Lab 5: Feature Selection

This notebook builds on top of Lab 4 by introducing feature selection into the process of selecting the best classifier for a binary classification problem.

The feature selection method applied here is Recursive Feature Elimination (RFE) as demonstrated in the tutorial at https://machinelearningmastery.com/feature-selection-in-python-with-scikit-learn/ (https://machinelearningmastery.com/feature-selection-in-python-with-scikit-learn/).

In this demonstration we use a modified version of the seeds data set (see https://archive.ics.uci.edu/ml/datasets/seeds (https://archive.ics.uci.edu/ml/datasets/seeds)), which is the same data set used in Lab 4.

A. Preparation

Import Python modules

In [157]:

```
import pandas as pd
import numpy as np

from sklearn import preprocessing #needed for scaling attributes to the nterval [0,1]

from sklearn import svm
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier
from sklearn.feature_selection import RFE

from sklearn.model_selection import train_test_split

import matplotlib.pyplot as plt
```

Load and prepare the dataset for training and evaluation

Feel free to apply any other pre-processing technique at this point.

In [2]:

```
lab5_df = pd.read_csv("./seeds_dataset_binary.csv")
lab5_df.describe()
```

Out[2]:

	area	perimeter	compactness	length of kernel	width of kernel	asymmetry coefficient	length of kerne groove
count	210.000000	210.000000	210.000000	210.000000	210.000000	210.000000	210.000000
mean	14.847524	14.559286	0.870999	5.628533	3.258605	3.700201	5.408071
std	2.909699	1.305959	0.023629	0.443063	0.377714	1.503557	0.491480
min	10.590000	12.410000	0.808100	4.899000	2.630000	0.765100	4.519000
25%	12.270000	13.450000	0.856900	5.262250	2.944000	2.561500	5.045000
50%	14.355000	14.320000	0.873450	5.523500	3.237000	3.599000	5.223000
75%	17.305000	15.715000	0.887775	5.979750	3.561750	4.768750	5.877000
max	21.180000	17.250000	0.918300	6.675000	4.033000	8.456000	6.550000

In [3]:

```
# target attribute
target_attribute_name = 'type'
target = lab5_df[target_attribute_name]

# predictor attributes
predictors = lab5_df.drop(target_attribute_name, axis=1).values

# scale all predictor values to the range [0, 1]
# note the target attribute is already binary
min_max_scaler = preprocessing.MinMaxScaler()
predictors = min_max_scaler.fit_transform(predictors)
```

Split the data set into a training (80%) and test (20%) data sets.

In [4]:

```
# pepare independent stratified data sets for training and test of the final model
predictors_train, predictors_test, target_train, target_test = train_test_split(
    predictors, target, test_size=0.20, shuffle=True, stratify=target)
```

B. Feature Selection

1. Apply RFE with SVM for selecting the best features

In [5]:

```
# create a base classifier used to evaluate a subset of attributes
estimatorSVM = svm.SVR(kernel="linear")
selectorSVM = RFE(estimatorSVM, 3)
selectorSVM = selectorSVM.fit(predictors_train, target_train)
# summarize the selection of the attributes
print(selectorSVM.support_)
print(selectorSVM.ranking_)
```

```
[False False False  True False  True  True]
[5 2 3 1 4 1 1]
```

2. Apply RFE with Logistic Regression for selecting the best features

In [6]:

```
# create a base classifier used to evaluate a subset of attributes
estimatorLR = LogisticRegression()
# create the RFE model and select 3 attributes
selectorLR = RFE(estimatorLR, 3)
selectorLR = selectorLR.fit(predictors_train, target_train)
# summarize the selection of the attributes
print(selectorLR.support_)
print(selectorLR.ranking_)
```

```
[False False  True False False  True  True]
[3 4 1 5 2 1 1]
```

B. Evaluate on the Test Data Set

Apply the selectors to prepare training data sets only with the selected features

Note: The same selectors are applied to the test data set. However, it is important that the test data set was not used by (it's invisible to) the selectors.

In [7]:

```
predictors_train_SVMselected = selectorSVM.transform(predictors_train)
predictors_test_SVMselected = selectorSVM.transform(predictors_test)
```

In [8]:

```
predictors_train_LRselected = selectorLR.transform(predictors_train)
predictors_test_LRselected = selectorLR.transform(predictors_test)
```

Train and evaluate SVM classifiers with both the selected features and all features

Here we train three models:

- model1 - with the features selected by SVM
- model2 - with the features selected by Logistic Regression
- model3 - with all features (i.e. without feature selection)

In [9]:

```
classifier = svm.SVC()
```

In [10]:

```
model1 = classifier.fit(predictors_train_SVMselected, target_train)
model1.score(predictors_test_SVMselected, target_test)
```

Out[10]:

```
0.9285714285714286
```

In [11]:

```
model2 = classifier.fit(predictors_train_LRselected, target_train)
model2.score(predictors_test_LRselected, target_test)
```

Out[11]:

```
0.9047619047619048
```

In [12]:

```
model3 = classifier.fit(predictors_train, target_train)
model3.score(predictors_test, target_test)
```

Out[12]:

```
0.9047619047619048
```

C. Conclusion

The results above, give evidence that model1 is most accurate.

However, when you execute this code again, it is very likely to get different results.

To get more accurate results, accounting for the variance in the results, it is better to run the whole experiment multiple times and measure the variance in the results. Then pick the model that gives better results.

Task 1

In [229]:

```
t1_df = pd.read_csv("./winequality_red.csv")
t1_df.describe()
```

Out[229]:

	fixed acidity	volatile acidity	citric acid	residual sugar	chlorides	free sulfur dioxide	total
count	1599.000000	1599.000000	1599.000000	1599.000000	1599.000000	1599.000000	1599.(
mean	8.319637	0.527821	0.270976	2.538806	0.087467	15.874922	46.4
std	1.741096	0.179060	0.194801	1.409928	0.047065	10.460157	32.8
min	4.600000	0.120000	0.000000	0.900000	0.012000	1.000000	6.(
25%	7.100000	0.390000	0.090000	1.900000	0.070000	7.000000	22.(
50%	7.900000	0.520000	0.260000	2.200000	0.079000	14.000000	38.(
75%	9.200000	0.640000	0.420000	2.600000	0.090000	21.000000	62.(
max	15.900000	1.580000	1.000000	15.500000	0.611000	72.000000	289.(

In [230]:

```
print("Number of Samples  in Dataset:\t",t1_df.shape[0])
print("Number of Features in Dataset:\t",t1_df.shape[1])
```

```
Number of Samples  in Dataset:   1599
Number of Features in Dataset:   12
```

In [231]:

```
# target attribute
target_attribute_name = 'quality'
target = t1_df[target_attribute_name]

print("\n****\t",'quality',"\t****")
print("Total Values = \t",t1_df['quality'].count(),"\n")
print(t1_df['quality'].value_counts(dropna=False))
print("**************************")
```

```
****     quality        ****
Total Values =   1599

5    681
6    638
7    199
4     53
8     18
3     10
Name: quality, dtype: int64
**************************
```

In [232]:

```
# predictor attributes
predictors = t1_df.drop(target_attribute_name, axis=1).values

# scale all predictor values to the range [0, 1]
# note the target attribute is already binary
min_max_scaler = preprocessing.MinMaxScaler()
predictors = min_max_scaler.fit_transform(predictors)
```

Split the data set into a training (80%) and test (20%) data sets.

In [233]:

```
# pepare independent stratified data sets for training and test of the final model
predictors_train, predictors_test, target_train, target_test = train_test_split(
    predictors, target, test_size=0.20, shuffle=True, stratify=target)
```

B. Feature Selection

In [237]:

```
# Add variable to allow the number of features selected to be varied
num_features = 6
```

In [238]:

```
# Function to print details on selected features and ranking
def print_feature_selection(support, ranking, df):
    # Print Attributes that were selected/dropped
    print("The following features were selected (ranking):")
    for i in range(len(support)):
        if(support[i]==True):
            print("\t",df.columns[i], "(",ranking[i],")")

    print("The following features were NOT selected  (ranking):")
    for i in range(len(support)):
        if(support[i]==False):
            print("\t",df.columns[i], "(",ranking[i],")")

    return
```

1. Apply RFE with SVM for selecting the best features

In [239]:

```
# create a base classifier used to evaluate a subset of attributes
estimatorSVM = svm.SVR(kernel="linear")
selectorSVM = RFE(estimatorSVM, num_features)
selectorSVM = selectorSVM.fit(predictors_train, target_train)
# summarize the selection of the attributes
print(selectorSVM.support_)
print(selectorSVM.ranking_)
```

```
[ True  True False False  True False False  True False  True  True]
[1 1 6 2 1 5 3 1 4 1 1]
```

In [240]:

```
print_feature_selection(selectorSVM.support_, selectorSVM.ranking_, t1_df)
```

```
The following features were selected (ranking):
         fixed acidity ( 1 )
         volatile acidity ( 1 )
         chlorides ( 1 )
         density ( 1 )
         sulphates ( 1 )
         alcohol ( 1 )
The following features were NOT selected  (ranking):
         citric acid ( 6 )
         residual sugar ( 2 )
         free sulfur dioxide ( 5 )
         total sulfur dioxide ( 3 )
         pH ( 4 )
```

2. Apply RFE with Logistic Regression for selecting the best features

In [241]:

```
# create a base classifier used to evaluate a subset of attributes
estimatorLR = LogisticRegression(solver='lbfgs', multi_class='auto')
# create the RFE model and select 3 attributes
selectorLR = RFE(estimatorLR, num_features)
selectorLR = selectorLR.fit(predictors_train, target_train)
# summarize the selection of the attributes
print(selectorLR.support_)
print(selectorLR.ranking_)
```

```
[False  True  True False False False  True  True False  True  True]
[2 1 1 3 5 4 1 1 6 1 1]
```

In [242]:

```
print_feature_selection(selectorLR.support_, selectorLR.ranking_, t1_df)
```

```
The following features were selected (ranking):
         volatile acidity ( 1 )
         citric acid ( 1 )
         total sulfur dioxide ( 1 )
         density ( 1 )
         sulphates ( 1 )
         alcohol ( 1 )
The following features were NOT selected  (ranking):
         fixed acidity ( 2 )
         residual sugar ( 3 )
         chlorides ( 5 )
         free sulfur dioxide ( 4 )
         pH ( 6 )
```

B. Evaluate on the Test Data Set

Apply the selectors to prepare training data sets only with the selected features

Note: The same selectors are applied to the test data set. However, it is important that the test data set was not used by (it's invisible to) the selectors.

In [243]:

```
predictors_train_SVMselected = selectorSVM.transform(predictors_train)
predictors_test_SVMselected = selectorSVM.transform(predictors_test)
```

In [244]:

```
predictors_train_LRselected = selectorLR.transform(predictors_train)
predictors_test_LRselected = selectorLR.transform(predictors_test)
```

Train and evaluate SVM classifiers with both the selected features and all features

Here we train three models:

- model1 - with the features selected by SVM
- model2 - with the features selected by Logistic Regression
- model3 - with all features (i.e. without feature selection)

In [245]:

```
classifier = svm.SVC()
```

In [246]:

```
model1 = classifier.fit(predictors_train_SVMselected, target_train)
model1.score(predictors_test_SVMselected, target_test)
```

```
C:\Users\mpower1\AppData\Local\Continuum\anaconda3\lib\site-packages\sklea
rn\svm\base.py:196: FutureWarning: The default value of gamma will change
from 'auto' to 'scale' in version 0.22 to account better for unscaled feat
ures. Set gamma explicitly to 'auto' or 'scale' to avoid this warning.
  "avoid this warning.", FutureWarning)
```

Out[246]:

```
0.6
```

In [247]:

```
model2 = classifier.fit(predictors_train_LRselected, target_train)
model2.score(predictors_test_LRselected, target_test)
```

```
C:\Users\mpower1\AppData\Local\Continuum\anaconda3\lib\site-packages\sklea
rn\svm\base.py:196: FutureWarning: The default value of gamma will change
from 'auto' to 'scale' in version 0.22 to account better for unscaled feat
ures. Set gamma explicitly to 'auto' or 'scale' to avoid this warning.
  "avoid this warning.", FutureWarning)
```

Out[247]:

```
0.6125
```

In [248]:

```
model3 = classifier.fit(predictors_train, target_train)
model3.score(predictors_test, target_test)
```

```
C:\Users\mpower1\AppData\Local\Continuum\anaconda3\lib\site-packages\sklea
rn\svm\base.py:196: FutureWarning: The default value of gamma will change
from 'auto' to 'scale' in version 0.22 to account better for unscaled feat
ures. Set gamma explicitly to 'auto' or 'scale' to avoid this warning.
  "avoid this warning.", FutureWarning)
```

Out[248]:

```
0.609375
```

Task 1 Conclusions

With the seeds dataset, there were only two output labels (0/1) but with the wine dataset there are now 6 output labels (3,4,5,6,7,8) which means the classification task has become a multiclass classification task.

The output classes are also not very well balanced with over 80% of the output labels being either 5 or 6. There are 11 feature attributes that are available to make the prediction.

As there are more features, I increased the number of features to be selected to 6 (although this can now be easily varied using a num_features variable). I also added a function to display what features were selected or not selected and how they were ranked.

Using this with num_features=6, SVM and LR both chose a different set of 6 features. Interestingly, both methods also had different lowest rank features (citric acid for SVM and pH for LR).

The three models performed similarly with a score of ~60-61%. The model that used 6 features chosen by RFE with Logisitic Regression peformed best.

Features selected by SVM Selector:

The following features were selected (ranking):

- alcohol (1)
- chlorides (1)
- density (1)
- fixed acidity (1)
- sulphates (1)
- volatile acidity (1)

The following features were NOT selected (ranking):

- residual sugar (2)
- total sulfur dioxide (3)
- pH (4)
- free sulfur dioxide (5)
- citric acid (6)

Features selected by Logistic Regression Selector:

The following features were selected (ranking):

- alcohol (1)
- citric acid (1)
- density (1)
- sulphates (1)
- total sulfur dioxide (1)
- volatile acidity (1)

The following features were NOT selected (ranking):

- fixed acidity (2)
- residual sugar (3)
- free sulfur dioxide (4)
- chlorides (5)
- pH (6)

Task 2

In [249]:

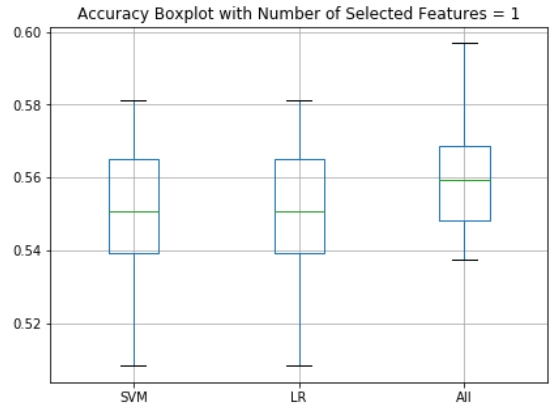
```
# Test Split Sizes
splits = np.arange(0.1,0.6,0.05)
print(splits)

# Number of Features to be selected
nfeatures = np.arange(1,12,1)
print(nfeatures)
```

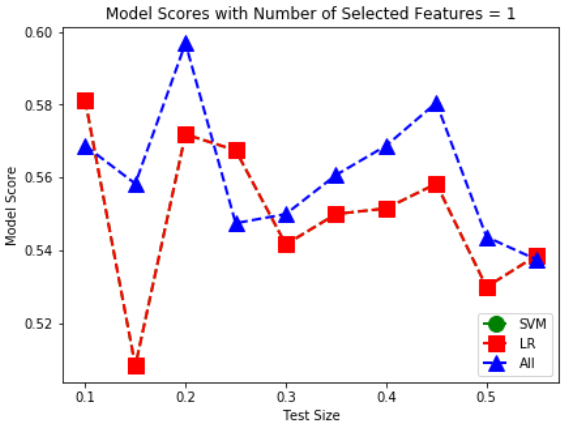
```
[0.1  0.15 0.2  0.25 0.3  0.35 0.4  0.45 0.5  0.55]
[ 1  2  3  4  5  6  7  8  9 10 11]
```

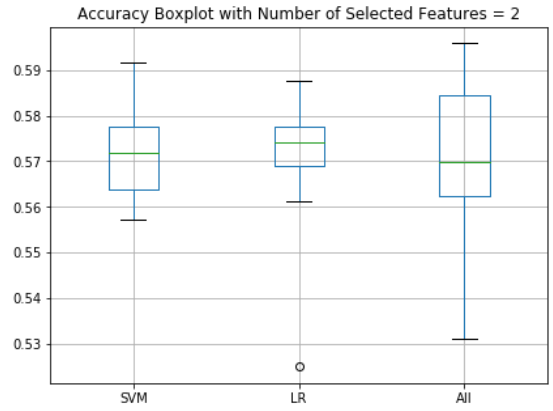
In [250]:

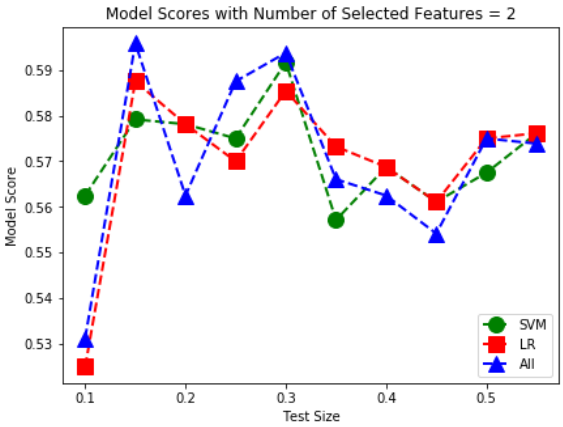
```
# Use these arrays to store the mean scores of each model across all number of selected
features
model1_mean_scores = []
model2_mean_scores = []
model3_mean_scores = []

# Iterate across all values of selected features
for n in range(len(nfeatures)):
    # Initialize empty arrays to store scores on each iteration of selected features
    model1_scores = []
    model2_scores = []
    model3_scores = []

    # Initialize empty arrays to store sum scores on each iteration of selected feature
s
    model1_sum_scores = 0
    model2_sum_scores = 0
    model3_sum_scores = 0

    # Iterate across all values of test size
    for i in range (len(splits)):
        # pepare independent stratified data sets for training and test of the final mo
del
        t2_predictors_train, t2_predictors_test, t2_target_train, t2_target_test = trai
n_test_split(
            predictors, target, test_size=splits[i], shuffle=True, stratify=target)

        # create a base classifier used to evaluate a subset of attributes
        estimatorSVM = svm.SVR(kernel="linear")
        selectorSVM = RFE(estimatorSVM, nfeatures[n])
        selectorSVM = selectorSVM.fit(t2_predictors_train, t2_target_train)

        # create a base classifier used to evaluate a subset of attributes
        estimatorLR = LogisticRegression(solver='lbfgs', multi_class='auto')
        # create the RFE model and select 3 attributes
        selectorLR = RFE(estimatorLR, nfeatures[n])
        selectorLR = selectorLR.fit(t2_predictors_train, t2_target_train)

        predictors_train_SVMselected = selectorSVM.transform(t2_predictors_train)
        predictors_test_SVMselected = selectorSVM.transform(t2_predictors_test)

        predictors_train_LRselected = selectorLR.transform(t2_predictors_train)
        predictors_test_LRselected = selectorLR.transform(t2_predictors_test)

        classifier = svm.SVC(gamma='auto')

        model1 = classifier.fit(predictors_train_SVMselected, t2_target_train)
        model1_scores.append(model1.score(predictors_test_SVMselected, t2_target_test))
        model1_sum_scores += model1.score(predictors_test_SVMselected, t2_target_test)

        model2 = classifier.fit(predictors_train_LRselected, t2_target_train)
        model2_scores.append(model2.score(predictors_test_LRselected, t2_target_test))
        model2_sum_scores += model2.score(predictors_test_LRselected, t2_target_test)

        model3 = classifier.fit(t2_predictors_train, t2_target_train)
        model3_scores.append(model3.score(t2_predictors_test, t2_target_test))
        model3_sum_scores += model3.score(t2_predictors_test, t2_target_test)

    # Compare the statistics of the accuracies across all cross-validation folds
```

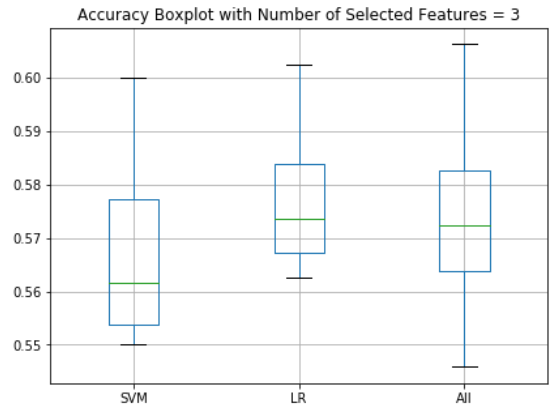
```
    accuracies_df = pd.DataFrame(data={'SVM' : model1_scores, 'LR' : model2_scores, 'Al
l' : model3_scores})

    plt.subplots(1, 2, figsize=(15, 5))

    plt.subplot(1,2,1)
    bp = accuracies_df.boxplot()
    plt.title('Accuracy Boxplot with Number of Selected Features = ' + str(nfeatures[n
]))

    plt.subplot(1,2,2)
    plt.plot(splits, model1_scores, color='green', marker='o', linestyle='dashed', line
width=2, markersize=12, label='SVM')
    plt.plot(splits, model2_scores, color='red', marker='s', linestyle='dashed', linewi
dth=2, markersize=12, label='LR')
    plt.plot(splits, model3_scores, color='blue', marker='^', linestyle='dashed', linew
idth=2, markersize=12, label='All')
    plt.title('Model Scores with Number of Selected Features = ' + str(nfeatures[n]))
    plt.xlabel('Test Size')
    plt.ylabel('Model Score')
    plt.legend(loc="lower right")

    plt.show()

    # Calculate mean score for each model for given value of selected features
    model1_mean_scores.append(model1_sum_scores/len(splits))
    model2_mean_scores.append(model2_sum_scores/len(splits))
    model3_mean_scores.append(model3_sum_scores/len(splits))
```

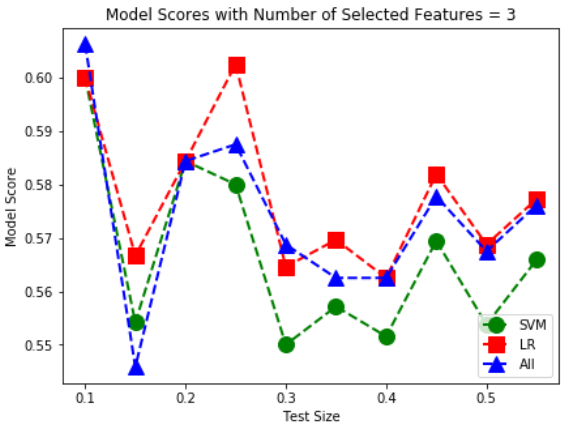




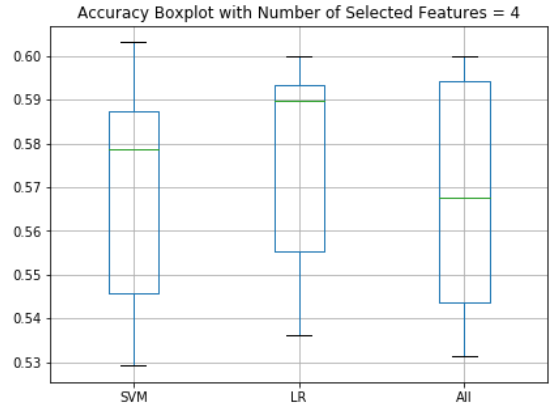


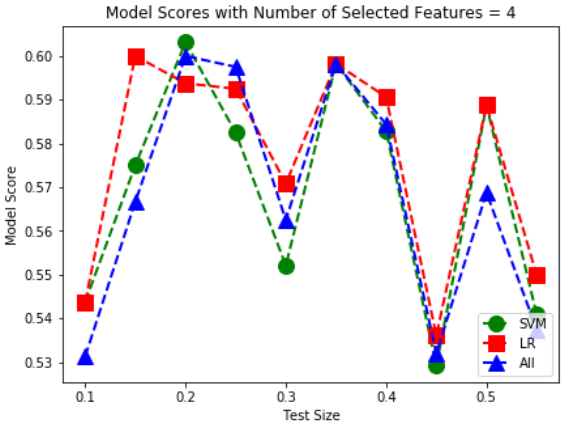


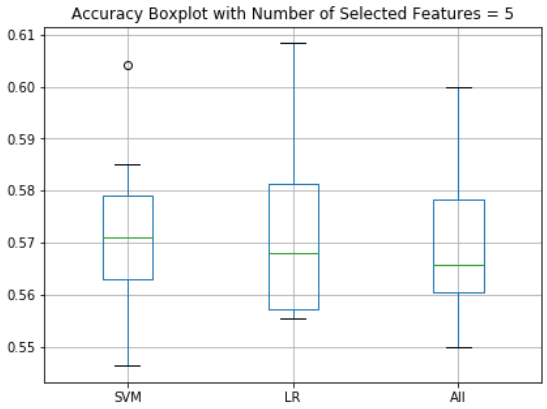


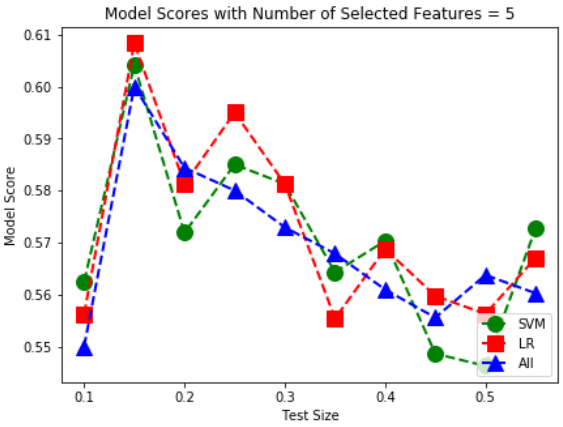


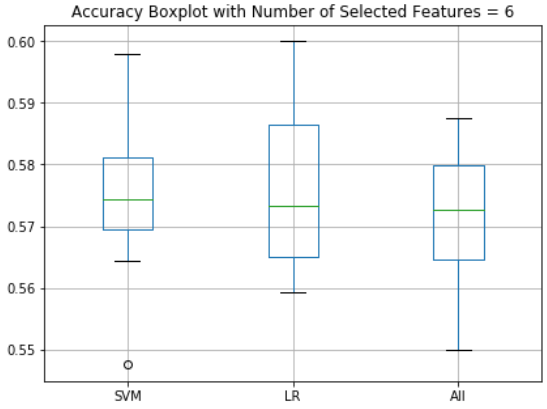

```
C:\Users\mpower1\AppData\Local\Continuum\anaconda3\lib\site-packages\sklea
rn\linear_model\logistic.py:758: ConvergenceWarning: lbfgs failed to conve
rge. Increase the number of iterations.
  "of iterations.", ConvergenceWarning)
```

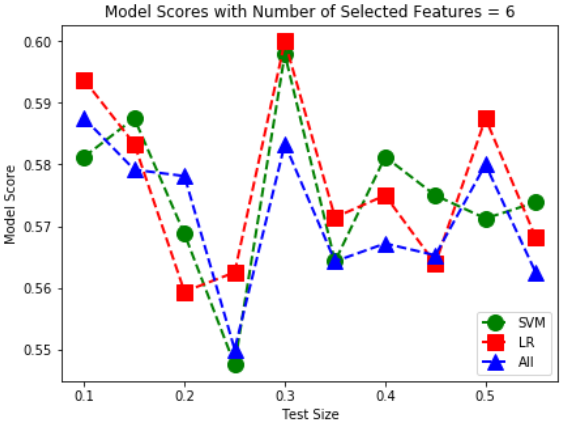


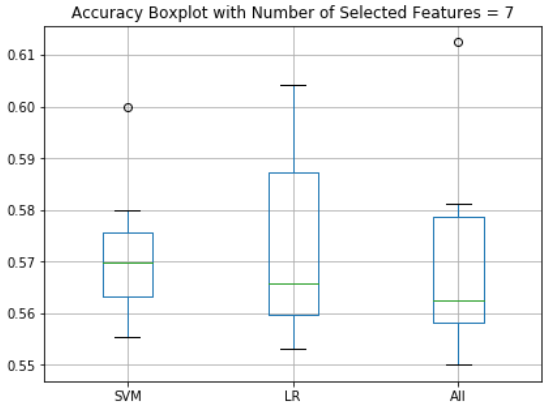


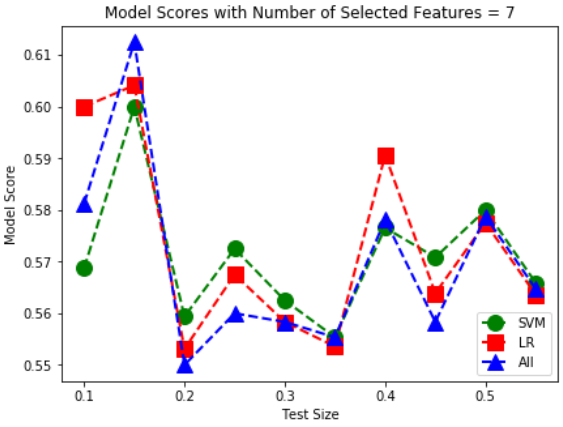

Accuracy Boxplot with Number of Selected Features = 5
SVM
LR
All

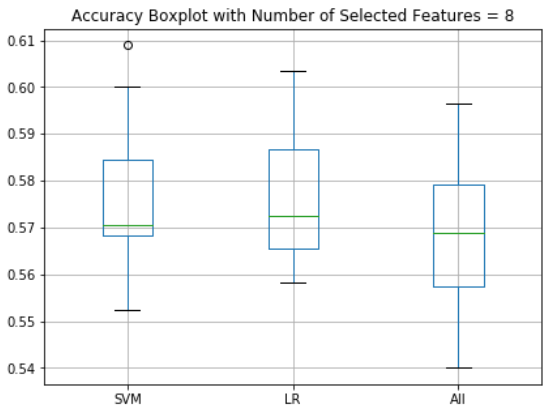

Model Scores with Number of Selected Features = 5
Model Score
Test Size
SVM
LR
All

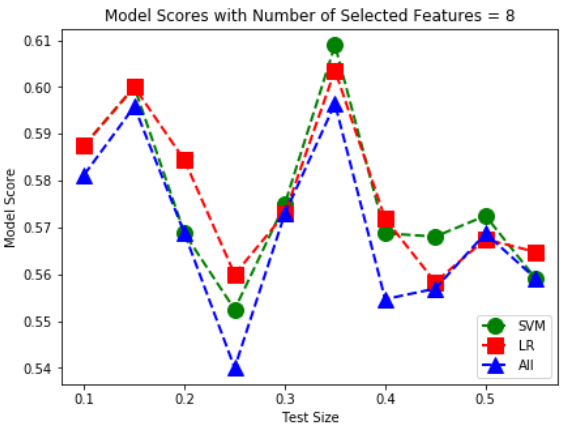

Accuracy Boxplot with Number of Selected Features = 6
SVM
LR
All

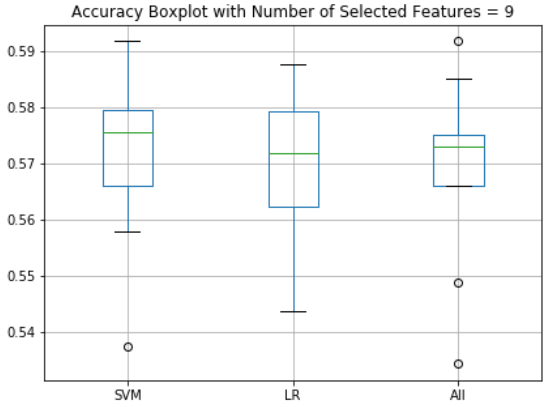

Model Scores with Number of Selected Features = 6
Model Score
Test Size
SVM
LR
All

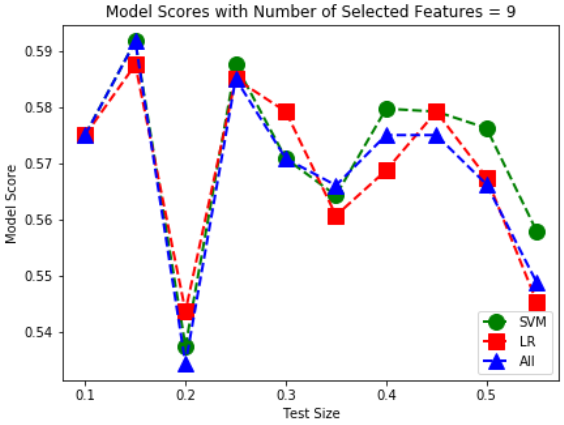

Accuracy Boxplot with Number of Selected Features = 7
SVM
LR
All

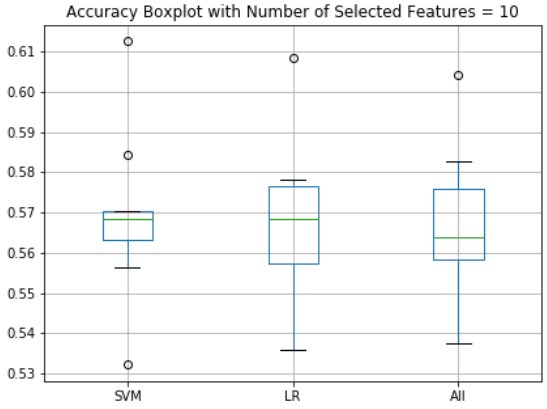

Model Scores with Number of Selected Features = 7
Model Score
Test Size
SVM
LR
All

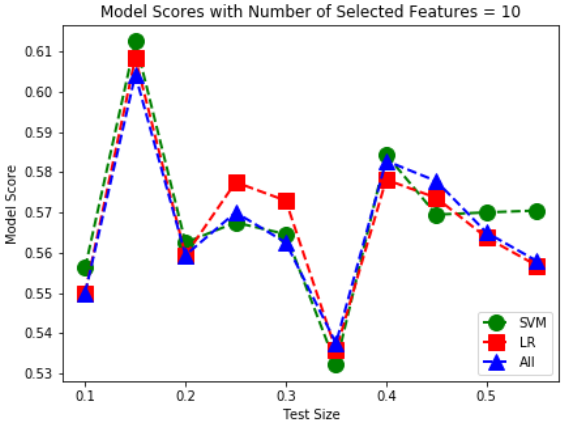

Accuracy Boxplot with Number of Selected Features = 8
SVM
LR
All

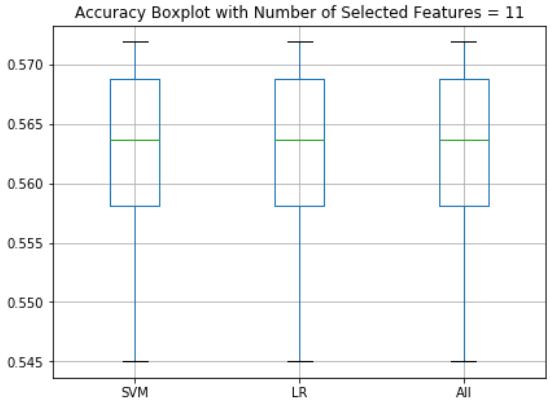

Model Scores with Number of Selected Features = 8
Model Score
Test Size
SVM
LR
All

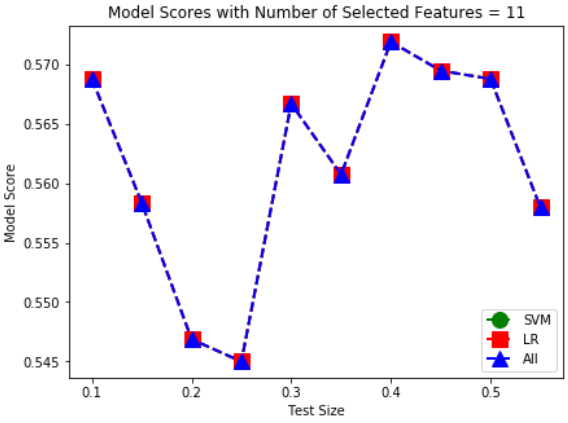

Accuracy Boxplot with Number of Selected Features = 9
0.59
0.58
0.57
0.56
0.55
0.54
SVM
LR
All


Model Scores with Number of Selected Features = 9
Model Score
0.59
0.58
0.57
0.56
0.55
0.54
0.1
0.2
0.3
0.4
0.5
Test Size
SVM
LR
All


Accuracy Boxplot with Number of Selected Features = 10
0.61
0.60
0.59
0.58
0.57
0.56
0.55
0.54
0.53
SVM
LR
All


Model Scores with Number of Selected Features = 10
Model Score
0.61
0.60
0.59
0.58
0.57
0.56
0.55
0.54
0.53
0.1
0.2
0.3
0.4
0.5
Test Size
SVM
LR
All


Accuracy Boxplot with Number of Selected Features = 11
0.570
0.565
0.560
0.555
0.550
0.545
SVM
LR
All


Model Scores with Number of Selected Features = 11
Model Score
0.570
0.565
0.560
0.555
0.550
0.545
0.1
0.2
0.3
0.4
0.5
Test Size
SVM
LR
All

In [251]:

```
mean_accuracies_df = pd.DataFrame(data={'SVM' : model1_mean_scores, 'LR' : model2_mean_
scores, 'All' : model3_mean_scores})
mean_accuracies_df.describe()
```

Out[251]:

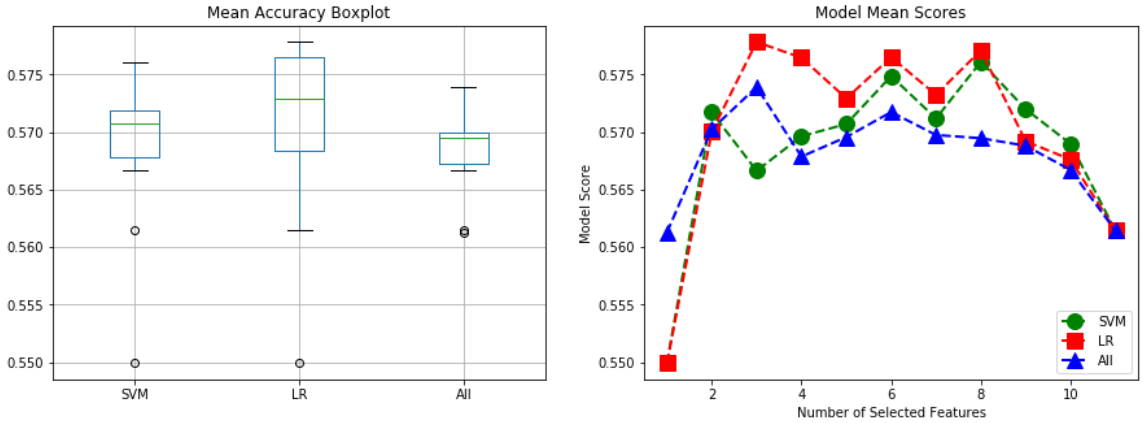
	SVM	LR	All
count	11.000000	11.000000	11.000000
mean	0.568466	0.570200	0.568251
std	0.007300	0.008348	0.003893
min	0.549916	0.549916	0.561273
25%	0.567805	0.568407	0.567289
50%	0.570698	0.572921	0.569465
75%	0.571847	0.576475	0.569993
max	0.576108	0.577824	0.573912

In [252]:

```
plt.subplots(1, 2, figsize=(15, 5))

plt.subplot(1,2,1)
mean_accuracies_df.boxplot()
plt.title('Mean Accuracy Boxplot')

plt.subplot(1,2,2)
plt.plot(nfeatures, model1_mean_scores, color='green', marker='o', linestyle='dashed',
linewidth=2, markersize=12, label='SVM')
plt.plot(nfeatures, model2_mean_scores, color='red', marker='s', linestyle='dashed', li
newidth=2, markersize=12, label='LR')
plt.plot(nfeatures, model3_mean_scores, color='blue', marker='^', linestyle='dashed', l
inewidth=2, markersize=12, label='All')
plt.title('Model Mean Scores')
plt.xlabel('Number of Selected Features')
plt.ylabel('Model Score')
plt.legend(loc="lower right")
```

Out[252]:

```
<matplotlib.legend.Legend at 0x23919e0a0b8>
```



Task 2 Conclustions

TBD

Task 3

In [253]:

```
from sklearn.pipeline import Pipeline
from sklearn.feature_selection import SelectPercentile
from sklearn.feature_selection import chi2
from sklearn.feature_selection import SelectKBest
```

Pipeline 1

In [254]:

```
# create pipeline
estimators = []
# Using Anova Univariate Feature Selection
# https://scikit-learn.org/stable/auto_examples/svm/plot_svm_anova.html#sphx-glr-auto-examples-svm-plot-svm-anova-py
estimators.append(('feat_sel', SelectPercentile(chi2)))
estimators.append(('scale',    preprocessing.MinMaxScaler()))
estimators.append(('classify', svm.SVC()))
p1_model = Pipeline(estimators)
p1_model.fit(predictors_train, target_train)
p1_model.score(predictors_test, target_test)
```

```
C:\Users\mpower1\AppData\Local\Continuum\anaconda3\lib\site-packages\sklearn\svm\base.py:196: FutureWarning: The default value of gamma will change
from 'auto' to 'scale' in version 0.22 to account better for unscaled features. Set gamma explicitly to 'auto' or 'scale' to avoid this warning.
  "avoid this warning.", FutureWarning)
```

Out[254]:

```
0.575
```

In [255]:

```
print(p1_model.steps)
```

```
[('feat_sel', SelectPercentile(percentile=10,
         score_func=<function chi2 at 0x00000239127109D8>)), ('scale', MinMaxScaler(copy=True, feature_range=(0, 1))), ('classify', SVC(C=1.0, cache_size=200, class_weight=None, coef0=0.0,
  decision_function_shape='ovr', degree=3, gamma='auto_deprecated',
  kernel='rbf', max_iter=-1, probability=False, random_state=None,
  shrinking=True, tol=0.001, verbose=False))]
```

Pipeline 2

In [256]:

```
# create pipeline
estimators = []
estimators.append(('feat_sel', SelectKBest(k=6)))
estimators.append(('scale',    preprocessing.MinMaxScaler()))
estimators.append(('classify', svm.SVC()))
p2_model = Pipeline(estimators)
p2_model.fit(predictors_train, target_train)
p2_model.score(predictors_test, target_test)
```

```
C:\Users\mpower1\AppData\Local\Continuum\anaconda3\lib\site-packages\sklea
rn\svm\base.py:196: FutureWarning: The default value of gamma will change
from 'auto' to 'scale' in version 0.22 to account better for unscaled feat
ures. Set gamma explicitly to 'auto' or 'scale' to avoid this warning.
  "avoid this warning.", FutureWarning)
```

Out[256]:

```
0.60625
```

In [257]:

```
print(p2_model.steps)
```

```
[('feat_sel', SelectKBest(k=6, score_func=<function f_classif at 0x0000023
9127108C8>)), ('scale', MinMaxScaler(copy=True, feature_range=(0, 1))),
('classify', SVC(C=1.0, cache_size=200, class_weight=None, coef0=0.0,
  decision_function_shape='ovr', degree=3, gamma='auto_deprecated',
  kernel='rbf', max_iter=-1, probability=False, random_state=None,
  shrinking=True, tol=0.001, verbose=False))]
```

Task 3 Conclustions

TBD

In [ ]:

```

```