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[[10,50]].describe()

10

50

count

208.000000

208.000000

mean

0.232959

0.017114

std

0.140313

0.013203

min

0.028900

0.001500

25%

0.119125

0.007900

50%

0.230200

0.014450

75%

0.294425

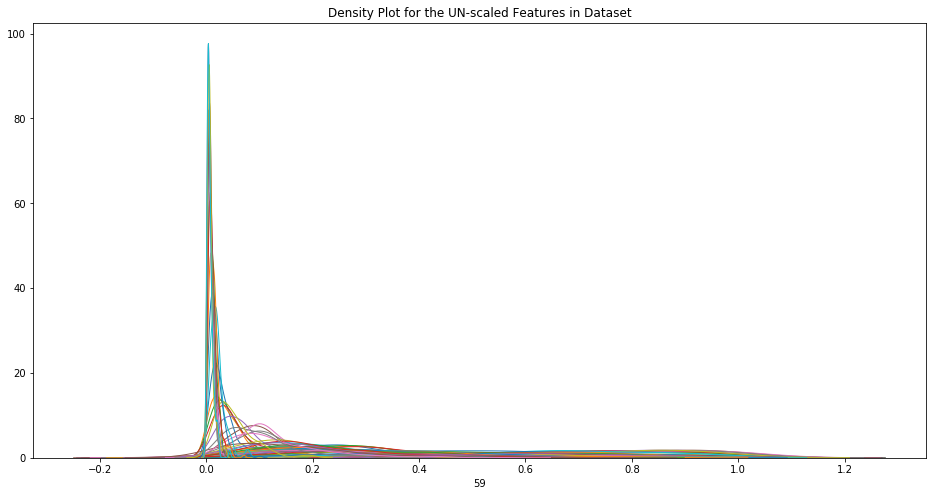
0.021600

max

0.734200

0.100400

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})



png

## 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[[10,50]].describe()

x shape: (208, 60)

10

50

count

2.080000e+02

2.080000e+02

mean

-2.989062e-17

1.024821e-16

std

1.002413e+00

1.002413e+00

min

-1.457818e+00

-1.185540e+00

25%

-8.132420e-01

-6.996136e-01

50%

-1.971152e-02

-2.022988e-01

75%

4.391181e-01

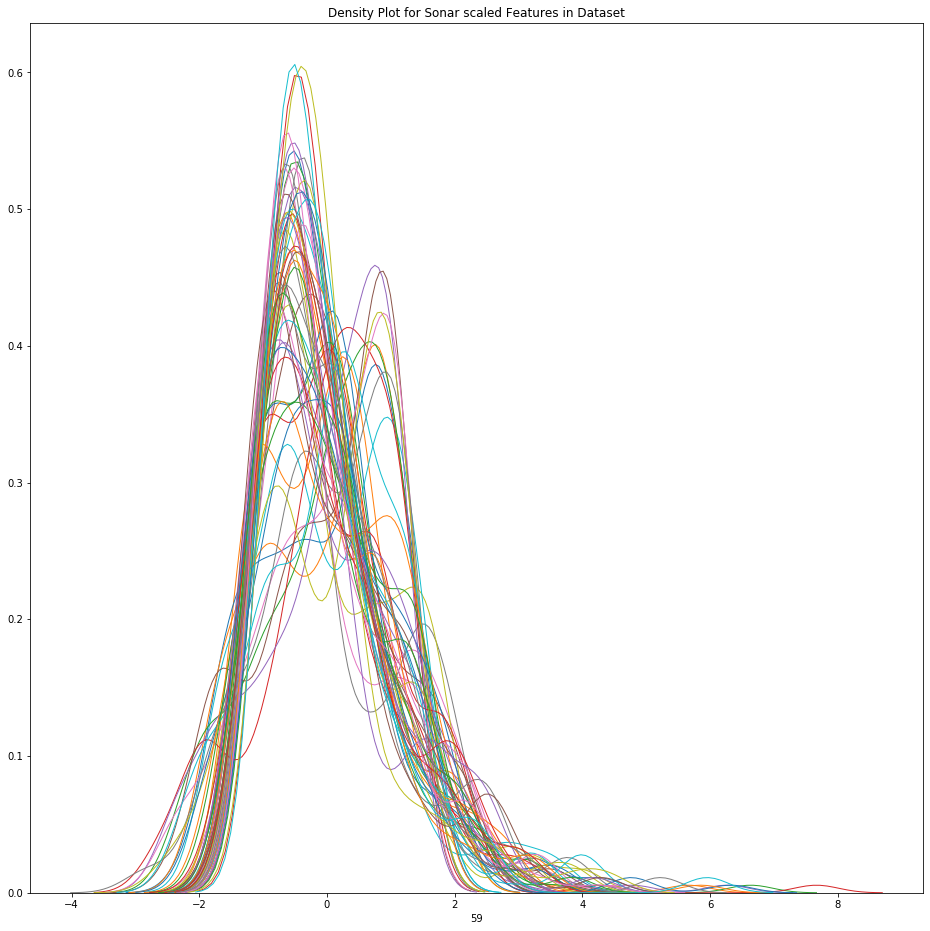
3.405716e-01

max

3.580913e+00

6.323535e+00

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})



png

## 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)

0.005107619999762392  
0.03811032199973852

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)

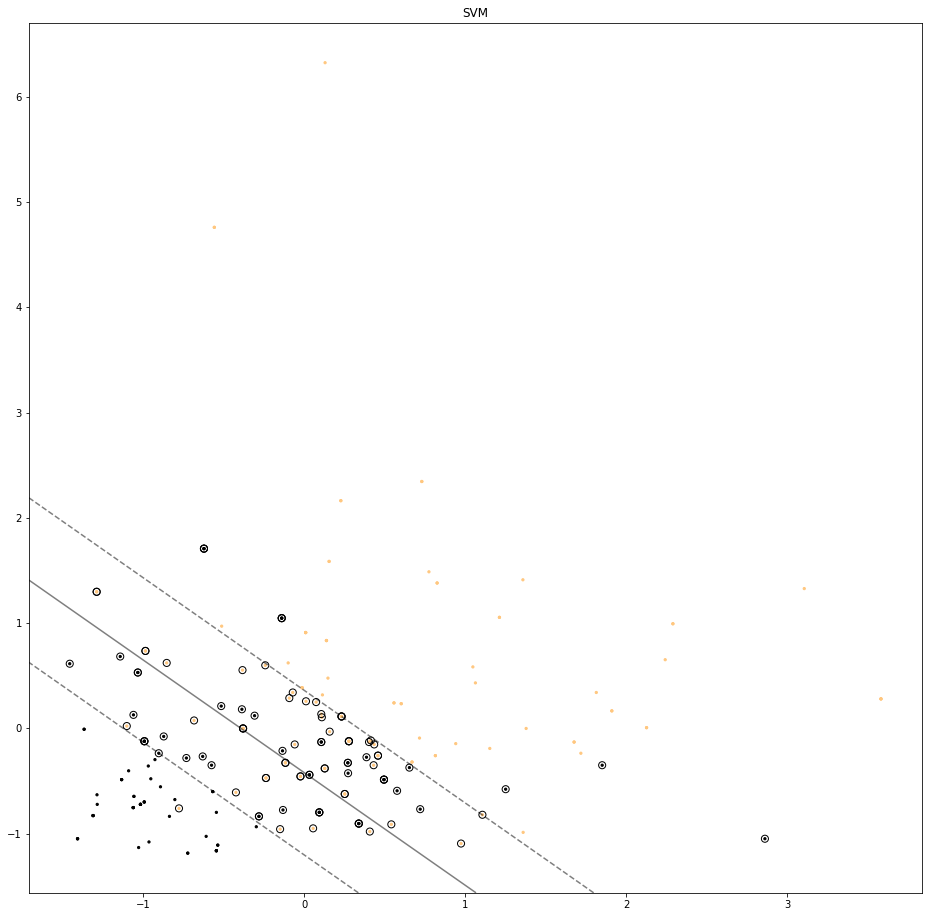
# fit SVM on features [10,50] -   
X = x[[10,50]]  
X1 = X.to\_numpy()

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

0.2013787650002996

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



png

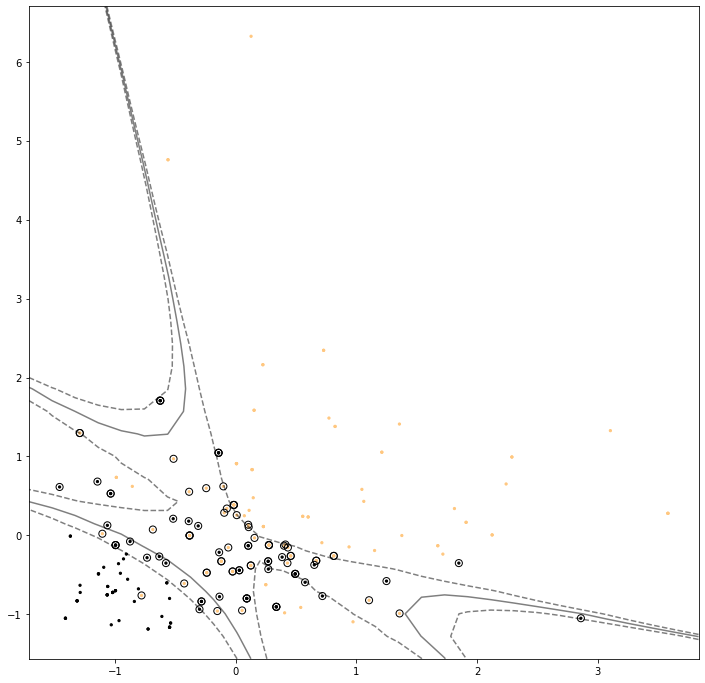
## Use ‘poly’ kernel with SVM

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=(12,12))  
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



png

## 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[[10,50]].head()

10

50

23

-1.135619

-0.487021

97

-0.987737

0.735386

94

-0.954159

-0.479428

47

-0.546946

-0.798317

10

-1.312793

-0.828688

X\_test[[10,50]].head()

10

50

24

-1.368517

-0.008688

54

-0.631961

-0.266836

114

0.111740

0.317794

205

0.276054

-0.122577

115

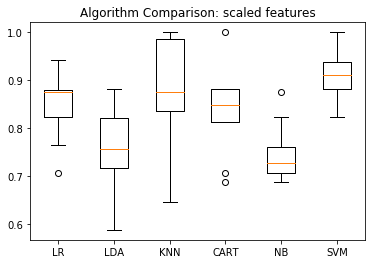
-0.014711

0.386127

# 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: 0.836397 (0.086663)  
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()



png

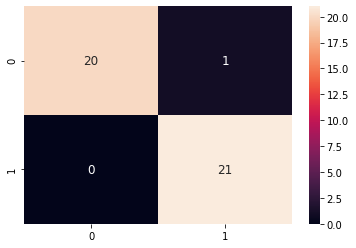
SVM has the highest median out of all!

# 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)

0.003969833999690309

# 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 0x125836210>



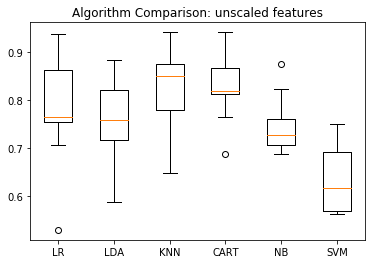
png

### 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: 0.824265 (0.065639)  
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()



png

### 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