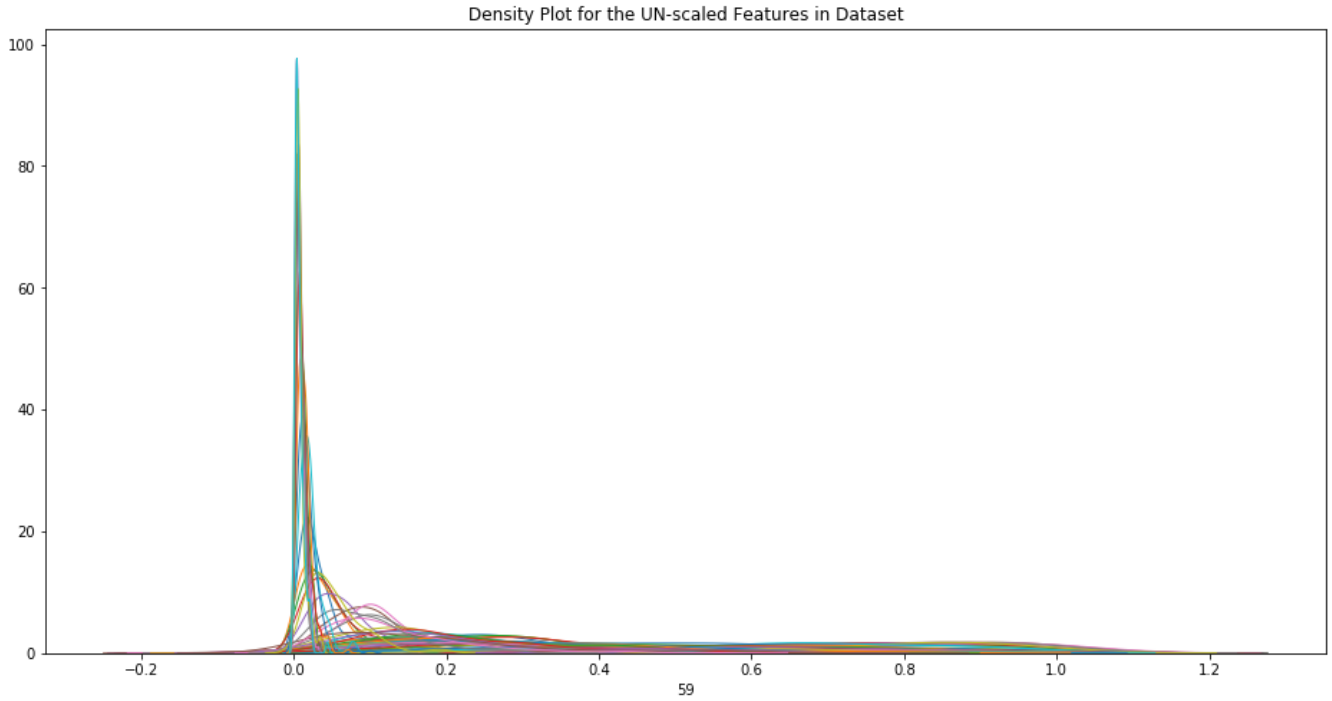
SVM on the Sonar Dataset

```
import matplotlib.pyplot as plt
## Importing required libraries
import numpy as np
%matplotlib inline
#%matplotlib notebook
import seaborn as sns
import pandas as pd
df = pd.read_csv('sonar.csv', header=None)
x_unscaled = df.sample(frac=1, replace=True, random_state=1)
y = x_unscaled[60]
x_unscaled.drop([60],axis=1, inplace=True)
x_unscaled.describe()
```

	0	1	2	3	4	5	6
count	208.000000	208.000000	208.000000	208.000000	208.000000	208.000000	208.000000
mean	0.030244	0.040169	0.045399	0.054121	0.070841	0.104084	0.123117
std	0.024354	0.034306	0.039356	0.048682	0.052852	0.057876	0.061085
min	0.001500	0.000600	0.001500	0.005800	0.007600	0.011600	0.013000
25%	0.013100	0.014600	0.017950	0.022900	0.035100	0.062900	0.087350
50%	0.022850	0.031800	0.034650	0.039900	0.060800	0.093200	0.105600
75%	0.040825	0.056300	0.060850	0.068800	0.088300	0.126175	0.149625
max	0.137100	0.233900	0.305900	0.426400	0.401000	0.307000	0.332200

8 rows × 60 columns

```
plt.figure(figsize=(16,8))
plt.title('Density Plot for the UN-scaled Features in Dataset')
for i in x_unscaled.columns:
    # Draw the density plot
    sns.distplot(x_unscaled[i], hist = False, kde = True,
                 kde_kws = {'linewidth': 1})
```



Scale the features

The SMV performs faster if features are scaled

```
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
scaler.fit(x_unscaled)
x = pd.DataFrame(scaler.transform(x_unscaled), index=x_unscaled.index, columns=x_unscaled.columns)
print("x shape: ",x.shape)
x.describe()
```

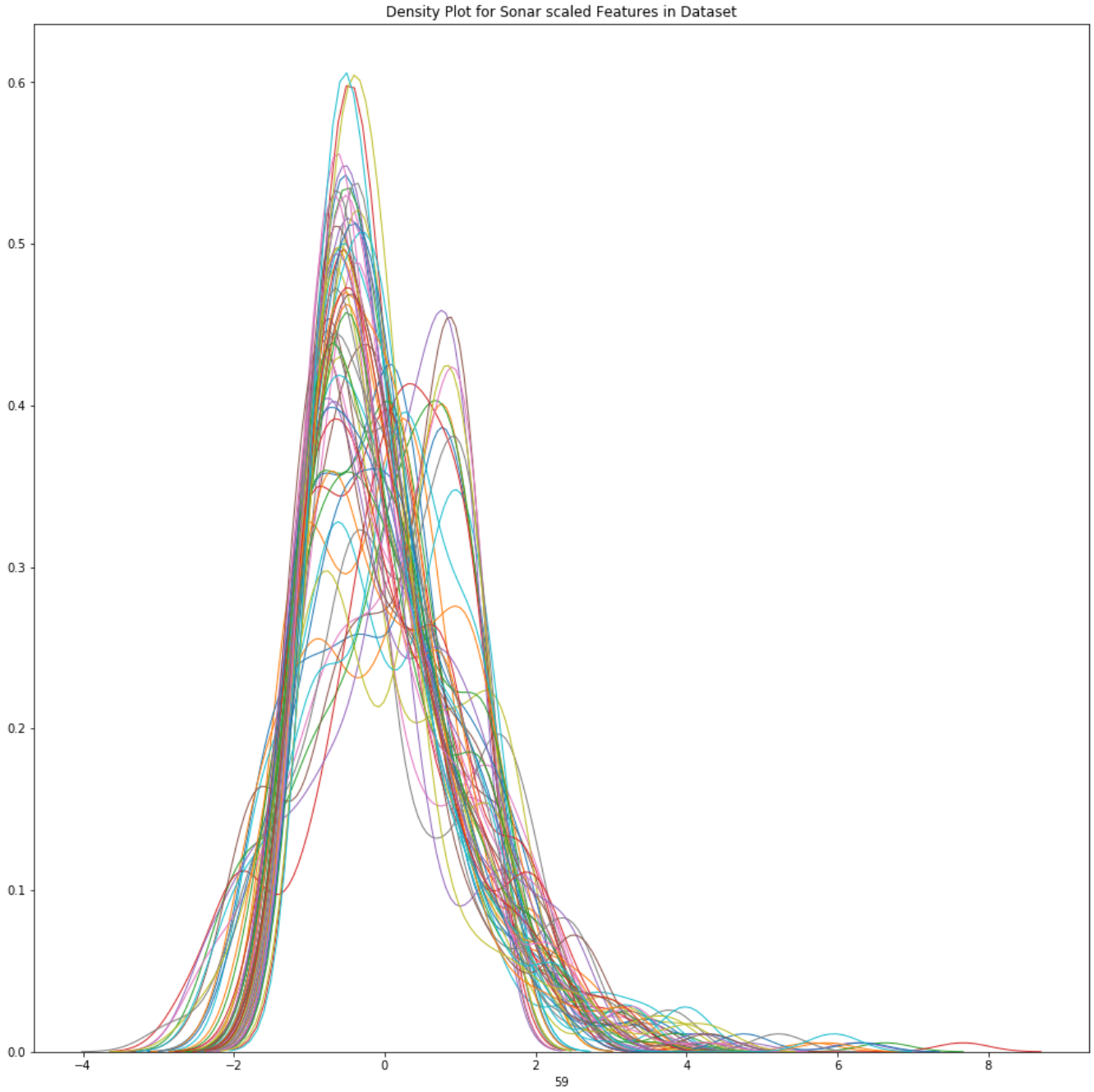
```
x shape:  (208, 60)
```

	0	1	2	3	4	5
count	2.080000e+02	2.080000e+02	2.080000e+02	2.080000e+02	2.080000e+02	2.080000e+02
mean	1.259676e-16	-1.708035e-17	-2.860959e-16	6.565261e-17	4.270089e-18	3.416071e-17
std	1.002413e+00	1.002413e+00	1.002413e+00	1.002413e+00	1.002413e+00	1.002413e+00
min	-1.183116e+00	-1.156214e+00	-1.118120e+00	-9.949667e-01	-1.199456e+00	-1.601829e+00
25%	-7.056588e-01	-7.471336e-01	-6.991299e-01	-6.428621e-01	-6.778783e-01	-7.133108e-01
50%	-3.043475e-01	-2.445492e-01	-2.737717e-01	-2.928166e-01	-1.904398e-01	-1.885136e-01
75%	4.355059e-01	4.713415e-01	3.935567e-01	3.022607e-01	3.311383e-01	3.826146e-01
max	4.398198e+00	5.660819e+00	6.635115e+00	7.665570e+00	6.261955e+00	3.514510e+00

8 rows × 60 columns

```
plt.figure(figsize=(16,16))
plt.title('Density Plot for Sonar scaled Features in Dataset')
for i in x.columns:
```

```
    # Draw the density plot
    sns.distplot(x[i], hist = False, kde = True,
                 kde_kws = {'linewidth': 1})
```



Fitting a Support Vector Machine

Use Scikit-Learn's support vector classifier to train an SVM model on this data. Let'set the C parameter to a very large number.

```
from sklearn.svm import SVC # "Support vector classifier"
from timeit import default_timer as timer

model = SVC(kernel='linear', C=1E4)
start = timer()
model.fit(x, y)
end = timer()
print(end - start)
```

```
model1 = SVC(kernel='linear', C=1E4)
start = timer()
model1.fit(x_unscaled, y)
end = timer()
print(end - start)
```

```
0.006013008000081754
0.037372960000539024
```

```
print(len(model.support_vectors_))
#print(model.support_vectors_)

print(len(model1.support_vectors_))
#print(model1.support_vectors_)
```

```
52
60
```

The SVM fit performed on unscalsed features is 6 times slower and has 8 more support vectors

From the previous experiences with the Sonar Dataset (https://github.com/borodark/ie7860/blob/master/Feature%20Selection%20and%20Visualization%20Sonar%20Data%20Set.ipynb) we know that several features are more important. Here is 24 best Features by F score: [0 1 3 7 8 9 10 11 12 33 35 36 43 44 45 46 47 48 49 50 51 53 57 59]

Let's fit SVM on 2 and visualize the hyperplane

Fit SVM on 2 features and visualize the hyperplane

```
# define the function for plotting the decision boundary
def plot_svc_decision_function(model, ax=None, plot_support=True):
    """Plot the decision function for a 2D SVC"""
    if ax is None:
        ax = plt.gca()
    xlim = ax.get_xlim()
    ylim = ax.get_ylim()

    # Create grid to evaluate model
    x = np.linspace(xlim[0], xlim[1], 30)
    y = np.linspace(ylim[0], ylim[1], 30)
    Y, X = np.meshgrid(y, x)
    xy = np.vstack([X.ravel(), Y.ravel()]).T
    P = model.decision_function(xy).reshape(X.shape)

    # Plot decision boundary and margins
    ax.contour(X, Y, P, colors='k',
               levels=[-1, 0, 1], alpha=0.5,
               linestyles=['--', '-', '--'])

    # Plot support vectors
    if plot_support:
        ax.scatter(model.support_vectors_[:, 0],
                   model.support_vectors_[:, 1],
                   s=50, linewidth=1, color='#000000', facecolors='none');
    ax.set_xlim(xlim)
    ax.set_ylim(ylim)
```

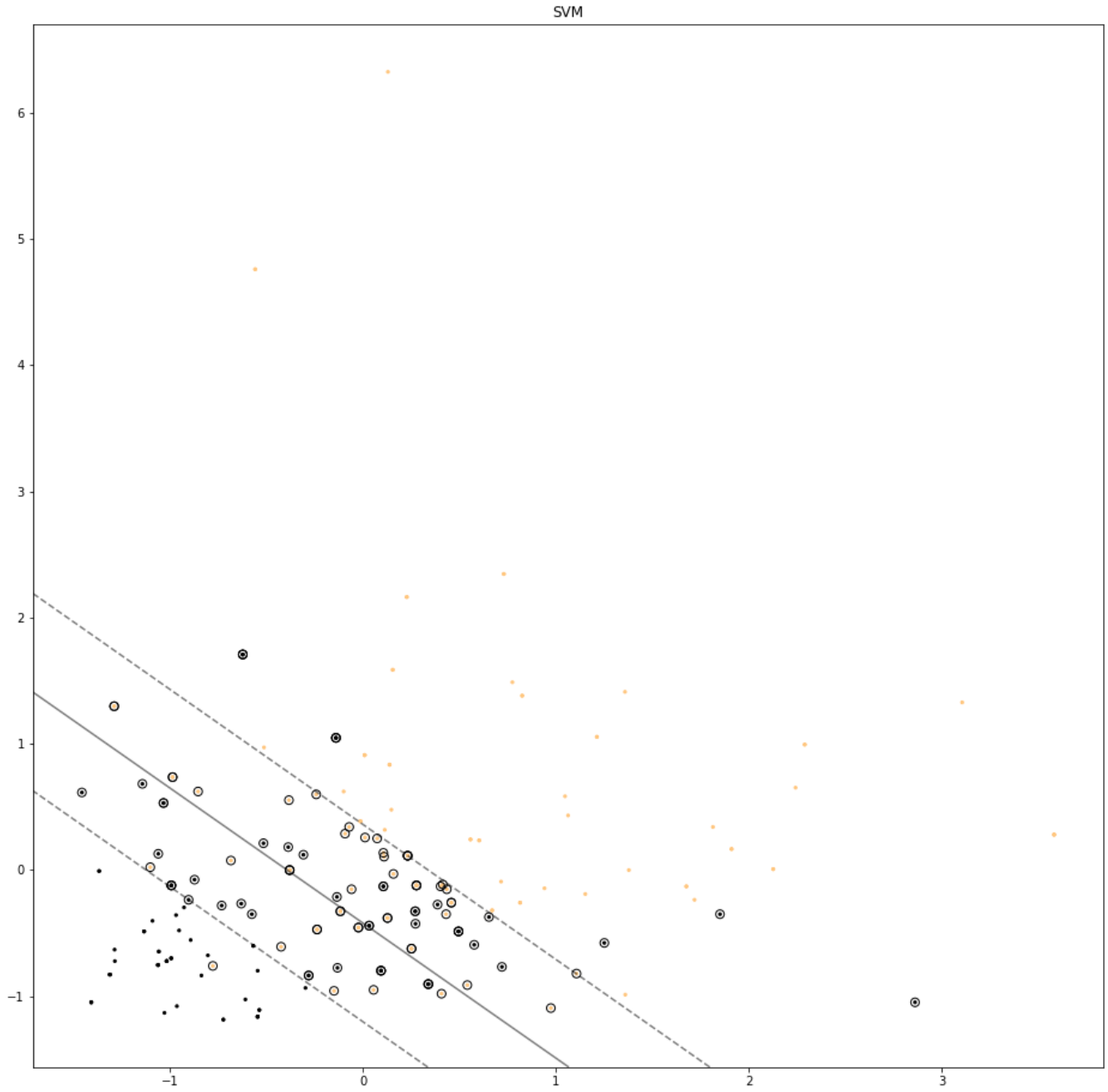
```
# fit SVM on features [10,50] -
X = x[[10,50]]
X1 = X.to_numpy()
```

```
model3 = SVC(kernel='linear', C=1E4)
start = timer()
model3.fit(X, y)
end = timer()
print(end - start)
```

```
0.20406295799966756
```

```
print("There are ", len(model3.support_vectors_), " support vectors ")
plt.figure(figsize=(16,16))
plt.title('SVM')
plt.scatter(X1[:, 0], X1[:, 1], c=y, s=5, cmap='copper')
plot_svc_decision_function(model3);
```

```
There are  99  support vectors
```



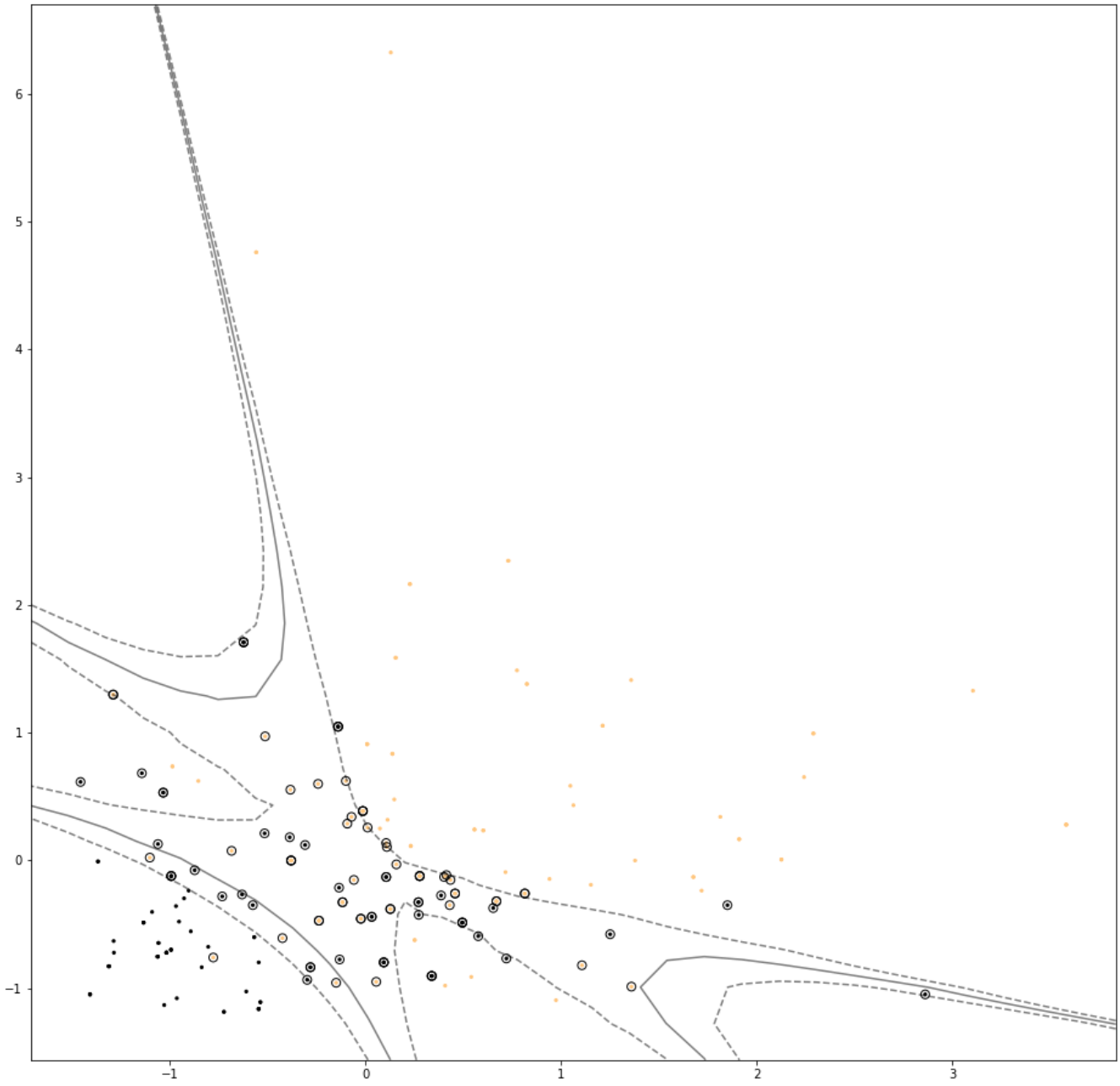
Use 'poly' kernel with SVM

```
poly = SVC(kernel='poly', C=1E4, gamma='scale')
poly.fit(X, y)
```

```
SVC(C=10000.0, break_ties=False, cache_size=200, class_weight=None, coef0=0.0,
    decision_function_shape='ovr', degree=3, gamma='scale', kernel='poly',
    max_iter=-1, probability=False, random_state=None, shrinking=True,
    tol=0.001, verbose=False)
```

```
print("There are ", len(poly.support_vectors_), " support vectors for poly kernel")
plt.figure(figsize=(16,16))
plt.scatter(X1[:, 0], X1[:, 1], c=y, s=5, cmap='copper')
plot_svc_decision_function(poly);
```

```
There are  97  support vectors for poly kernel
```



Comparing to other Classifiers

Scaled features dataset

```
# Data Processing
from sklearn.model_selection import train_test_split
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import StratifiedKFold

# Metrics
from sklearn.metrics import classification_report
from sklearn.metrics import confusion_matrix
from sklearn.metrics import accuracy_score

# Models
from sklearn.linear_model import LogisticRegression
```

```
from sklearn.tree import DecisionTreeClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis
from sklearn.naive_bayes import GaussianNB
```

```
# Create a validation dataset
# Split-out validation dataset
X_train, X_test, Y_train, Y_test = train_test_split(x, y, test_size=0.20, random_state=1)
X_train.head()
```

	0	1	2	3	4	5	6	7
23	-0.771515	-0.735446	-0.809927	-0.957903	-0.943409	0.029719	-0.341615	-1.032878
97	0.776106	-0.358507	0.351530	1.500652	2.017258	1.501922	1.617757	-0.647787
94	-1.141956	-0.270847	-0.720780	-0.644921	-0.520457	0.317231	1.174683	0.400647
47	0.290416	-0.352663	-0.565409	-0.651098	-1.004102	-0.533182	-0.640280	-0.339009
10	-1.084332	-0.989660	-0.769174	-0.422539	-0.755641	-1.310852	-1.370532	-1.230119

5 rows × 60 columns

```
X_test.head()
```

	0	1	2	3	4	5	6	
24	-0.038865	0.708024	-0.162975	-0.758171	-0.440797	-0.389426	-0.390846	-1.17963
54	-0.701543	-0.939986	-0.677480	-0.824062	-0.516663	-0.645763	-1.020995	-1.39566
114	-0.775631	-0.525061	-0.471168	-0.323703	0.964619	1.768651	0.423098	-0.45876
205	0.903702	0.103169	-0.697856	-0.513139	-0.677878	0.225435	0.042382	-0.16642
115	0.459173	0.100247	-0.017793	0.623479	-0.548906	0.301643	1.264939	0.23040

5 rows × 60 columns

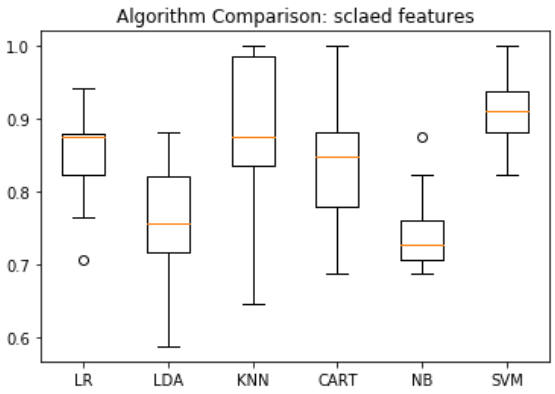
```
# Spot Check Algorithms
models = []
models.append(('LR', LogisticRegression(solver='liblinear', multi_class='ovr')))
models.append(('LDA', LinearDiscriminantAnalysis()))
models.append(('KNN', KNeighborsClassifier()))
models.append(('CART', DecisionTreeClassifier()))
models.append(('NB', GaussianNB()))
models.append(('SVM', SVC(gamma='auto')))

# Evaluate each model in turn
results = []
names = []
for name, model in models:
    kfold = StratifiedKFold(n_splits=10)
    cv_results = cross_val_score(model, X_train, Y_train, cv=kfold, scoring='accuracy')
    results.append(cv_results)
    names.append(name)
    print('%s: %f (%f)' % (name, cv_results.mean(), cv_results.std()))
```

```
LR: 0.850368 (0.069524)
LDA: 0.758824 (0.079809)
KNN: 0.880147 (0.108518)
CART: 0.842647 (0.087987)
```

```
NB: 0.747426 (0.057335)
SVM: 0.904779 (0.053212)
```

```
# Compare Algorithms
plt.boxplot(results, labels=names)
plt.title('Algorithm Comparison: scaled features')
plt.show()
```



SVM has the highest median out of all!

```
# Make predictions on validation dataset
model = SVC(gamma='auto',C=1E10)
start = timer()
model.fit(X_train, Y_train)
end = timer()
print(end - start)
predictions = model.predict(X_test)
```

```
0.004186975999800779
```

```
# Evaluate predictions
print("There are ", len(model.support_vectors_), " support vectors ")
print(accuracy_score(Y_test, predictions))
print(classification_report(Y_test, predictions))
cm = confusion_matrix(Y_test,predictions)
print('Confusion Matrix:')
sns.heatmap(cm,annot=True,fmt="d", annot_kws={"size": 12}) # font size
```

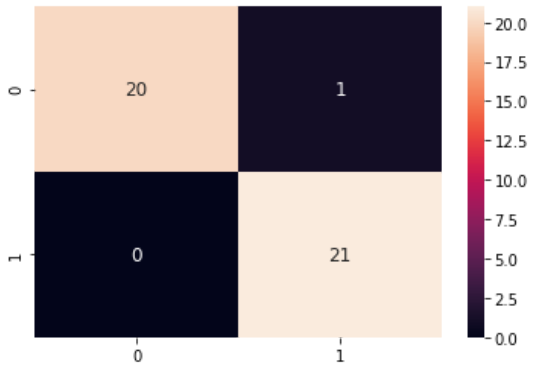
```
There are  91  support vectors
0.9761904761904762
              precision    recall  f1-score   support

           0       1.00      0.95      0.98        21
           1       0.95      1.00      0.98        21

    accuracy                           0.98        42
   macro avg       0.98      0.98      0.98        42
weighted avg       0.98      0.98      0.98        42

Confusion Matrix:
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x123ae5850>
```



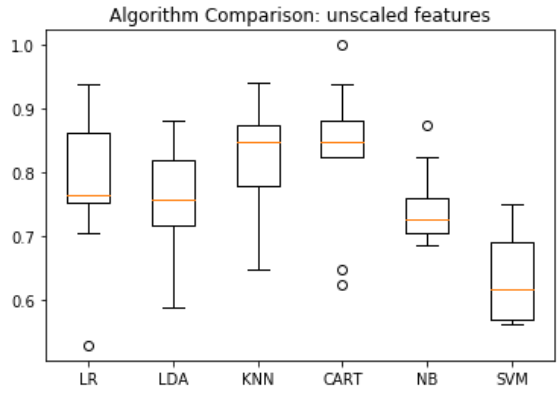
Unscaled features

```
# Create a validation dataset
# Split-out validation dataset
X_train, X_test, Y_train, Y_test = train_test_split(x_unscaled, y, test_size=0.20, random_state=1)
# Spot Check Algorithms
models = []
models.append(('LR', LogisticRegression(solver='liblinear', multi_class='ovr')))
models.append(('LDA', LinearDiscriminantAnalysis()))
models.append(('KNN', KNeighborsClassifier()))
models.append(('CART', DecisionTreeClassifier()))
models.append(('NB', GaussianNB()))
models.append(('SVM', SVC(gamma='auto')))

# Evaluate each model in turn
results = []
names = []
for name, model in models:
    kfold = StratifiedKFold(n_splits=10)
    cv_results = cross_val_score(model, X_train, Y_train, cv=kfold, scoring='accuracy')
    results.append(cv_results)
    names.append(name)
    print('%s: %f (%f)' % (name, cv_results.mean(), cv_results.std()))
```

```
LR: 0.785294 (0.114080)
LDA: 0.758824 (0.079809)
KNN: 0.819485 (0.088426)
CART: 0.831250 (0.110838)
NB: 0.747426 (0.057335)
SVM: 0.631985 (0.066043)
```

```
# Compare Algorithms
plt.boxplot(results, labels=names)
plt.title('Algorithm Comparison: unscaled features')
plt.show()
```



Conclusion

- The SVM classifier is able to reach %98 accuracy and performs faster then MLP - fit takes only `0.003992759000539081` seconds compare to several minutes for Neural Networks
- The SVM classifier is sensitive to feature sclaing: slower speed and lower precision