Task-D: Collinear features and their effect on linear models

In [55]:
```
%matplotlib inline
import warnings
warnings.filterwarnings("ignore")
import pandas as pd
import numpy as np
from sklearn.datasets import load_iris
from sklearn.model_selection import GridSearchCV
import seaborn as sns
import matplotlib.pyplot as plt
import sklearn.linear_model
```

In [56]:
```
data = pd.read_csv('task_d.csv')
```

In [57]:
```
data.head()
```

Out[57]:

	x	y	z	x*x	2*y	2*z+3*x*x	w	target
0	-0.581066	0.841837	-1.012978	-0.604025	0.841837	-0.665927	-0.536277	0
1	-0.894309	-0.207835	-1.012978	-0.883052	-0.207835	-0.917054	-0.522364	0
2	-1.207552	0.212034	-1.082312	-1.150918	0.212034	-1.166507	0.205738	0
3	-1.364174	0.002099	-0.943643	-1.280666	0.002099	-1.266540	-0.665720	0
4	-0.737687	1.051772	-1.012978	-0.744934	1.051772	-0.792746	-0.735054	0

In [58]:
```
X = data.drop(['target'], axis=1).values
Y = data['target'].values
#print(X)
```

Doing perturbation test to check the presence of collinearity

Task: 1 Logistic Regression

```
1. Finding the Correlation between the features
    a. check the correlation between the features
    b. plot heat map of correlation matrix using seaborn heatmap
2. Finding the best model for the given data
    a. Train Logistic regression on data(X,Y) that we have created in the above cell
    b. Find the best hyper prameter alpha with hyper parameter tuning using k-fold cross
validation (grid search CV or
    random search CV make sure you choose the alpha in log space)
    c. Creat a new Logistic regression with the best alpha
    (search for how to get the best hyper parameter value), name the best model as 'best_model'

3. Getting the weights with the original data
    a. train the 'best_model' with X, Y
    b. Check the accuracy of the model 'best_model_accuracy'
    c. Get the weights W using best_model.coef_

4. Modifying original data
    a. Add a noise(order of 10^-2) to each element of X
    and get the new data set X' (X' = X + e)
    b. Train the same 'best_model' with data (X', Y)
    c. Check the accuracy of the model 'best_model_accuracy_edited'
    d. Get the weights W' using best_model.coef_

5.  Checking deviations in metric and weights
    a. find the difference between 'best_model_accuracy_edited' and 'best_model_accuracy'
    b. find the absolute change between each value of W and W' ==> |(W-W')|
    c. print the top 4 features which have higher % change in weights
    compare to the other feature
```

Task: 2 Linear SVM

```
1. Do the same steps (2, 3, 4, 5) we have done in the above task 1.
```

Do write the observations based on the results you get from the deviations of weights in both Logistic Regression and linear SVM

Task: 1 Logistic Regression

In [59]:
```python
data.drop('target', axis = 1, inplace = True)
```

In [60]:
```python
# 1. Finding the Correlation between the features
corelations = data.corr()
```

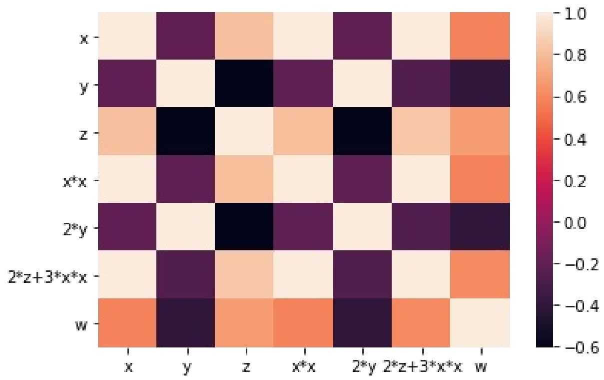
In [61]:
```python
corelations
```

Out[61]:

	x	y	z	x*x	2*y	2*z+3*x*x	w
x	1.000000	-0.205926	0.812458	0.997947	-0.205926	0.996252	0.583277
y	-0.205926	1.000000	-0.602663	-0.209289	1.000000	-0.261123	-0.401790
z	0.812458	-0.602663	1.000000	0.807137	-0.602663	0.847163	0.674486
x*x	0.997947	-0.209289	0.807137	1.000000	-0.209289	0.997457	0.583803
2*y	-0.205926	1.000000	-0.602663	-0.209289	1.000000	-0.261123	-0.401790
2*z+3*x*x	0.996252	-0.261123	0.847163	0.997457	-0.261123	1.000000	0.606860
w	0.583277	-0.401790	0.674486	0.583803	-0.401790	0.606860	1.000000

In [62]:
```python
sns.heatmap(corelations)
```

Out[62]: <AxesSubplot:>



In [63]:
```python
clf = sklearn.linear_model.SGDClassifier(max_iter=1000, loss = 'log')
Cs = np.logspace(-5,5,11)
tuned_parameters = [{'alpha': Cs}]
model = GridSearchCV(clf, tuned_parameters, scoring = 'accuracy', cv=4)
model.fit(X, Y)
```

Out[63]:
```
GridSearchCV(cv=4, estimator=SGDClassifier(loss='log'),
             param_grid=[{'alpha': array([1.e-05, 1.e-04, 1.e-03, 1.e-02, 1.e-01, 1.e+00, 1.e+01, 1.e+02,
       1.e+03, 1.e+04, 1.e+05])}],
             scoring='accuracy')
```

In [64]:
```python
#2.finding best hyper parameter
best_param = model.best_params_
print(best_param)
```

```
{'alpha': 0.0001}
```

In [65]:
```python
# 3. Getting the weights with the original data
best_model = sklearn.linear_model.SGDClassifier(alpha = 1e-05, loss = 'log')
best_model.fit(X,Y)
best_model_accuracy = best_model.score(X, Y)
print(best_model_accuracy)
W = best_model.coef_
print(W)
```

```
1.0
[[ 12.03098932 -23.27128945  23.9332139   14.31455572 -23.27128945
   15.77274148  13.19109547]]
```

In [67]:
```python
#4. Modifying original data
# a. Add a noise(order of 10^-2) to each element of X and get the new data set X' (X' = X + e)
df = pd.read_csv('task_d.csv')
```

```
df.drop(['target'], axis = 1, inplace = True)
df.applymap(lambda x: x + 0.01)
X_noise = df.values
```

In [69]:
```
# b. Train the same 'best_model' with data (X', Y)
best_model = sklearn.linear_model.SGDClassifier(alpha = 1e-05, loss = 'log')
best_model.fit(X_noise,Y)
best_model_accuracy_edited = best_model.score(X_noise, Y)
print(best_model_accuracy_edited)
W_noise = best_model.coef_
print(W_noise)
```

```
1.0
[[ 21.71700043 -28.9284769   37.41549186  20.74322511 -28.9284769
   23.18634811   8.14267415]]
```

In [70]:
```
#5.  Checking deviations in metric and weights
```

In [71]:
```
#a. find the difference between 'best_model_accuracy_edited' and 'best_model_accuracy'
best_model_accuracy_edited - best_model_accuracy
```

Out[71]:
```
0.0
```

In [72]:
```
#b. find the absolute change between each value of W and W' ==> |(W-W')|
abs = np.abs(W - W_noise)
abs
```

Out[72]:
```
array([[ 9.68601111,  5.65718746, 13.48227796,  6.42866939,  5.65718746,
         7.41360662,  5.04842132]])
```

In [73]:
```
l = sorted(abs[0], reverse = True)
cols = data.columns
cols
```

Out[73]:
```
Index(['x', 'y', 'z', 'x*x', '2*y', '2*z+3*x*x', 'w'], dtype='object')
```

In [74]:
```
# c. print the top 4 features which have higher % change in weights compare to the other feature
def Top_4_feature(abs):
    list = []
    for i in range(4):
        idx = abs.argmax()
        print(idx)
        list.append('Feature Name ' + cols[idx] + ' : ' + str(l[i]))
        abs = np.delete(abs, idx)
    return list
```

In [75]:
```
Top_4_feature(abs)
```

```
2
0
3
1
```

Out[75]:
```
['Feature Name z : 13.482277956714597',
 'Feature Name x : 9.686011113345108',
 'Feature Name x*x : 7.4136066229736155',
 'Feature Name y : 6.428669387808686']
```

Task: 2 Linear SVM

In [40]:
```
clf = sklearn.linear_model.SGDClassifier(max_iter=1000, loss = 'hinge')
Cs = np.logspace(-5,5,11)
tuned_parameters = [{'alpha': Cs}]
model = GridSearchCV(clf, tuned_parameters, scoring = 'accuracy', cv=4)
model.fit(X, Y)
```

Out[40]:
```
GridSearchCV(cv=4, estimator=SGDClassifier(),
             param_grid=[{'alpha': array([1.e-05, 1.e-04, 1.e-03, 1.e-02, 1.e-01, 1.e+00, 1.e+01, 1.e+02,
       1.e+03, 1.e+04, 1.e+05])}],
             scoring='accuracy')
```

In [41]:
```
#2.finding best hyper parameter
best_param = model.best_params_
print(best_param)
```

```
{'alpha': 1e-05}
```

In [42]:
```
# 3. Getting the weights with the original data
best_model = sklearn.linear_model.SGDClassifier(alpha = 1e-05, loss = 'hinge')
best_model.fit(X,Y)
best_model_accuracy = best_model.score(X, Y)
print(best_model_accuracy)
W = best_model.coef_
print(W)
```

```
1.0
[[ 17.77791249 -19.57998183  37.12646288  14.98991774 -19.57998183
   17.97342313 -14.69082429]]
```

In [43]:
```
# b. Train the same 'best_model' with data (X', Y)
best_model = sklearn.linear_model.SGDClassifier(alpha = 1e-05, loss = 'hinge')
best_model.fit(X_noise,Y)
best_model_accuracy_edited = best_model.score(X_noise, Y)
print(best_model_accuracy_edited)
W_noise = best_model.coef_
print(W_noise)
```

```
1.0
[[ 27.80684591 -28.9035864   64.61034056  22.61125655 -28.9035864
   28.15088679  24.33218649]]
```

In [44]:
```
#5.  Checking deviations in metric and weights
```

In [45]:
```
#a. find the difference between 'best_model_accuracy_edited' and 'best_model_accuracy'
best_model_accuracy_edited - best_model_accuracy
```

Out[45]:
```
0.0
```

In [46]:
```
#b. find the absolute change between each value of W and W' ==> |(W-W')|
abs = np.abs(W - W_noise)
abs
```

Out[46]:
```
array([[10.02893343,  9.32360457, 27.48387767,  7.6213388 ,  9.32360457,
        10.17746365, 39.02301078]])
```

In [47]:
```
l = sorted(abs[0], reverse = True)
cols = data.columns
cols
```

Out[47]:
```
Index(['x', 'y', 'z', 'x*x', '2*y', '2*z+3*x*x', 'w'], dtype='object')
```

In [48]:
```
# c. print the top 4 features which have higher % change in weights compare to the other feature
def Top_4_feature(abs):
    list = []
    for i in range(4):
        idx = abs.argmax()
        list.append('Feature Name ' + cols[idx] + ' : ' + str(l[i]))
        abs = np.delete(abs, idx)
    return list
```

In [49]:
```
Top_4_feature(abs)
```

Out[49]:
```
['Feature Name w : 39.02301078138523',
 'Feature Name z : 27.483877674389305',
 'Feature Name 2*y : 10.177463653563741',
 'Feature Name x : 10.02893342611118']
```

Observation Task 1: deviations on weights(Logistic regression) a) more deviations when colinearity between the features are high otherwise only a min deviation b) executing the model again and again based on the weight deviation and colinearity the values should be increased c) Implementation done on

Logistictic loss Task 2: deviations on weights(SVM) a) more deviations when colinearity between the features are high otherwise only a min deviation b) executing the model again and again based on the weight deviation and colinearity the values should be increased c) Implementation done on hinge loss

In [ ]:

In [ ]: