Behaviour of Linear Models

April 7, 2021

1 Importing all necessary packages

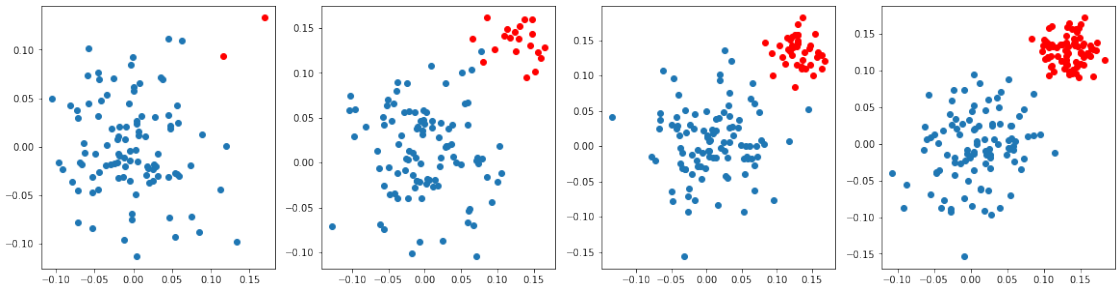
[1]:
```
import numpy as np
import pandas as pd

from sklearn.svm import SVC
from sklearn.datasets import load_iris
from sklearn.datasets import make_classification
from sklearn.linear_model import SGDClassifier
from sklearn.linear_model import SGDRegressor
from sklearn.linear_model import LogisticRegression
from sklearn.preprocessing import Normalizer
from sklearn.preprocessing import StandardScaler
from sklearn.preprocessing import MinMaxScaler
from sklearn.model_selection import GridSearchCV
from sklearn.model_selection import train_test_split

import scipy as sp
import scipy.optimize

%matplotlib inline
import matplotlib.pyplot as plt
import plotly
import plotly.figure_factory as ff
import plotly.graph_objs as go
from plotly.offline import download_plotlyjs, init_notebook_mode, plot, iplot
import seaborn as sns

import warnings
warnings.filterwarnings("ignore")
init_notebook_mode(connected=True)
```

2 Task A:

2.1 Imbalance Data and it's effects on SVM and Logistic Regression

[2]:
```
def draw_line(coef,intercept, mi, ma):
    # for the separating hyper plane ax+by+c=0, the weights are [a, b] and the intercept is c
    # to draw the hyper plane we are creating two points
    # 1. ((b*min-c)/a, min) i.e ax+by+c=0 ==> ax = (-by-c) ==> x = (-by-c)/a here in place of y we are keeping the minimum value of y
    # 2. ((b*max-c)/a, max) i.e ax+by+c=0 ==> ax = (-by-c) ==> x = (-by-c)/a here in place of y we are keeping the maximum value of y
    points=np.array([[((-coef[1]*mi - intercept)/coef[0]), mi],[((-coef[1]*ma - intercept)/coef[0]), ma]])
    plt.plot(points[:,0], points[:,1])
```

[3]:
```
# here we are creating 2d imbalanced data points
ratios = [(100,2), (100, 20), (100, 40), (100, 80)]
plt.figure(figsize=(20,5))
for j,i in enumerate(ratios):
    plt.subplot(1, 4, j+1)
    X_p=np.random.normal(0,0.05,size=(i[0],2))
    X_n=np.random.normal(0.13,0.02,size=(i[1],2))
    y_p=np.array([1]*i[0]).reshape(-1,1)
    y_n=np.array([0]*i[1]).reshape(-1,1)
    X=np.vstack((X_p,X_n))
    y=np.vstack((y_p,y_n))
    plt.scatter(X_p[:,0],X_p[:,1])
    plt.scatter(X_n[:,0],X_n[:,1],color='red')
plt.show()
```

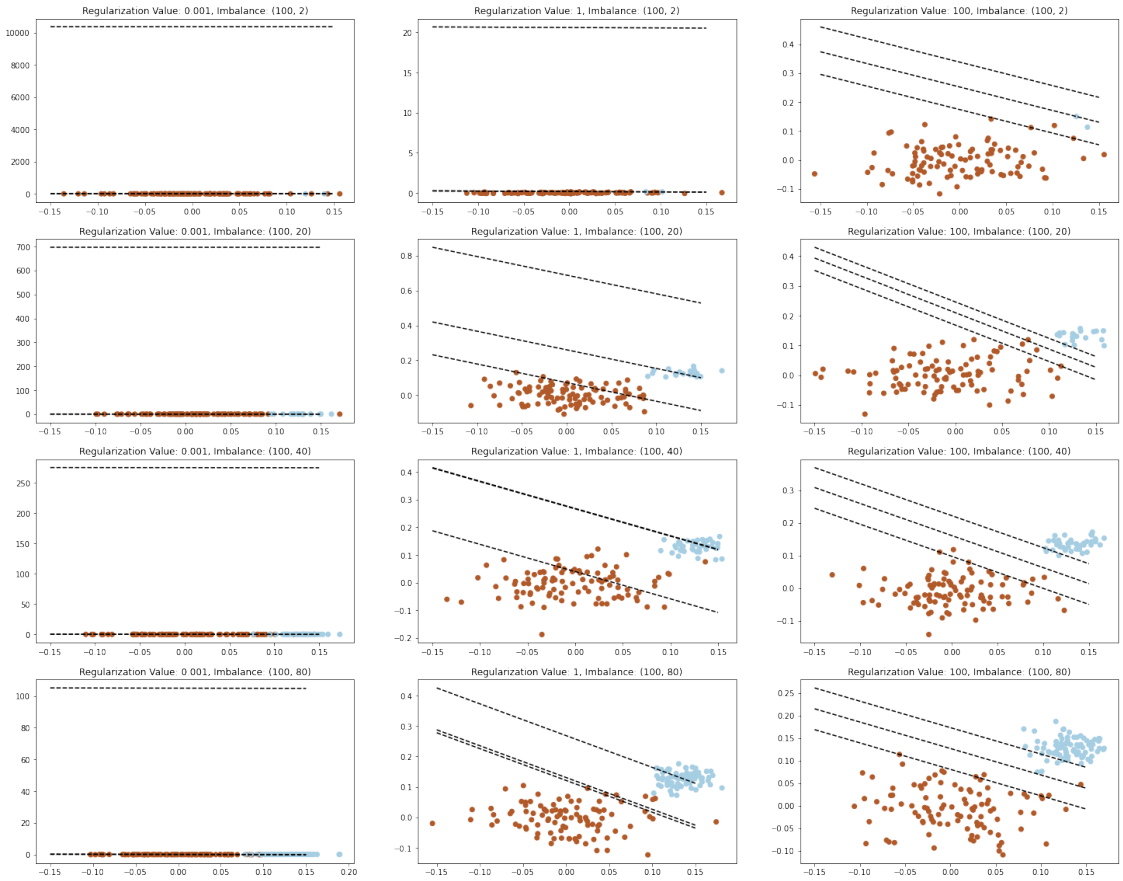


2.1.1 Applying on SVM

[4]:
```
# ref: https://scikit-learn.org/0.15/auto_examples/svm/
 ↪plot_separating_hyperplane.html

reg_strengths = [0.001,1,100]
ratios = [(100,2), (100, 20), (100, 40), (100, 80)]
plt.figure(figsize=(20,5))
fig, axs = plt.subplots(4, 3, figsize=(25, 20))
for p,r in enumerate(reg_strengths):
    for j,i in enumerate(ratios):
        # plt.subplot(4, 3, j+1)

        X_p=np.random.normal(0,0.05,size=(i[0],2))
        X_n=np.random.normal(0.13,0.02,size=(i[1],2))
        y_p=np.array([1]*i[0]).reshape(-1,1)
        y_n=np.array([0]*i[1]).reshape(-1,1)
        X=np.vstack((X_p,X_n))
        y=np.vstack((y_p,y_n))

        clf = SVC(kernel="linear", C=r)
        clf.fit(X,y)

        # get the separating hyperplane
        w = clf.coef_[0]
        a = -w[0] / w[1]
        # xx = np.linspace(-5,5)
        xx = np.linspace(-0.15, 0.15)
        yy = a * xx - (clf.intercept_[0]) / w[1]

        # plot the parallels to the separating hyperplane that pass through the
        # support vectors
        b = clf.support_vectors_[0]
        yy_down = a * xx + (b[1] - a * b[0])
        b = clf.support_vectors_[-1]
        yy_up = a * xx + (b[1] - a * b[0])

        # plot the line, the points, and the nearest vectors to the plane
        axs[j, p].plot(xx, yy, 'k--')
        axs[j, p].plot(xx, yy_down, 'k--')
        axs[j, p].plot(xx, yy_up, 'k--')

        axs[j, p].scatter(clf.support_vectors_[:, 0], clf.support_vectors_[:,␣
 ↪1],
                    s=80, facecolors='none')
        axs[j, p].scatter(X[:, 0], X[:, 1], c=y, cmap=plt.cm.Paired)
```

```
        axs[j, p].axis('tight')
        axs[j, p].set_title(f"Regularization Value: {r}, Imbalance: {i}")

plt.show()
```

```
<Figure size 1440x360 with 0 Axes>
```



2.2 Observations:

When the value of c is 0.001, the hyperplane could not differentiate the points, irrespective of the imbalance.

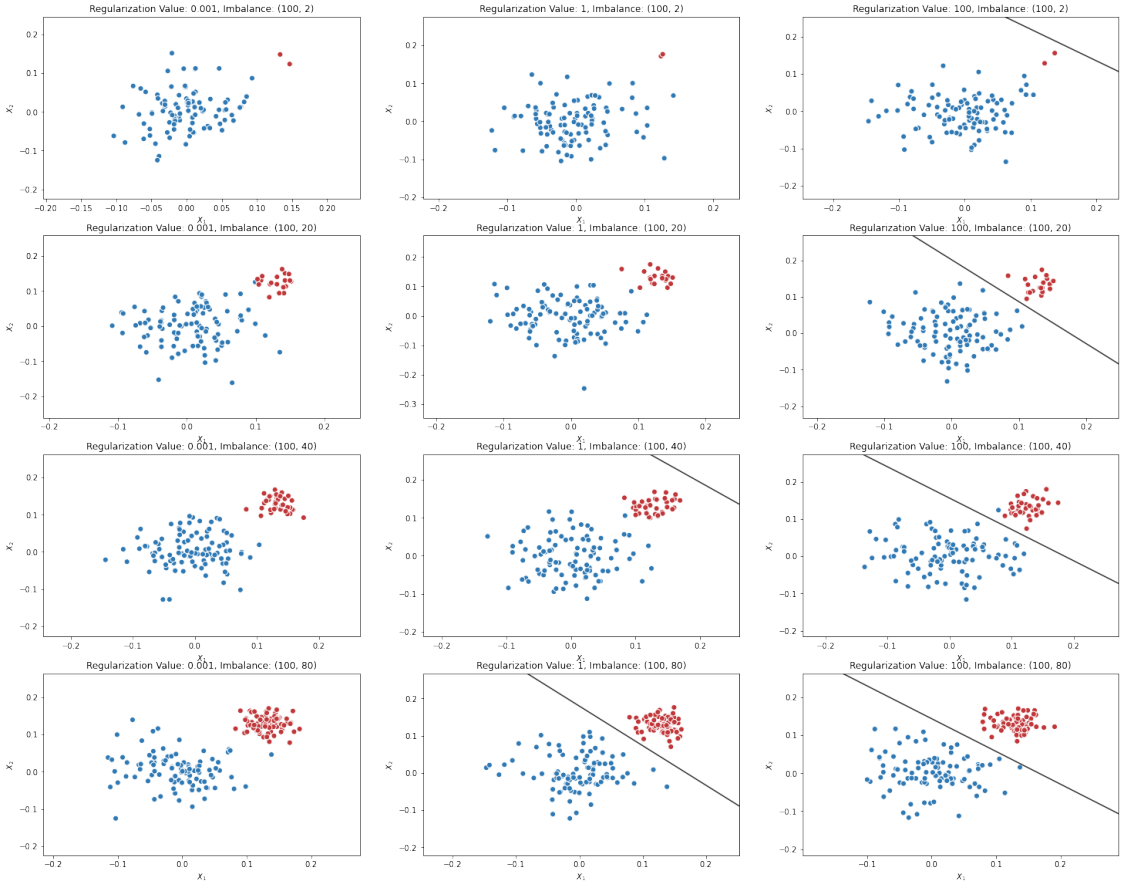
When the value of c is 1, the hyperplane could not differentiate when the imbalance is (100,2). It was able to do a slightly better job in the other imbalances.

When the value of c is 100, the hyperplane was able to differentiate in all imbalances.

2.3 Applying on Logistic Regression

[5]:
```
# ref: https://stackoverflow.com/questions/28256058/
 ↪plotting-decision-boundary-of-logistic-regression

reg_strengths = [0.001,1,100]
ratios = [(100,2), (100, 20), (100, 40), (100, 80)]
plt.figure(figsize=(20,5))
fig, axs = plt.subplots(4, 3, figsize=(25, 20))
# inc = 1
for p,r in enumerate(reg_strengths):
    for j,i in enumerate(ratios):
        X_p=np.random.normal(0,0.05,size=(i[0],2))
        X_n=np.random.normal(0.13,0.02,size=(i[1],2))
        y_p=np.array([1]*i[0]).reshape(-1,1)
        y_n=np.array([0]*i[1]).reshape(-1,1)
        X=np.vstack((X_p,X_n))
        y=np.vstack((y_p,y_n))

        clf = LogisticRegression(C=r)
        clf.fit(X, y)

        x_min, x_max = X[:, 0].min() - 0.1, X[:, 0].max() + 0.1
        y_min, y_max = X[:, 1].min() - 0.1, X[:, 1].max() + 0.1

        xx, yy = np.mgrid[x_min:x_max:.01, y_min:y_max:.01]
        grid = np.c_[xx.ravel(), yy.ravel()]

        probs = clf.predict_proba(grid)[:, 1].reshape(xx.shape)

        axs[j, p].contour(xx, yy, probs, levels=[.5], cmap="Greys", vmin=0,␣
 ↪vmax=.6)

        axs[j, p].scatter(X[:,0], X[:, 1], c=y[:], s=50,
                cmap="RdBu", vmin=-.2, vmax=1.2,
                edgecolor="white", linewidth=1)

        axs[j, p].set(aspect="equal",
            xlim=(x_min, x_max), ylim=(y_min, y_max),
            xlabel="$X_1$", ylabel="$X_2$")

        axs[j, p].axis('tight')
        axs[j, p].set_title(f"Regularization Value: {r}, Imbalance: {i}")
```

```
<Figure size 1440x360 with 0 Axes>
```



2.4 Observations:

When the value of c is 0.01, the model is not able to seperate irrespective of the imbalance value.

When the value of c is 1, the model is only able to seperate when the imbalance is 100:80 i.e., almost balanced dataset.

When the value of c is 100, the model is able to seperate in almost all of the cases except highly imbalanced dataset 100:2.

3 Task B

3.1 Features with varying variance and standardizations effect's on SVM and Logistic Regresssion

3.2 Features with varying variance and standardizations effects on Logistic Regresssion

[6]:
```
data = pd.read_csv('/WData/cS/AAIC/Assignments/Behaviour of Linear Models/
↪task_b.csv')
data=data.iloc[:,1:]
```

[7]:
```
data.head()
```

[7]:

	f1	f2	f3	y
0	-195.871045	-14843.084171	5.532140	1.0
1	-1217.183964	-4068.124621	4.416082	1.0
2	9.138451	4413.412028	0.425317	0.0
3	363.824242	15474.760647	1.094119	0.0
4	-768.812047	-7963.932192	1.870536	0.0

[8]:
```
data.corr()['y']
```

[8]:
```
f1    0.067172
f2   -0.017944
f3    0.839060
y     1.000000
Name: y, dtype: float64
```

[9]:
```
data.std()
```

[9]:
```
f1      488.195035
f2    10403.417325
f3        2.926662
y         0.501255
dtype: float64
```

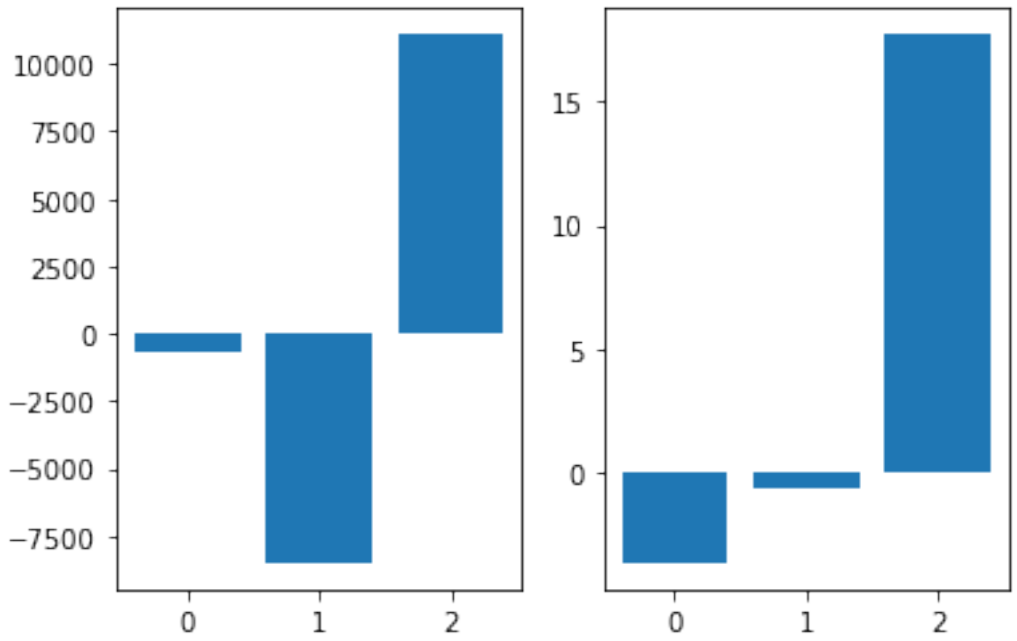
[10]:
```
X=data[['f1','f2','f3']].values
Y=data['y'].values
print(X.shape)
print(Y.shape)
```

```
(200, 3)
(200,)
```

[11]:
```
X_std, Y_std = np.zeros_like(X), np.zeros_like(Y)
for i in range(len(X[0])):
    X_std[:, i] = (X[:, i]-X[:,i].mean())/X[:,i].std()
Y_std = (Y-Y.mean())/Y.std()
```

[12]:
```
# Reference: https://machinelearningmastery.com/
 ↪calculate-feature-importance-with-python/
print("\nWithout Standardization: Logistic Regression\n")
plt.figure(figsize=(5,10))
fig, (ax1, ax2) = plt.subplots(1,2)
clf = SGDClassifier(loss="log")
clf.fit(X,Y)
importance = clf.coef_[0]
for i,v in enumerate(importance):
        print('Feature %0d Score: %.5f' % (i,v), end="\t")
ax1.bar([x for x in range(len(importance))], importance)

print("\n\nWith Standardization: Logistic Regression\n")

clf = SGDClassifier(loss="log")
clf.fit(X_std,Y_std)
importance = clf.coef_[0]
for i,v in enumerate(importance):
        print('Feature %0d Score: %.5f' % (i,v), end="\t")
# plot feature importance
ax2.bar([x for x in range(len(importance))], importance)
plt.show()
```

```
Without Standardization: Logistic Regression

Feature 0 Score: -711.29717     Feature 1 Score: -8504.68035    Feature 2 Score:
11094.26207

With Standardization: Logistic Regression

Feature 0 Score: -3.60903       Feature 1 Score: -0.64508       Feature 2 Score:
17.68562
<Figure size 360x720 with 0 Axes>
```



3.3 Observations:

When the data is not standardized, the top two features with highest variance, F2 and F3 have almost the same feature importance.

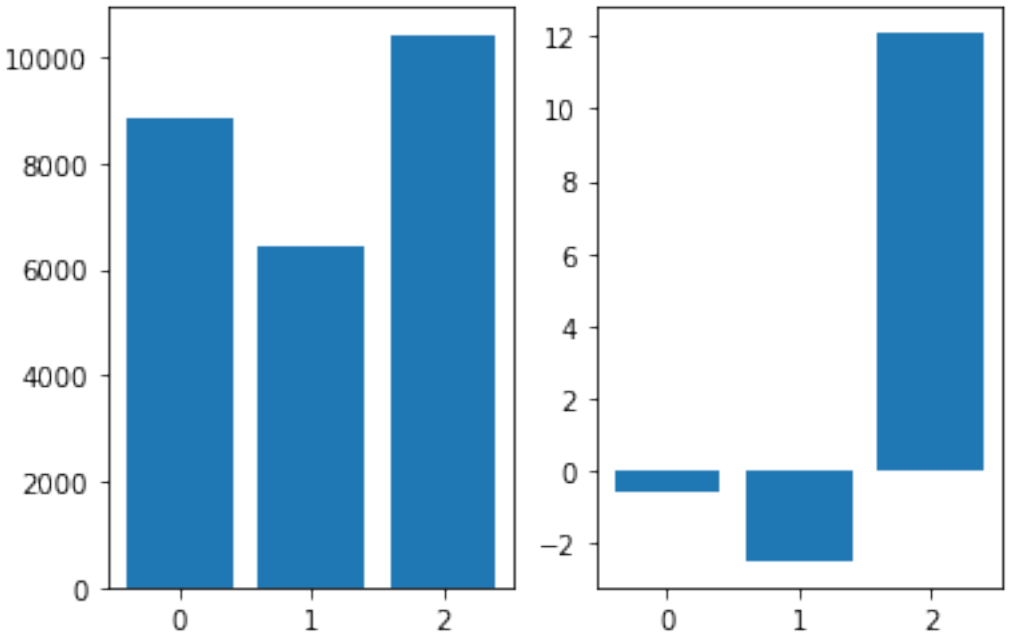
But, when the data is standardized. F3 has a relatively higher feature importance while the remaining two have low feature importance.

3.4 Features with varying variance and standardizations effects on SVM

[13]:
```
print("\nWithout Standardization: SVM\n")
fig, (ax1, ax2) = plt.subplots(1,2)
plt.figure(figsize=(5,10))
clf = SGDClassifier(loss="hinge")
clf.fit(X,Y)
importance = clf.coef_[0]
for i,v in enumerate(importance):
        print('Feature %0d Score: %.5f' % (i,v), end="\t")
ax1.bar([x for x in range(len(importance))], importance)

print("\n\nWith Standardization: SVM\n")

clf = SGDClassifier(loss="hinge")
clf.fit(X_std,Y_std)
importance = clf.coef_[0]
for i,v in enumerate(importance):
```

```
        print('Feature %0d Score: %.5f' % (i,v), end="\t")
ax2.bar([x for x in range(len(importance))], importance)
plt.show()
```

```
Without Standardization: SVM

Feature 0 Score: 8833.62790     Feature 1 Score: 6414.59610     Feature 2 Score:
10416.56461

With Standardization: SVM

Feature 0 Score: -0.56847       Feature 1 Score: -2.49318       Feature 2 Score:
12.08089
```




```
<Figure size 360x720 with 0 Axes>
```

3.5 Observations:

When the data is not standardized, feature 2 has the highest feature importance while feature 1 and 3 have almost similar like in logistic regression.

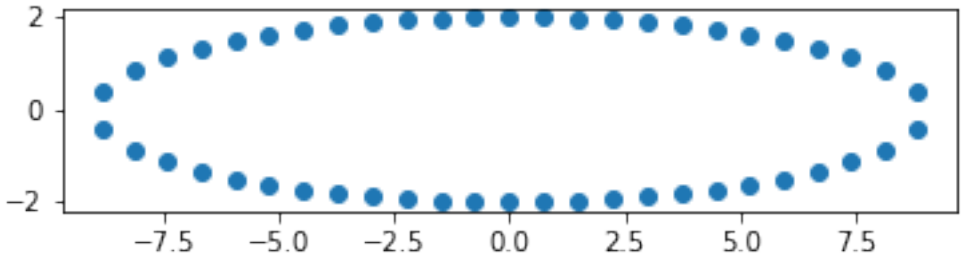
When the data is standardized, jsut like logistic regression, feature 3 has relatively higher feature importance than feature 1 and 2.

4 Task 3

4.1 Regression Outlier Effect: Visualization best fit Linear Regression for different scenarios

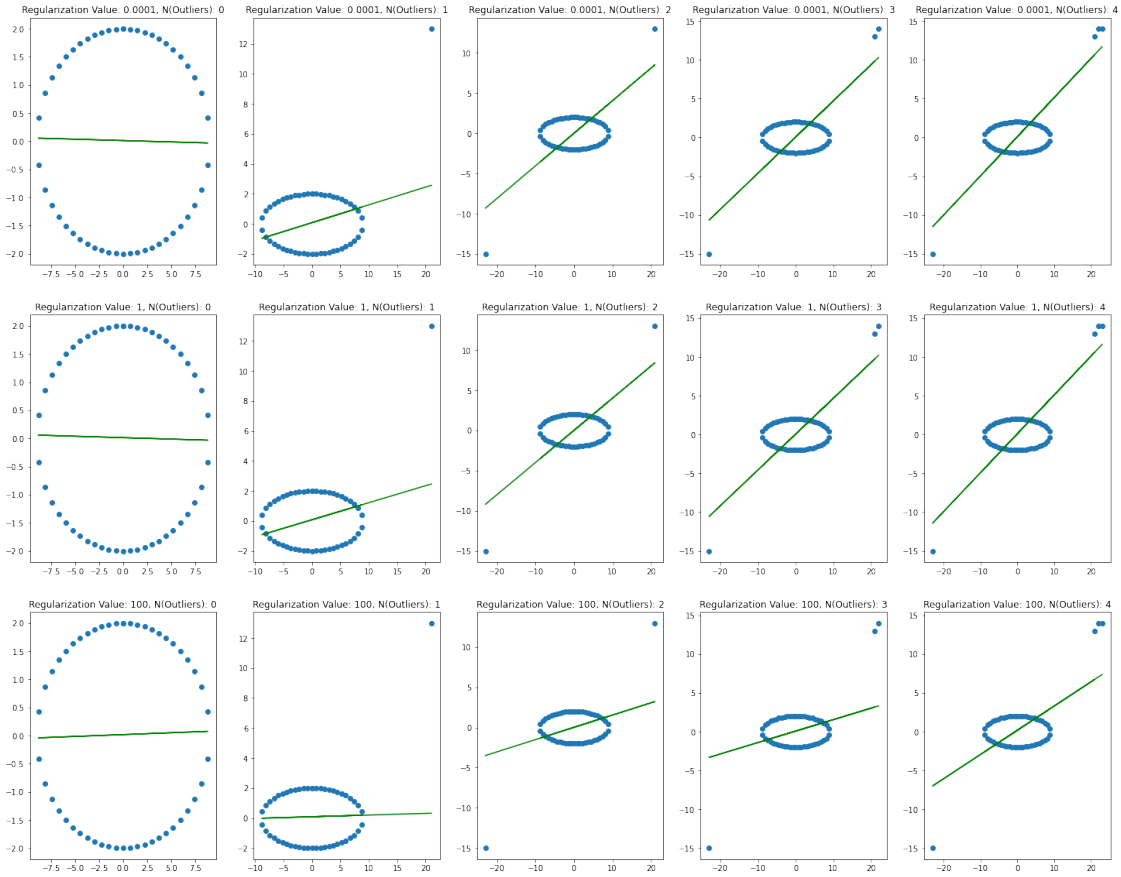
[14]:
```
def angles_in_ellipse(num,a,b):
    assert(num > 0)
    assert(a < b)
    angles = 2 * np.pi * np.arange(num) / num
    if a != b:
        e = (1.0 - a ** 2.0 / b ** 2.0) ** 0.5
        tot_size = sp.special.ellipeinc(2.0 * np.pi, e)
        arc_size = tot_size / num
        arcs = np.arange(num) * arc_size
        res = sp.optimize.root(
            lambda x: (sp.special.ellipeinc(x, e) - arcs), angles)
        angles = res.x
    return angles
```

[15]:
```
a = 2
b = 9
n = 50

phi = angles_in_ellipse(n, a, b)
e = (1.0 - a ** 2.0 / b ** 2.0) ** 0.5
arcs = sp.special.ellipeinc(phi, e)

fig = plt.figure()
ax = fig.gca()
ax.axes.set_aspect('equal')
ax.scatter(b * np.sin(phi), a * np.cos(phi))
plt.show()
```



[16]:
```
X= b * np.sin(phi)
Y= a * np.cos(phi)
```

[17]:
```
regs = [0.0001, 1, 100]
outliers = [(0,2),(21, 13), (-23, -15), (22,14), (23, 14)]

x_cp, y_cp = np.copy(X), np.copy(Y)
fig, axs = plt.subplots(3, 5, figsize=(25,20))

for i, r in enumerate(regs):
    for j, o in enumerate(outliers):
        x_cp = np.append(x_cp, o[0])
        y_cp = np.append(y_cp, o[1])
        x_cp = x_cp.reshape(-1,1)
        y_cp = y_cp.reshape(-1,1)

        clf = SGDRegressor(alpha=r, eta0=0.001,␣
 ↪learning_rate='constant',random_state=0)
        clf.fit(x_cp, y_cp)

        y_p = clf.predict(x_cp)

        axs[i, j].scatter(x_cp, y_cp)
        axs[i, j].plot(x_cp,y_p, color ='green')
        axs[i, j].axis('tight')
        axs[i, j].set_title(f"Regularization Value: {r}, N(Outliers): {j}")
    x_cp, y_cp = np.copy(X), np.copy(Y)
```



4.2 Observations:

As the regularization value increases, the tendency to shift towards the outliers decreses. This can be seen in the case where the Regularization Value is 100 and the number of outliers is 4. The decision boundary shifts very little towards the outliers but in the case of same number of outliers with less regularization (0.0001 and 1) the decision boundary shifts towards the outliers

5 Task D

5.1 Collinear Features and their effect on Linear Models

[18]:
```
data = pd.read_csv('/WData/cS/AAIC/Assignments/Behaviour of Linear Models/
↪task_d.csv')
```

[19]:
```
data.head()
```

[19]:

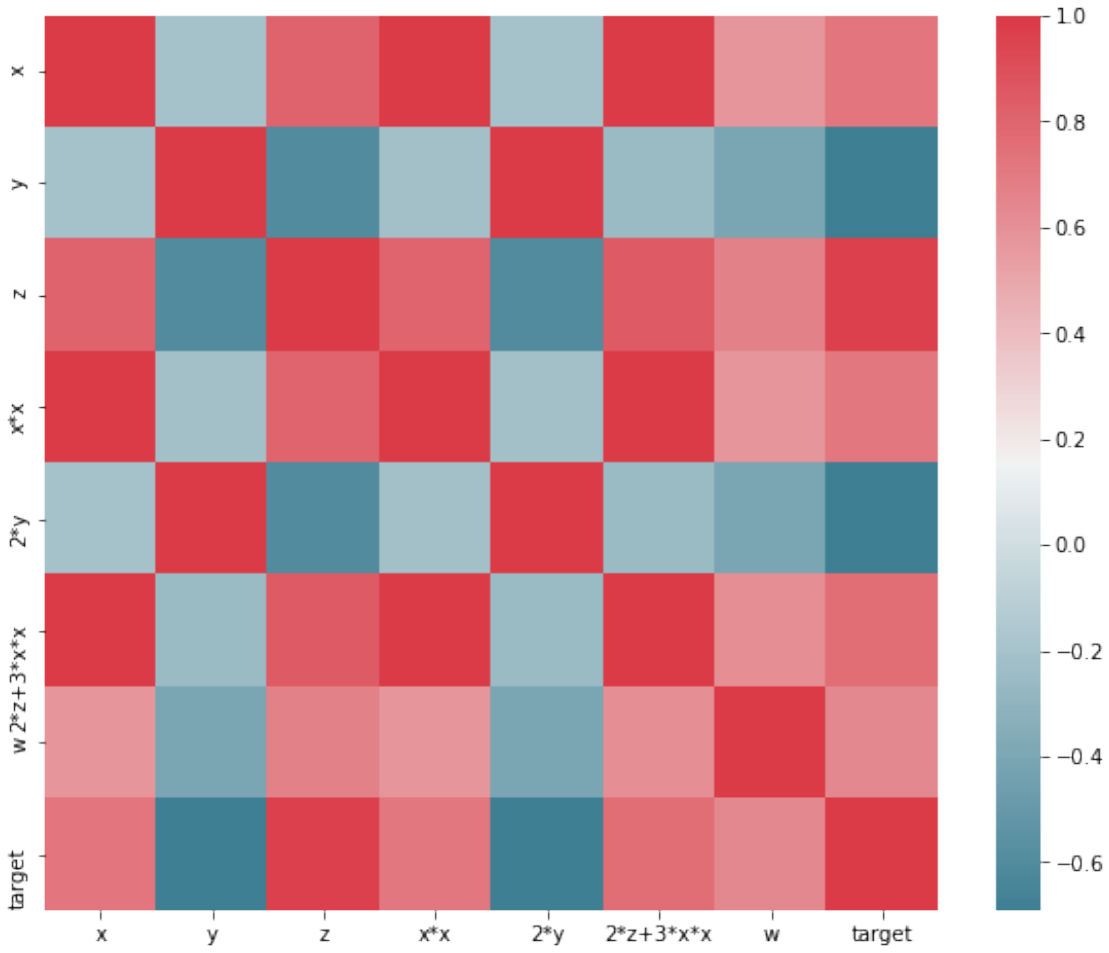
	x	y	z	x*x	2*y	2*z+3*x*x	w	\
0	-0.581066	0.841837	-1.012978	-0.604025	0.841837	-0.665927	-0.536277	
1	-0.894309	-0.207835	-1.012978	-0.883052	-0.207835	-0.917054	-0.522364	
2	-1.207552	0.212034	-1.082312	-1.150918	0.212034	-1.166507	0.205738	

```
3 -1.364174  0.002099 -0.943643 -1.280666  0.002099  -1.266540 -0.665720
4 -0.737687  1.051772 -1.012978 -0.744934  1.051772  -0.792746 -0.735054

   target
0       0
1       0
2       0
3       0
4       0
```

[20]:
```
X = data.drop(['target'], axis=1).values
Y = data['target'].values
```

[21]:
```
# https://seaborn.pydata.org/examples/many_pairwise_correlations.html
f, ax = plt.subplots(figsize=(10, 8))
corr = data.corr()
sns.heatmap(corr, mask=np.zeros_like(corr, dtype=np.bool), cmap=sns.
 ↪diverging_palette(220, 10, as_cmap=True),
            square=True, ax=ax)
```

[21]:
```
<AxesSubplot:>
```



[22]:
```
corr
```

[22]:
```
                   x         y         z       x*x       2*y  2*z+3*x*x  \
x           1.000000 -0.205926  0.812458  0.997947 -0.205926   0.996252
y          -0.205926  1.000000 -0.602663 -0.209289  1.000000  -0.261123
z           0.812458 -0.602663  1.000000  0.807137 -0.602663   0.847163
x*x         0.997947 -0.209289  0.807137  1.000000 -0.209289   0.997457
2*y        -0.205926  1.000000 -0.602663 -0.209289  1.000000  -0.261123
2*z+3*x*x   0.996252 -0.261123  0.847163  0.997457 -0.261123   1.000000
w           0.583277 -0.401790  0.674486  0.583803 -0.401790   0.606860
target      0.728290 -0.690684  0.969990  0.719570 -0.690684   0.764729

                   w    target
x           0.583277  0.728290
y          -0.401790 -0.690684
z           0.674486  0.969990
x*x         0.583803  0.719570
```

```
2*y        -0.401790 -0.690684
2*z+3*x*x   0.606860  0.764729
w           1.000000  0.641750
target      0.641750  1.000000
```

5.2 Observations

Top 2 Features with Highest Correlation with X : X*X*, *2(Z)+3(X*X)

Top 2 Features with Highest Correlation with Y : 2*Y, Z

Top 2 Features with Highest Correlation with Z : 2(Z)+3(X*X), X

Top 2 Features with Highest Correlation with X*X* : *X, 2(Z)+3(X*X)

Top 2 Features with Highest Correlation with 2*Y : Y, X

Top 2 Features with Highest Correlation with 2*Z+3*(X*X*): *X, X*X

Top 2 Features with Highest Correlation with W : Z, 2(Z)+3(X*X)

5.3 Effect on Logistic Regression

[23]:
```
grid={"alpha":np.logspace(-3,3), "penalty":["l1","l2"]}

lr = SGDClassifier(loss='log')
lr_cv = GridSearchCV(lr, grid, cv=10)
lr_cv.fit(X,Y)

print("Best Parameters: ",lr_cv.best_params_)
```

```
Best Parameters:  {'alpha': 0.001, 'penalty': 'l1'}
```

[24]:
```
best_model = SGDClassifier(loss='log', alpha=lr_cv.best_params_['alpha'])
best_model.fit(X,Y)
```

[24]:
```
SGDClassifier(alpha=0.001, loss='log')
```

[25]:
```
best_model_accuracy = best_model.score(X,Y)
best_model_accuracy
```

[25]:
```
1.0
```

[26]:
```
best_model_coefs = best_model.coef_
best_model_coefs
```

[26]:
```
array([[ 1.68750159, -1.59084878,  3.20594999,  1.53960162, -1.59084878,
         1.77271701,  0.61343182]])
```

[27]:
```
X_mod = np.copy(X)
X_mod += 0.01
```

[28]:
```
best_model.fit(X_mod, Y)
```

[28]:
```
SGDClassifier(alpha=0.001, loss='log')
```

[29]:
```
best_model_edited_accuracy = best_model.score(X_mod, Y)
best_model_edited_accuracy
```

[29]:
```
1.0
```

[30]:
```
best_model_edited_coefs = best_model.coef_
best_model_edited_coefs
```

[30]:
```
array([[ 1.22744645, -2.54721889,  4.43462828,  0.98118446, -2.54721889,
         1.41839609,  1.34843579]])
```

[31]:
```
pers = []
for i in range(len(X[0])):
    p = ((best_model_edited_coefs[0][i] - best_model_coefs[0][i])/
 ↪best_model_coefs[0][i])*100
    pers.append(abs(p))
    print(f"Feature {i+1}: W on Original Data {best_model_coefs[0][i]}, W on␣
 ↪Noisy Data {best_model_edited_coefs[0][i]}, Percentage Change {p}%\n")
```

```
Feature 1: W on Original Data 1.6875015854081166, W on Noisy Data
1.227446454617004, Percentage Change -27.262500655953442%

Feature 2: W on Original Data -1.590848781169827, W on Noisy Data
-2.547218890363502, Percentage Change 60.11697155089814%

Feature 3: W on Original Data 3.2059499942661334, W on Noisy Data
4.43462827978583, Percentage Change 38.324936063169964%

Feature 4: W on Original Data 1.5396016213724006, W on Noisy Data
0.981184456182716, Percentage Change -36.270237536637026%

Feature 5: W on Original Data -1.590848781169827, W on Noisy Data
-2.547218890363502, Percentage Change 60.11697155089814%

Feature 6: W on Original Data 1.7727170085300368, W on Noisy Data
1.418396091163132, Percentage Change -19.98744952871598%

Feature 7: W on Original Data 0.6134318188562703, W on Noisy Data
1.348435787783469, Percentage Change 119.81836388885026%
```

[32]:
```
top4 = sorted(range(len(pers)), key=lambda i: pers[i])[-4:]
print(f"The features with most changes are {[data.columns[i] for i in top4]}")
```

```
The features with most changes are ['z', 'y', '2*y', 'w']
```

5.4 Effect on SVM

[33]:
```
grid={"alpha":np.logspace(-3,3), "penalty":["l1","l2"]}

svm = SGDClassifier(loss='hinge')
svm_cv = GridSearchCV(svm, grid, cv=10)
svm_cv.fit(X,Y)

print("Best Parameters: ",svm_cv.best_params_)
```

```
Best Parameters:  {'alpha': 0.001, 'penalty': 'l1'}
```

[34]:
```
best_model_svm = SGDClassifier(loss='hinge', alpha=svm_cv.best_params_['alpha'])
best_model_svm.fit(X,Y)
```

[34]:
```
SGDClassifier(alpha=0.001)
```

[35]:
```
best_model_accuracy_svm = best_model_svm.score(X,Y)
best_model_accuracy_svm
```

[35]:
```
1.0
```

[36]:
```
best_model_coefs_svm = best_model_svm.coef_
best_model_coefs_svm
```

[36]:
```
array([[ 2.08437935, -3.00880243,  4.61365302,  1.82099623, -3.00880243,
         2.1959298 ,  2.23136189]])
```

[37]:
```
X_mod = np.copy(X)
X_mod += 0.01
```

[38]:
```
best_model_svm.fit(X_mod, Y)
```

[38]:
```
SGDClassifier(alpha=0.001)
```

[39]:
```
best_model_edited_accuracy_svm = best_model_svm.score(X_mod, Y)
best_model_edited_accuracy_svm
```

[39]:
```
1.0
```

[40]:
```
best_model_edited_coefs_svm = best_model_svm.coef_
best_model_edited_coefs_svm
```

[40]:
```
array([[ 1.2503602 , -1.43654945,  2.76759992,  1.07400506, -1.43654945,
         1.30075284,  1.85724315]])
```

[41]:
```
pers_svm = []
for i in range(len(X[0])):
```

```
    p = ((best_model_edited_coefs_svm[0][i] - best_model_coefs_svm[0][i])/
↪best_model_coefs_svm[0][i])*100
    pers_svm.append(abs(p))
    print(f"Feature {i+1}: W on Original Data {best_model_coefs_svm[0][i]}, W␣
↪on Noisy Data {best_model_edited_coefs_svm[0][i]}, Percentage Change {p}%")
```

```
Feature 1: W on Original Data 2.0843793469396603, W on Noisy Data
1.2503602021375058, Percentage Change -40.01282904796975%
Feature 2: W on Original Data -3.0088024299908596, W on Noisy Data
-1.4365494455961407, Percentage Change -52.25510883409833%
Feature 3: W on Original Data 4.613653018386573, W on Noisy Data
2.7675999232730617, Percentage Change -40.01282904796965%
Feature 4: W on Original Data 1.8209962336243142, W on Noisy Data
1.0740050627918254, Percentage Change -41.02101679506268%
Feature 5: W on Original Data -3.0088024299908596, W on Noisy Data
-1.4365494455961407, Percentage Change -52.25510883409833%
Feature 6: W on Original Data 2.1959298048195888, W on Noisy Data
1.3007528412733524, Percentage Change -40.765281366531724%
Feature 7: W on Original Data 2.2313618866402223, W on Noisy Data
1.857243149416723, Percentage Change -16.766385563159933%
```

[42]:
```
top4 = sorted(range(len(pers_svm)), key=lambda i: pers[i])[-4:]
print(f"The features with most changes are {[data.columns[i] for i in top4]}")
```

```
The features with most changes are ['z', 'y', '2*y', 'w']
```

5.5 Observations

Both Logistic Regression and SVM have the same top 4 features with highest change.

Three of the top 4 features in both cases have very high correlation (X, 2(Z)+3(XX), XX)

6 Task E

6.1 Implementing Decision Function of SVM RBF Kernel

[43]:
```
X, y = make_classification(n_samples=5000, n_features=5, n_redundant=2,
                           n_classes=2, weights=[0.7], class_sep=0.7,␣
↪random_state=15)
```

[44]:
```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.2,␣
↪stratify=y)
X_train, X_cv, y_train, y_cv = train_test_split(X_train, y_train, test_size = 0.
↪2, stratify=y_train)
```

[45]:
```
svm = SVC(gamma=0.001, C=100.)
svm.fit(X_train, y_train)
```

[45]:
```
SVC(C=100.0, gamma=0.001)
```

[46]:
```
# support vectors
sup_vecs = svm.support_vectors_
# intercept
inte = svm.intercept_
# gamma
ga = 0.001
# alpha values
al = svm.dual_coef_[0]
```

[47]:
```
def rbf(ga, xi, xq):
    dist = sum((xi-xq)**2)
    # dist = np.sqrt(np.dot(xi, xi) - 2 * np.dot(xi, xq) + np.dot(xq, xq))
    # dist = np.linalg.norm(xi-xq)
    return np.exp(-ga*dist)
```

[48]:
```
def decisionfunction(sup_vecs, X_cv, ga, inte, al):
    decisions = np.zeros(X_cv.shape[0])
    temp = 0
    for n, xq in enumerate(X_cv):
        for i in range(len(sup_vecs)):
            temp += (al[i] * rbf(ga, sup_vecs[i], xq))
        decisions[n] = temp+inte
        temp = 0
    return decisions
```

[49]:
```
decisions = decisionfunction(sup_vecs, X_cv, ga, inte, al)
svm_decs = svm.decision_function(X_cv)
cos_sim = np.dot(decisions, svm_decs)/(np.linalg.norm(decisions)*np.linalg.
↪norm(svm_decs))
cos_sim
```

[49]:
```
0.9999999999999998
```

[50]:
```
np.all(np.round(decisions, 7) == np.round(svm_decs, 7))
```

[50]:
```
True
```

7 Task F

7.1 Implementing the Decision Function of Logistic Regression and getting probabilities

[51]:
```
unique, counts = np.unique(y_train, return_counts=True)
n_0, n_1 = dict(zip(unique, counts))[0], dict(zip(unique, counts))[1]
```

[52]:
```
y_p = (n_1 + 1)/(n_1 + 2)
y_m = 1/(n_0 + 2)
```

[53]:
```
y_p, y_m
```

[53]:
```
(0.9989701338825953, 0.0004478280340349306)
```

[54]:
```
def initialize_weights(dim):
    ''' In this function, we will initialize our weights and bias'''
    #initialize the weights to zeros array of (1,dim) dimensions
    #you use zeros_like function to initialize zero, check this link https://
 ↪docs.scipy.org/doc/numpy/reference/generated/numpy.zeros_like.html
    #initialize bias to zero
    b = 0
    w = np.zeros_like(dim)
    return w,b
```

[55]:
```
def sigmoid(z):
    ''' In this function, we will return sigmoid of z'''
    # compute sigmoid(z) and return
    return (1/(1+np.exp(-z)))
```

[56]:
```
def logloss(y_true,y_pred, y_p, y_m):
    '''In this function, we will compute log loss '''
    loss = 0
    n = len(y_true)
    for i in range(n):
        if(y_true[i] == 0):
            loss += ( ( np.dot(y_m, np.log10(y_pred[i])) ) + ( np.
 ↪dot((1-y_m),np.log10(1-y_pred[i])) ) )
        else:
            loss += ( ( np.dot(y_p, np.log10(y_pred[i])) ) + ( np.
 ↪dot((1-y_p),np.log10(1-y_pred[i])) ) )
    loss = -1/n * loss
    return loss
```

[57]:
```
def gradient_dw(x,y,w,b,alpha,N):
    '''In this function, we will compute the gardient w.r.to w '''
    dw = (np.dot(x,(y - sigmoid(np.dot(w,x) + b))) - (alpha/N)*w)
    return dw
```

[58]:
```
 def gradient_db(x,y,w,b):
    '''In this function, we will compute gradient w.r.to b '''
    db = y - sigmoid(np.dot(w,x) + b)
    return db
```

[59]:
```
def train(X_train,y_train,epochs,alpha,eta0, y_p, y_m):
    ''' In this function, we will implement logistic regression'''
    #Here eta0 is learning rate
    #implement the code as follows
```

```
    # initalize the weights (call the initialize_weights(X_train[0]) function)
    # for every epoch
        # for every data point(X_train,y_train)
           #compute gradient w.r.to w (call the gradient_dw() function)
           #compute gradient w.r.to b (call the gradient_db() function)
           #update w, b
        # predict the output of x_train[for all data points in X_train] using␣
↪w,b
        #compute the loss between predicted and actual values (call the loss␣
↪function)
        # store all the train loss values in a list
        #compute the loss between predicted and actual values (call the loss␣
↪function)
        # you can also compare previous loss and current loss, if loss is not␣
↪updating then stop the process and return w,b
    N = len(X_train)
    w, b = initialize_weights(X_train[0])

    # x_train_loss will store the loss values at the end of each epoch
    x_train_loss = []

    # prev_train_loss will keep track of the loss values in the previous epoch
    prev_train_loss = 0

    # epochs_ran will be incremented each time an epoch is finished
    epochs_ran = 0

    # iterating over the number of epochs
    for i in range(epochs):
        for j in range(len(X_train)):
            # calculating the dw and db values and updating w and b values
            dw = gradient_dw(X_train[j], y_train[j], w, b, alpha, N)
            db = gradient_db(X_train[j], y_train[j], w, b)
            w = w + (eta0 * dw)
            b = b + (eta0 * db)

        # x_train_pred will store the probabilities obtained using this epoch's␣
↪w and b values
        x_train_pred = []

        # calculating and storing x_train_pred
    # return w,b,x_train_loss,x_test_loss, epochs_ran
        for k in range(len(X_train)):
            x_train_pred.append(sigmoid(np.dot(w, X_train[k])+b))

        # calculating the logloss values of train probabilities obtained
        current_train_loss = logloss(y_train, x_train_pred, y_p, y_m)
```

```
        # storing the logloss values obtained
        x_train_loss.append(current_train_loss)

        # printing the informationg of this epoch
        print(f"Epoch: {i}\nTrain Loss: {current_train_loss}\nPrev Train Loss:␣
↪{prev_train_loss}", "="*10, "\n")

        # if this is the first epoch, storing the current loss values in the␣
↪prev_train_loss
        if(i == 0):
            prev_train_loss = current_train_loss
        # if this is not the first epoch,
        else:
            # checking if the difference of train loss between current epoch␣
↪and previous epoch is less than 10^-5
            if(prev_train_loss-current_train_loss < 0.00001):
                # if it is, breaking the execution
                print("No improvement in train loss")
                break
            else:
                # if it is not, then storing the current train as previous loss␣
↪values and moving on to the next epoch
                prev_train_loss = current_train_loss

        # incrementing the epochs_ran value
        epochs_ran += 1
    # returing the values
    return w,b,x_train_loss, epochs_ran
```

[60]:
```
y_cv_mod = np.copy(y_cv)
y_cv_mod[y_cv_mod == 1] = y_p
y_cv_mod[y_cv_mod == 0] = y_m
```

[61]:
```
alpha=0.001
eta0=0.0001
N=len(decisions)
epochs=50
w,b,x_train_loss,epochs_ran =␣
↪train(decisions,y_cv_mod,epochs,alpha,eta0,y_p,y_m)
```

```
Epoch: 0
Train Loss: 0.2816742609374453
Prev Train Loss: 0 ==========

Epoch: 1
Train Loss: 0.2655737239076833
```

```
Prev Train Loss: 0.2816742609374453 ==========

Epoch: 2
Train Loss: 0.2520408337300121
Prev Train Loss: 0.2655737239076833 ==========

Epoch: 3
Train Loss: 0.2405301005574132
Prev Train Loss: 0.2520408337300121 ==========

Epoch: 4
Train Loss: 0.2306167379781315
Prev Train Loss: 0.2405301005574132 ==========

Epoch: 5
Train Loss: 0.2219728606002217
Prev Train Loss: 0.2306167379781315 ==========

Epoch: 6
Train Loss: 0.2143459661959857
Prev Train Loss: 0.2219728606002217 ==========

Epoch: 7
Train Loss: 0.20754123738156896
Prev Train Loss: 0.2143459661959857 ==========

Epoch: 8
Train Loss: 0.20140767955476496
Prev Train Loss: 0.20754123738156896 ==========

Epoch: 9
Train Loss: 0.19582754102198152
Prev Train Loss: 0.20140767955476496 ==========

Epoch: 10
Train Loss: 0.19070834158536026
Prev Train Loss: 0.19582754102198152 ==========

Epoch: 11
Train Loss: 0.1859768970576504
Prev Train Loss: 0.19070834158536026 ==========

Epoch: 12
Train Loss: 0.1815748413119741
Prev Train Loss: 0.1859768970576504 ==========

Epoch: 13
Train Loss: 0.1774552615288387
```

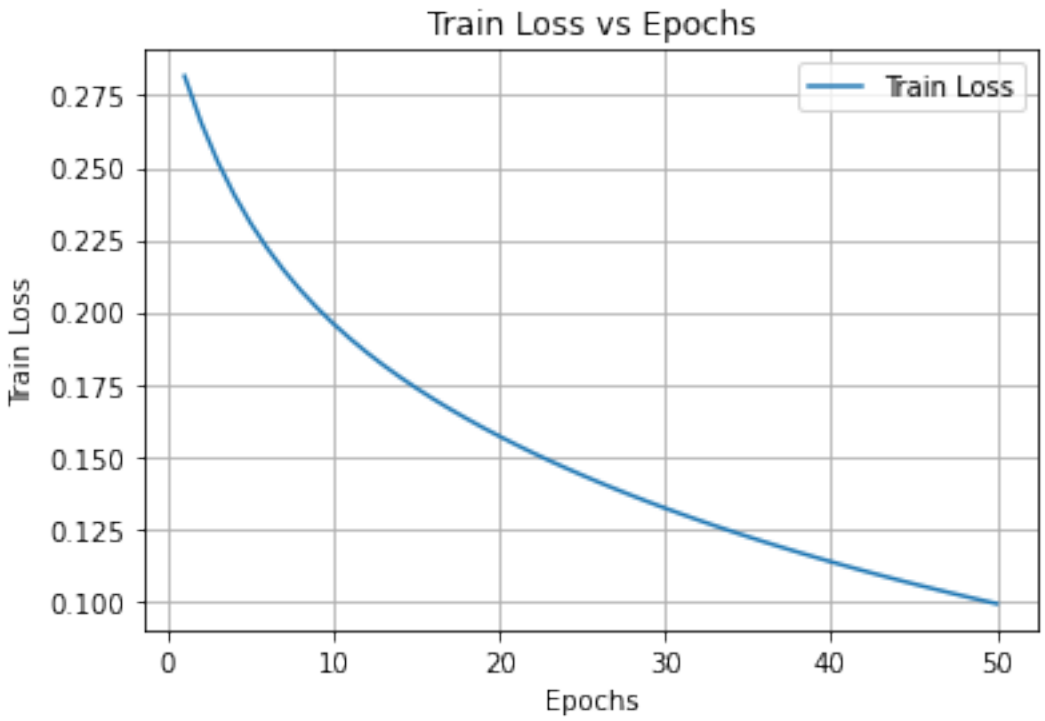
```
Prev Train Loss: 0.1815748413119741 ==========

Epoch: 14
Train Loss: 0.1735801583943823
Prev Train Loss: 0.1774552615288387 ==========

Epoch: 15
Train Loss: 0.169918518111424
Prev Train Loss: 0.1735801583943823 ==========

Epoch: 16
Train Loss: 0.16644483960226297
Prev Train Loss: 0.169918518111424 ==========

Epoch: 17
Train Loss: 0.16313800197386885
Prev Train Loss: 0.16644483960226297 ==========

Epoch: 18
Train Loss: 0.15998038777082127
Prev Train Loss: 0.16313800197386885 ==========

Epoch: 19
Train Loss: 0.15695719970301567
Prev Train Loss: 0.15998038777082127 ==========

Epoch: 20
Train Loss: 0.15405592466095303
Prev Train Loss: 0.15695719970301567 ==========

Epoch: 21
Train Loss: 0.1512659105925678
Prev Train Loss: 0.15405592466095303 ==========

Epoch: 22
Train Loss: 0.14857803042671808
Prev Train Loss: 0.1512659105925678 ==========

Epoch: 23
Train Loss: 0.14598441356405198
Prev Train Loss: 0.14857803042671808 ==========

Epoch: 24
Train Loss: 0.14347823014304162
Prev Train Loss: 0.14598441356405198 ==========

Epoch: 25
Train Loss: 0.14105351677685626
```

```
Prev Train Loss: 0.14347823014304162 ==========

Epoch: 26
Train Loss: 0.13870503506805398
Prev Train Loss: 0.14105351677685626 ==========

Epoch: 27
Train Loss: 0.13642815617516857
Prev Train Loss: 0.13870503506805398 ==========

Epoch: 28
Train Loss: 0.13421876619616727
Prev Train Loss: 0.13642815617516857 ==========

Epoch: 29
Train Loss: 0.1320731882706549
Prev Train Loss: 0.13421876619616727 ==========

Epoch: 30
Train Loss: 0.12998811817480532
Prev Train Loss: 0.1320731882706549 ==========

Epoch: 31
Train Loss: 0.1279605708559013
Prev Train Loss: 0.12998811817480532 ==========

Epoch: 32
Train Loss: 0.12598783587551363
Prev Train Loss: 0.1279605708559013 ==========

Epoch: 33
Train Loss: 0.12406744013774827
Prev Train Loss: 0.12598783587551363 ==========

Epoch: 34
Train Loss: 0.12219711659854904
Prev Train Loss: 0.12406744013774827 ==========

Epoch: 35
Train Loss: 0.12037477790399255
Prev Train Loss: 0.12219711659854904 ==========

Epoch: 36
Train Loss: 0.1185984941051389
Prev Train Loss: 0.12037477790399255 ==========

Epoch: 37
Train Loss: 0.11686647375593279
```

```
Prev Train Loss: 0.1185984941051389 ==========

Epoch: 38
Train Loss: 0.11517704782774084
Prev Train Loss: 0.11686647375593279 ==========

Epoch: 39
Train Loss: 0.11352865597621722
Prev Train Loss: 0.11517704782774084 ==========

Epoch: 40
Train Loss: 0.11191983477854017
Prev Train Loss: 0.11352865597621722 ==========

Epoch: 41
Train Loss: 0.11034920762576272
Prev Train Loss: 0.11191983477854017 ==========

Epoch: 42
Train Loss: 0.10881547600925343
Prev Train Loss: 0.11034920762576272 ==========

Epoch: 43
Train Loss: 0.10731741198444945
Prev Train Loss: 0.10881547600925343 ==========

Epoch: 44
Train Loss: 0.10585385163138111
Prev Train Loss: 0.10731741198444945 ==========

Epoch: 45
Train Loss: 0.10442368936120298
Prev Train Loss: 0.10585385163138111 ==========

Epoch: 46
Train Loss: 0.1030258729425001
Prev Train Loss: 0.10442368936120298 ==========

Epoch: 47
Train Loss: 0.10165939914142817
Prev Train Loss: 0.1030258729425001 ==========

Epoch: 48
Train Loss: 0.10032330988656174
Prev Train Loss: 0.10165939914142817 ==========

Epoch: 49
Train Loss: 0.09901668888331672
```

```
Prev Train Loss: 0.10032330988656174 ==========
```

[62]:
```
# Plotting epochs vs loss for train and test
import matplotlib.pyplot as plt
a = np.arange(1,epochs_ran+1)
fig, ax = plt.subplots()
ax.plot(a, x_train_loss, label="Train Loss")
ax.set(xlabel='Epochs', ylabel='Train Loss', title='Train Loss vs Epochs')
ax.grid()
ax.legend()
plt.show()
```



[63]:
```
test_decisions = decisionfunction(sup_vecs, X_test, ga, inte, al)
svm_decs_test = svm.decision_function(X_test)
np.all(np.round(test_decisions, 7) == np.round(svm_decs_test, 7))
```

[63]:
```
True
```

[64]:
```
z = np.dot(w, test_decisions)+b
probs = sigmoid(z)
```

[65]:
```
probs
```

```
[65]: array([0.12194153, 0.09959089, 0.06250471, 0.12679045, 0.17192108,
             0.41541018, 0.0556793 , 0.38745359, 0.18920315, 0.37387138,
             0.40159398, 0.11271902, 0.09047785, 0.21837703, 0.07349848,
             0.17032578, 0.30403917, 0.10102154, 0.10252634, 0.38719181,
             0.4013879 , 0.10633997, 0.07683491, 0.1303696 , 0.10824497,
             0.34564051, 0.0962674 , 0.23969843, 0.43405466, 0.14778028,
             0.09267949, 0.08436074, 0.35903594, 0.14277053, 0.20195204,
             0.20131095, 0.10827412, 0.3798482 , 0.1150874 , 0.11293276,
             0.05883223, 0.07793399, 0.08454302, 0.10128411, 0.26294772,
             0.12832373, 0.05706534, 0.11618551, 0.09978059, 0.16722061,
             0.06234677, 0.13322606, 0.12427388, 0.08709549, 0.05470527,
             0.11231077, 0.08873865, 0.08775788, 0.30724092, 0.09826117,
             0.11499293, 0.07450289, 0.11579345, 0.40956082, 0.43028559,
             0.05911207, 0.10211069, 0.16055927, 0.28401551, 0.36345866,
             0.08673326, 0.41189962, 0.10470003, 0.13916115, 0.33873207,
             0.10530881, 0.52628922, 0.09327183, 0.13600895, 0.21788501,
             0.10330836, 0.3052371 , 0.1917786 , 0.24522753, 0.07258168,
             0.34507477, 0.12183811, 0.16294008, 0.36645441, 0.11219473,
             0.2782298 , 0.44186935, 0.07891272, 0.11851151, 0.35960444,
             0.11594995, 0.13103586, 0.21584583, 0.33503073, 0.34717424,
             0.25534081, 0.1236157 , 0.3604829 , 0.07276537, 0.48182435,
             0.16792864, 0.1616127 , 0.20018512, 0.08875559, 0.36988983,
             0.14715059, 0.35577784, 0.08264811, 0.34748711, 0.07581827,
             0.12127986, 0.08312354, 0.07943279, 0.11003716, 0.10519145,
             0.17756878, 0.44410938, 0.2281117 , 0.04951743, 0.07532671,
             0.33198812, 0.41676692, 0.27844567, 0.06474697, 0.15144895,
             0.10982998, 0.04846589, 0.10750064, 0.26109325, 0.18575008,
             0.17537733, 0.09399215, 0.1582461 , 0.07292951, 0.3999882 ,
             0.1400405 , 0.29537805, 0.41532079, 0.07555078, 0.15420612,
             0.12853217, 0.10340744, 0.10657248, 0.42024813, 0.28479525,
             0.37191319, 0.36740814, 0.09519606, 0.34698026, 0.12758552,
             0.16898382, 0.14438986, 0.17609319, 0.19279132, 0.22809052,
             0.08487717, 0.1419971 , 0.09036468, 0.43401743, 0.11718949,
             0.14985038, 0.13289373, 0.15425902, 0.11034888, 0.45400442,
             0.30715742, 0.10306666, 0.06924641, 0.07766493, 0.14729713,
             0.44733892, 0.28027378, 0.16011769, 0.42704896, 0.0837242 ,
             0.08211444, 0.27456427, 0.16666868, 0.11670922, 0.13153355,
             0.11305113, 0.1776642 , 0.0695781 , 0.14149285, 0.1917812 ,
             0.10610649, 0.26778968, 0.54208332, 0.21983919, 0.18131963,
             0.36278305, 0.14110889, 0.42063835, 0.2524492 , 0.11490636,
             0.20701122, 0.35965326, 0.31935364, 0.11956778, 0.08975872,
             0.09909876, 0.05994151, 0.13658995, 0.10443455, 0.13254028,
             0.40645496, 0.10344614, 0.06640223, 0.08527676, 0.38387648,
             0.17393156, 0.41190409, 0.13986071, 0.07834475, 0.09884724,
             0.34980021, 0.28906705, 0.21857373, 0.14034234, 0.07089165,
             0.11428101, 0.22201268, 0.05988736, 0.43551991, 0.11872072,
             0.19256033, 0.0823615 , 0.13167652, 0.20283548, 0.17629123,
```

```
0.10111619, 0.06170588, 0.16797282, 0.09218123, 0.06425084,
0.11392941, 0.44732483, 0.07421953, 0.44945035, 0.3751348 ,
0.4087646 , 0.33858268, 0.461023  , 0.41351618, 0.23510359,
0.17928677, 0.15571725, 0.08606675, 0.09106221, 0.10551229,
0.33401251, 0.13078833, 0.15738338, 0.1066862 , 0.09916657,
0.1206282 , 0.11565266, 0.07115219, 0.14777354, 0.08624621,
0.05803766, 0.21784206, 0.27991628, 0.17659873, 0.07970326,
0.3659964 , 0.40571117, 0.05601125, 0.5433336 , 0.11679545,
0.07038473, 0.38365126, 0.21994662, 0.17260278, 0.10360547,
0.15604448, 0.40624369, 0.44192632, 0.08660222, 0.06436724,
0.37493205, 0.17053932, 0.03681842, 0.47968035, 0.37701832,
0.37726421, 0.40330905, 0.18313258, 0.13834622, 0.10686957,
0.08574271, 0.08932936, 0.1103552 , 0.08020868, 0.16700368,
0.11459243, 0.16583337, 0.42301807, 0.0622236 , 0.07930271,
0.15664053, 0.10900778, 0.39529481, 0.33023964, 0.08953824,
0.07669434, 0.32977777, 0.06687289, 0.14255103, 0.1082791 ,
0.1185603 , 0.07573903, 0.06777586, 0.15529698, 0.07774481,
0.1319333 , 0.46741034, 0.26553277, 0.11021778, 0.08353402,
0.07501521, 0.21975729, 0.15118047, 0.1010513 , 0.11295325,
0.1156205 , 0.09320525, 0.07624281, 0.3699689 , 0.36664176,
0.13115056, 0.38751319, 0.08558495, 0.15006362, 0.17043973,
0.14934202, 0.33910235, 0.12176975, 0.11124669, 0.0849782 ,
0.13092108, 0.18901446, 0.31970484, 0.08717229, 0.11604014,
0.1369188 , 0.06126725, 0.26907699, 0.03972285, 0.07861289,
0.18404379, 0.20342273, 0.27605085, 0.37368616, 0.16172745,
0.22718491, 0.39891815, 0.15493606, 0.07928769, 0.0991356 ,
0.10807809, 0.18863261, 0.19437603, 0.10087263, 0.09551228,
0.08680472, 0.08145378, 0.25633567, 0.0946826 , 0.10908433,
0.09835554, 0.14613165, 0.09936475, 0.25587927, 0.10673215,
0.43846917, 0.29142695, 0.2985442 , 0.36895964, 0.34085207,
0.39585659, 0.18172093, 0.38775081, 0.17556805, 0.13011722,
0.26601242, 0.19154015, 0.08362028, 0.19056907, 0.11512423,
0.08333659, 0.12326489, 0.37584131, 0.10396505, 0.131052  ,
0.07269795, 0.42077337, 0.09264653, 0.37953948, 0.37024166,
0.13408136, 0.25068087, 0.07705668, 0.06443674, 0.19556603,
0.08020543, 0.09845535, 0.35750609, 0.14861762, 0.0944147 ,
0.04674827, 0.07359262, 0.0823596 , 0.61042523, 0.11839773,
0.06868394, 0.22746347, 0.1239879 , 0.08485191, 0.13551024,
0.08900706, 0.09061777, 0.27400287, 0.10113894, 0.2115457 ,
0.23241958, 0.18093886, 0.40155105, 0.0790456 , 0.40719461,
0.37184078, 0.09663149, 0.39359592, 0.16304048, 0.07915816,
0.1412844 , 0.45936524, 0.09899119, 0.09323184, 0.25850093,
0.40708832, 0.08840097, 0.25185718, 0.38023652, 0.0787368 ,
0.42259366, 0.02252209, 0.07192629, 0.35775112, 0.17635003,
0.17685143, 0.24217863, 0.05887598, 0.11883841, 0.07751239,
0.09251758, 0.12099439, 0.09341267, 0.39819631, 0.09500474,
0.07226381, 0.36109457, 0.20154499, 0.10887599, 0.11686799,
```

0.10855128, 0.37260927, 0.14579988, 0.10093418, 0.12368258,
0.04980529, 0.24712553, 0.13167872, 0.08332578, 0.21779165,
0.07340445, 0.30218338, 0.07017299, 0.07041816, 0.08019985,
0.1807902 , 0.15951316, 0.08675146, 0.09750777, 0.35174463,
0.09440038, 0.39437658, 0.28753439, 0.08842532, 0.10153524,
0.28599078, 0.14869266, 0.09789696, 0.06904571, 0.07770622,
0.07639755, 0.08281354, 0.13296927, 0.14930948, 0.09012375,
0.0695222 , 0.121819 , 0.23377737, 0.04809784, 0.12743038,
0.10598232, 0.12519741, 0.39122306, 0.09388697, 0.08533637,
0.12394501, 0.15595166, 0.29824805, 0.20737571, 0.24249944,
0.26940217, 0.06672698, 0.3703321 , 0.11321707, 0.12151521,
0.20970129, 0.36434619, 0.36664311, 0.1812824 , 0.08877985,
0.22931058, 0.08990356, 0.09147812, 0.12030915, 0.16006243,
0.39234515, 0.20778619, 0.37578968, 0.32413011, 0.14495808,
0.0556566 , 0.08660817, 0.12778075, 0.09091598, 0.4100598 ,
0.17381585, 0.08475499, 0.09730965, 0.08170385, 0.43861013,
0.56429375, 0.40593983, 0.16999556, 0.23324026, 0.1426247 ,
0.12372962, 0.21241559, 0.30767591, 0.37854739, 0.29347749,
0.08517738, 0.09052858, 0.10633867, 0.10706916, 0.38865042,
0.28705896, 0.15953287, 0.09823414, 0.39514173, 0.08592139,
0.08623332, 0.09071556, 0.08941372, 0.14698016, 0.36999152,
0.18954574, 0.36167771, 0.05865457, 0.11450106, 0.21499677,
0.16386174, 0.08751644, 0.35939231, 0.14113122, 0.4592907 ,
0.17513692, 0.09220079, 0.38413941, 0.45984763, 0.09055174,
0.38614016, 0.09599833, 0.08691137, 0.31101671, 0.15610118,
0.10860601, 0.1344823 , 0.05028299, 0.11502768, 0.09598386,
0.12203014, 0.08998266, 0.41935876, 0.12565633, 0.04248853,
0.22837162, 0.28704025, 0.41373365, 0.0694922 , 0.20358289,
0.26020461, 0.09502352, 0.05054865, 0.10248057, 0.26018787,
0.09912734, 0.12676941, 0.18121535, 0.25092127, 0.09896052,
0.09919398, 0.105124 , 0.21279148, 0.04997992, 0.17496721,
0.06687006, 0.07807829, 0.3970969 , 0.09703026, 0.07293569,
0.11657908, 0.13491323, 0.10549609, 0.19161835, 0.35567392,
0.18989241, 0.15559372, 0.12952609, 0.38585722, 0.10881303,
0.06922562, 0.06312728, 0.09542586, 0.09053357, 0.20486105,
0.13448465, 0.40787826, 0.31314159, 0.07354585, 0.08164561,
0.4527594 , 0.40093248, 0.13098136, 0.0812744 , 0.06845612,
0.07571799, 0.18899018, 0.14788937, 0.35947803, 0.11075609,
0.070845 , 0.11050574, 0.40030832, 0.41206364, 0.45048731,
0.25002028, 0.40779612, 0.1122302 , 0.07894902, 0.13105809,
0.09081011, 0.15010633, 0.04614849, 0.17507438, 0.05237939,
0.0923017 , 0.23791382, 0.26471814, 0.09656709, 0.09534456,
0.3866224 , 0.11436475, 0.36622683, 0.43448298, 0.13277095,
0.06656   , 0.42168504, 0.05793141, 0.35923099, 0.07840215,
0.57156805, 0.06883012, 0.19556401, 0.35020006, 0.06865783,
0.39621494, 0.37782127, 0.38351622, 0.10171063, 0.31674394,
0.09158179, 0.08046139, 0.21736911, 0.06093041, 0.13218732,

0.39192343, 0.11238763, 0.35875094, 0.36236113, 0.06260985,
0.10891664, 0.2896653 , 0.12450635, 0.15834534, 0.08754686,
0.14356559, 0.14708162, 0.36436098, 0.38936476, 0.15071423,
0.05760624, 0.22229784, 0.05310266, 0.32543066, 0.13376758,
0.10417958, 0.38228982, 0.16356054, 0.12807461, 0.35795596,
0.05751894, 0.10214022, 0.15173813, 0.13489773, 0.11385102,
0.06565065, 0.10323535, 0.3141085 , 0.3782865 , 0.08603472,
0.35624452, 0.36115915, 0.3503121 , 0.06910297, 0.06498416,
0.45054588, 0.15216432, 0.13145808, 0.14777794, 0.07540679,
0.29690778, 0.08643149, 0.10259741, 0.09113231, 0.37663651,
0.34393181, 0.45284541, 0.12211431, 0.44772777, 0.37083283,
0.10961699, 0.34832967, 0.25097634, 0.20245819, 0.11437375,
0.09331645, 0.19038542, 0.07515629, 0.37790896, 0.37283709,
0.05654162, 0.07052224, 0.11723124, 0.07397143, 0.08245115,
0.10353816, 0.12522755, 0.32542927, 0.39036227, 0.50790772,
0.12778185, 0.04913548, 0.10030432, 0.14905704, 0.11323924,
0.19428863, 0.39547474, 0.08052503, 0.26644299, 0.45963568,
0.53145334, 0.41714462, 0.08579367, 0.3786521 , 0.40323593,
0.10557255, 0.09362745, 0.13485691, 0.12928144, 0.58282284,
0.2082515 , 0.34572697, 0.12434103, 0.08669091, 0.35537725,
0.39156083, 0.07767179, 0.08054582, 0.18365283, 0.10621065,
0.10426169, 0.11799756, 0.40251956, 0.49329901, 0.12666745,
0.40881789, 0.05387205, 0.08705249, 0.10398842, 0.17444165,
0.08003031, 0.08981622, 0.10129576, 0.14413844, 0.16083797,
0.09234315, 0.26687366, 0.10717202, 0.12354362, 0.14023058,
0.07875918, 0.12473849, 0.20084891, 0.44654028, 0.41379743,
0.11621092, 0.07720075, 0.30554375, 0.15368117, 0.40648235,
0.1203124 , 0.2939009 , 0.31836878, 0.22164822, 0.42373115,
0.34915597, 0.15791169, 0.21892608, 0.37158758, 0.05227016,
0.200432 , 0.12188184, 0.12815031, 0.36228052, 0.11254327,
0.121796 , 0.32348302, 0.06934303, 0.09188727, 0.42247644,
0.20880782, 0.11767141, 0.07704588, 0.11136951, 0.31900188,
0.37156286, 0.11264579, 0.09989899, 0.31645576, 0.10507371,
0.07207445, 0.14109635, 0.09413268, 0.15559008, 0.06026428,
0.33086091, 0.0602737 , 0.45703654, 0.09421442, 0.38032292,
0.13726772, 0.06815755, 0.11879527, 0.22652524, 0.08659523,
0.13112848, 0.20270764, 0.14510962, 0.27198668, 0.37328301,
0.07906904, 0.08153022, 0.10323733, 0.3628973 , 0.33637768,
0.09891968, 0.07031751, 0.13474328, 0.45407543, 0.08747866,
0.11972353, 0.08916808, 0.10074811, 0.27089297, 0.18170174,
0.1849244 , 0.11545036, 0.21683231, 0.30472767, 0.13949173,
0.07999946, 0.09955301, 0.09285114, 0.10145894, 0.37201208,
0.20252257, 0.09105741, 0.09122812, 0.32162685, 0.12159563,
0.3363362 , 0.09871737, 0.18301985, 0.06504304, 0.08246381,
0.09433815, 0.38107168, 0.23079482, 0.20927847, 0.37615733,
0.32447924, 0.11043518, 0.11501198, 0.09110092, 0.08738411,
0.08833829, 0.20632863, 0.39711755, 0.19381359, 0.25646894,

```
0.17694296, 0.22980964, 0.05337424, 0.12661552, 0.40290444,
0.05177162, 0.05946781, 0.14017372, 0.34511385, 0.40907887,
0.29958584, 0.15348545, 0.29536335, 0.064512  , 0.43160819,
0.27289943, 0.27915497, 0.43206882, 0.10780979, 0.08108301,
0.20227019, 0.07392932, 0.07207215, 0.10685248, 0.36663492,
0.32527512, 0.09990973, 0.07220888, 0.15531972, 0.38077144,
0.08378505, 0.38754279, 0.22643868, 0.04876317, 0.09073869,
0.11265903, 0.12307465, 0.12011509, 0.09590651, 0.17828502,
0.1056965 , 0.08145462, 0.33134871, 0.12127447, 0.43265128,
0.17540365, 0.09527274, 0.08977436, 0.09226111, 0.09891119,
0.13084288, 0.11848895, 0.11135195, 0.05703488, 0.14730285,
0.32020857, 0.05784168, 0.09079071, 0.1105165 , 0.40005342])
```