certain feature was useful in predicting the target.

The histogram of select_feature.scores_ indicates that most of the scores are very close to zero. There are a handful of high ranking scores but they are so low in appearance that they can't really be seen on the histogram. We really only know these high scores exist because of the extent of the x-axis. Next, a threshold was created and used to pick which features are important.

```
# Create array of booleans of features with scores greater than 10
feature_filter = select_feature.scores_ > 10

# Print sum of 'True' in feature_filter
print('Number of filtered features:', np.sum(feature_filter))
print('Number of un-filtered features:', len(feature_filter) - np.sum(feature_filter))
        >>> Number of filtered features: 13
        >>> Number of un-filtered features: 487

# Plot the filter results
sns.countplot(feature_filter)
plt.title('Features Filtered by Threshold')
plt.savefig('./_assets/3-4-feature-filter-threshold')
plt.show()

# Generate a new feature matrix consisting of all polynomial combinations of
# the features with degrees less than or equal to 2
interactions = PolynomialFeatures(degree=2, interaction_only=True)
X_interactions = interactions.fit_transform(X_train[:,feature_filter])
print('No of features and their interactions:', X_interactions.shape[1])
        >>> No of features and their interactions: 92

# Fit_transform to see if there's any relationships in the model
logr.fit(X_interactions, y_train)
X_val_filtered = interactions.fit_transform(X_val[:,feature_filter])
# Print validation score on the filtered X
roc_auc_score(y_val, logr.predict_proba(X_val_filtered)[:,1])
        >>> 0.80815
```



The ROC/AUC score of 0.808 is just slightly below the original *overfit* score of 0.824 on the training dataset. Thus, it seems that using p_classif as a feature selection could be valid because the lower score would mean it's not overfitting but also doing a good job at solving the madelon problem.

```
# np array of whether or not the feature is important
feature_filter

counter = -1
important_features = []
for i in feature_filter:
    counter += 1
    if i == True:
        important_features.append(counter)
print('Number of important features:', len(important_features))
print('List of important features:', important_features)
          >>> Number of important features: 13
          >>> List of important features: [48, 64, 105, 128, 241, 336, 338, 378, 442, 453, 472, 475, 493]
```

The feature selection method above is known as **univariate selection** which decides the best features by looking at how they work individually. Univariate selection does not look at how they would preform together in harmony. The weakness in the method above is collinearity. These methods are simple to run and understand and are in general particularly good for gaining a better understanding of data (but not necessarily for optimizing the feature set for better generalization).

In datasets with smaller amount of noise, the correlation is relatively strong, with a very low p-value, while in large datasets with lots of noise (like Madelon), the correlation is very small. Thus, the p-value is high meaning that it is very likely to observe such correlation is by chance.

Stability Selection Pipeline

```
# Import packages
from sklearn.linear_model import RandomizedLogisticRegression, LogisticRegression
from sklearn.preprocessing import PolynomialFeatures
from sklearn.pipeline import make_pipeline
from sklearn.metrics import roc_auc_score

# Instatiate and fit the logistic regression model
logr = LogisticRegression()
logr.fit(X_train,y_train)

# Manually chosen threshold for testing
threshold = 0.05
```

```
stability_selection = RandomizedLogisticRegression(n_resampling=300, n_jobs=1, scaling=0.15, sample_fraction=0.50,
selection_threshold=threshold)
interactions = PolynomialFeatures(degree=4, interaction_only=True)
model = make_pipeline(stability_selection, interactions, logr)
model.fit(X_train, y_train)

print('Number of features picked by stability selection: %i' % np.sum(model.steps[0][1].all_scores_ >= threshold))
        >>> Number of features picked by stability selection: 13

print('Area Under the Curve: %0.5f' % roc_auc_score(y_val, model.predict_proba(X_val)[:,1]))
        >>> Area Under the Curve: 0.89703

feature_filter = model.steps[0][1].all_scores_ >= threshold

counter = -1
important_features = []
for i in feature_filter:
    counter += 1
    if i == True:
        important_features.append(counter)

print('Number of important features:', len(important_features))
        >>> Number of important features: 13

print('List of important features:', important_features)
        >>> List of important features: [48, 64, 105, 128, 241, 323, 336, 338, 378, 442, 453, 472, 475]
```

Find Optimal Threshold

To find the optimal threshold, the same computations were run above in a loop of threshold values ranging from 0.01 to 0.13 in steps of 0.01.

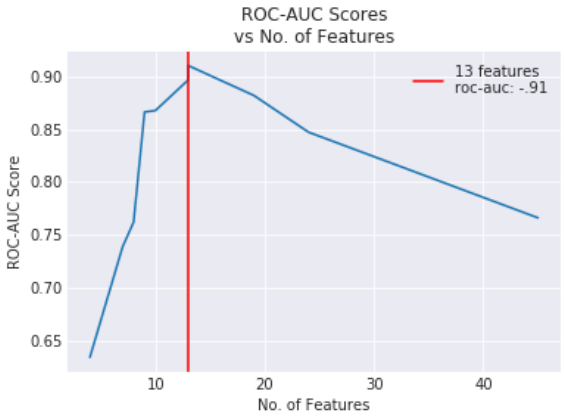
```
from sklearn.pipeline import make_pipeline
from sklearn.linear_model import LogisticRegression, RandomizedLogisticRegression
from sklearn.preprocessing import PolynomialFeatures

# Instatiate and fit the logistic regression model
logr = LogisticRegression()
logr.fit(X_train,y_train)

# Build a function that makes ranges with floats
def frange(start, stop, step, roundval):
    i = start
    while i < stop:
        yield round(i, roundval)
        i += step

# Make list of thresholds to test
threshold_list = list(frange(0.01, .13, .01, 2))
threshold_list
        >>> [0.01, 0.02, 0.03, 0.04, 0.05, 0.06, 0.07, 0.08, 0.09, 0.10, 0.11, 0.12, 0.13]

features_list = []
rocauc_list = []

for thresh in threshold_list:
    stability_selection = RandomizedLogisticRegression(n_resampling=300,
                                        n_jobs=1,
                                        random_state=101,
                                        scaling=0.15,
                                        sample_fraction=0.50,
                                        selection_threshold=thresh)
    interactions = PolynomialFeatures(degree=4, interaction_only=True)
    model = make_pipeline(stability_selection, interactions, logr)
    model.fit(X_train, y_train)
    feature_filter = model.steps[0][1].all_scores_ >= thresh
```

```
    counter = -1
    important_features = []
    for i in feature_filter:
        counter += 1
        if i == True:
            important_features.append(counter)
    print('Number of important features:', len(important_features))
    print('List of important features:', important_features)

    features_list.append(np.sum(model.steps[0][1].all_scores_ >= thresh))
    rocauc_list.append(roc_auc_score(y_val, model.predict_proba(X_val)[:,1]))
```

```
>>>Number of important features: 45
>>>List of important features: [4, 10, 18, 48, 55, 64, 105, 119, 128, 136, 152, 184, 196, 199, 204, 205, 211, 226, 241, 245, 251, 282, 296,
306,                323, 329, 333, 336, 338, 347, 377, 378, 384, 424, 425, 430, 431, 442, 453, 456, 471, 472, 475, 493, 496]

>>>Number of important features: 24
>>>List of important features: [4, 48, 64, 105, 128, 199, 204, 205, 211, 241, 245, 296, 323, 336, 338, 378, 424, 431, 442, 453, 472, 475,
493, 496]

>>>Number of important features: 19
>>>List of important features: [48, 64, 105, 128, 204, 241, 296, 323, 336, 338, 378, 424, 431, 442, 453, 472, 475, 493, 496]

>>>Number of important features: 13
>>>List of important features: [48, 64, 105, 128, 241, 323, 336, 338, 378, 442, 453, 472, 475]

>>>Number of important features: 13
>>>List of important features: [48, 64, 105, 128, 241, 323, 336, 338, 378, 442, 453, 472, 475]

>>>Number of important features: 10
>>>List of important features: [48, 64, 105, 128, 241, 336, 378, 442, 472, 475]

>>>Number of important features: 9
>>>List of important features: [48, 64, 105, 128, 241, 336, 378, 442, 475]

>>>Number of important features: 8
>>>List of important features: [48, 64, 105, 128, 241, 336, 378, 475]

>>>Number of important features: 7
>>>List of important features: [48, 64, 105, 241, 336, 378, 475]

>>>Number of important features: 4
>>>List of important features: [48, 241, 378, 475]

>>>Number of important features: 4
>>>List of important features: [48, 241, 378, 475]

>>>Number of important features: 4
>>>List of important features: [48, 241, 378, 475]

>>>Number of important features: 4
>>>List of important features: [48, 241, 378, 475]
```

threshold	no-of-features	score
0.01	45	0.766233
0.02	24	0.847478
0.03	19	0.882289
0.04	13	0.910600
0.05	13	0.897033
0.06	10	0.867989
0.07	9	0.866556
0.08	8	0.762178
0.09	7	0.738956
0.10	4	0.634022
0.11	4	0.634022
0.12	4	0.634022
0.13	4	0.634022

```
# Results to DataFrame
results_df = pd.DataFrame({'threshold': threshold_list,
                           'score': rocauc_list,
                           'no-of-features': features_list})
results_df.set_index('threshold',inplace=True)
results_df

# Plot ROC-AUC score vs number of features
plt.plot(features_list, rocauc_list)
plt.axvline(13, c='r', label='13 features\nroc-auc: -.91')
plt.xlabel('No. of Features')
plt.ylabel('ROC-AUC Score')
plt.title('ROC-AUC Scores\nvs No. of Features')
plt.legend()
plt.savefig('./_assets/5-2-rocauc-scores-vs-features.png')
plt.show()
```



The code and plot above show that the optimal threshold is 0.04. It selects thirteen features and has a roc_auc_score of 0.9106. The thirteen features are: [48, 64, 105, 128, 241, 323, 336, 338, 378, 442, 453, 472, 475].