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

In [1]:

```
# necessary libraries
%matplotlib inline
import warnings
warnings.filterwarnings("ignore")
import pandas as pd
import numpy as np
from sklearn.linear_model import SGDClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import GridSearchCV
import seaborn as sns
import matplotlib.pyplot as plt
```

In [2]:

```
# read the dataset
data = pd.read_csv('task_d.csv')
data.shape
```

Out[2]:

```
(100, 8)
```

In [3]:

```
data.head(6) # display first few rows of data
```

Out[3]:

	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
5	-0.111201	1.681575	-0.804974	-0.164556	1.681575	-0.245289	0.489916	0

In [23]:
```
# obtain the Input & output variables:
X = data.drop(['target'], axis=1).values
Y = data['target'].values
```

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
```

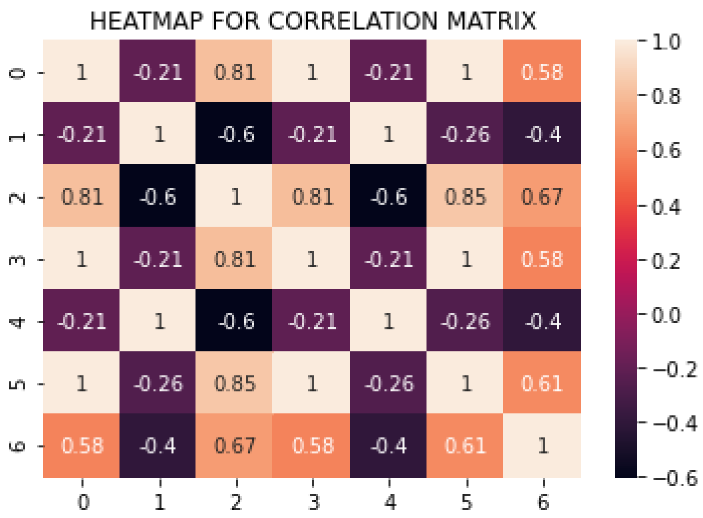
Task: 2 Linear SVM

```
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

In [24]:

```
# Correlation of features:
plt.tight_layout()
cor = pd.DataFrame(X).corr()
sns.heatmap(cor,annot = True,cbar = True)
plt.title("HEATMAP FOR CORRELATION MATRIX",fontsize = 12)
plt.show()
```



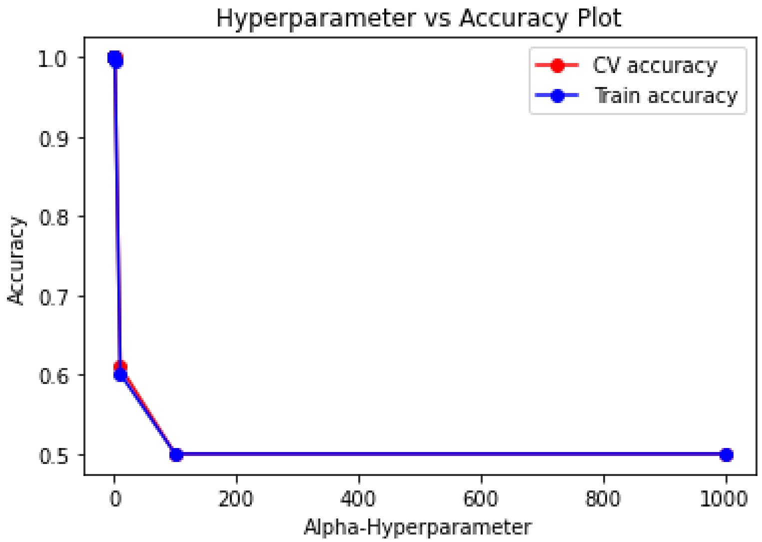
EFFECT OF COLLINEAR FEATURES ON LOGISTIC REGRESSION

In [29]:

```
# Apply Logistic regression along with gridsearchCV:

alphas = [0.0001,0.001,0.01,0.1,1,10,100,1000]
```

```
params = {'loss':['log'],'alpha' :[0.0001,0.001,0.01,0.1,1,10,100,1000]}

# Initialize the estimator with the parameters:
estimator = SGDClassifier()

clf = GridSearchCV(estimator,param_grid = params,cv = 10,
                   scoring = 'accuracy',return_train_score=True)
clf = clf.fit(X,Y)

best_params = clf.best_params_
print("The best parameter :",best_params) # ->we got 0.0001 as our best alpha

# cv results for visually finding the best parameter alpha:
cv_results = pd.DataFrame(clf.cv_results_)
cv_score = list(cv_results['mean_test_score'])
print("cv scores :-\n",cv_score)
train_score = list(cv_results['mean_train_score'])
print("train score :-\n",train_score)

# lets plot the results:
plt.plot(alphas,cv_score,c = 'r',label = 'CV accuracy',marker = 'o')
plt.plot(alphas,train_score,c = 'b',label = 'Train accuracy',marker = 'o')
plt.xlabel("Alpha-Hyperparameter")
plt.ylabel("Accuracy")
plt.title("Hyperparameter vs Accuracy Plot")
plt.legend()
plt.show()
```

```
The best parameter : {'alpha': 0.0001, 'loss': 'log'}
cv scores :-
 [1.0, 1.0, 1.0, 1.0, 1.0, 0.61, 0.5, 0.5]
train score :-
 [1.0, 1.0, 1.0, 1.0, 0.9966666666666667, 0.6011111111111112, 0.5, 0.5]
```



OBSERVATION :

Here, we can infer that for both training data and as well as CV data, the best parameters of the model indicates overfitting which could be the cause of some collinearity present in the data.

In [30]:

```
# Train Log Regression with the best alpha :

from sklearn.metrics import accuracy_score

#initialize the best model & fit to the training data
best_model = SGDClassifier(loss = 'log',alpha = 0.0001).fit(X,Y)

ypredicted = best_model.predict(X) # obtain the predicted values for the train data

best_model_accuracy = accuracy_score(Y,ypredicted) # accuracy of the best model

W = best_model.coef_ # weights of the best model

print("Best model training accuracy :-",best_model_accuracy)
print("Optimal Weights of the best model :-\n",W[0])
```

```
Best model training accuracy :- 1.0
```

```
Optimal Weights of the best model :-
 [ 7.1915934  -9.39088705 15.1388034   6.07566793 -9.39088705  7.29589796
  1.94796763]
```

In [31]:

```python
# Modifying original data for a perturbation test:

X_dash = pd.DataFrame(X) + 0.01 # adding a small noise

best_model = SGDClassifier(loss = 'log',alpha = 0.0001).fit(X_dash,Y) # fitting the best model to the new data

ypredicted = best_model.predict(X_dash) # prediction values
best_model_accuracy_edited = accuracy_score(Y,ypredicted) # accuracy scores
W_dash = best_model.coef_ # model weights

print("Best model training accuracy after perturbation :-",best_model_accuracy_edited)
print("Optimal Weights of the best model after perturbation :-\n",W_dash[0])
```

```
Best model training accuracy after perturbation :- 1.0
Optimal Weights of the best model after perturbation :-
 [ 3.81599253 -3.63631809 10.08599659  3.6723088  -3.63631809  4.52155832
  0.8193584 ]
```

In [33]:

```python
# Checking deviation in metric and weights:

accuracy_diff = (best_model_accuracy - best_model_accuracy_edited)
weights_diff = abs(W[0] - W_dash[0])

print("(A) The difference in accuracy : \n",accuracy_diff)
print("\n")
print("(B) The difference in weights : \n",weights_diff)
print("\n")

# percentage change in weights before and after adding a small noise:
percent_change = [(weights_diff[j]/abs(W_dash[0][j]))*100 for j in range(len(weights_diff))]
print("(C) The %change in weights : \n",percent_change)
print("\n")

# Top 4 features that has high percentage change in weights:
print("(D) The Top 4 features with high %change in weights : \n")
print("(i)   feature (y)")
print("(ii)  feature (2*y)")
```

```
print("(iii) feature (w)")
print("(iv)  feature (x)")
```

```
(A) The difference in accuracy :
 0.0


(B) The difference in weights :
 [3.37560087 5.75456896 5.05280682 2.40335914 5.75456896 2.77433964
 1.12860923]


(C) The %change in weights :
 [88.4593156093714, 158.25262874571155, 50.09724894739781, 65.44545328292116, 158.25262874571155, 61.35804171164554,
137.7430480985557]


(D) The Top 4 features with high %change in weights :

(i)   feature (y)
(ii)  feature (2*y)
(iii) feature (w)
(iv)  feature (x)
```

OBSERVATION :

Here, its evident that the weights change drastically for the collinear features and that is because of various dependencies between the features which is nullifying the actual effect of the optimal model. Thus, we can opt for feature forward selection to obtain useful features.

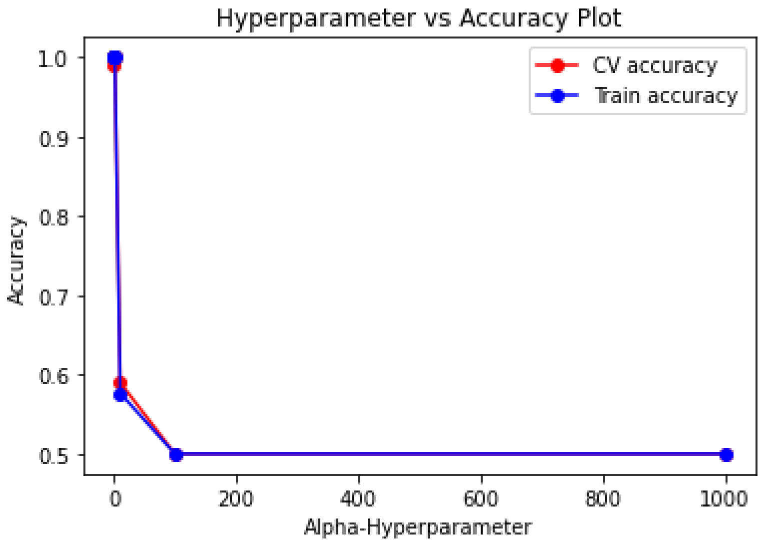
EFFECT OF COLLINEAR FEATURES ON SVM

In [38]:

```
# Apply SVM along with gridsearchCV:

alphas = [0.0001,0.001,0.01,0.1,1,10,100,1000]
params = {'loss':['hinge'],'alpha' :[0.0001,0.001,0.01,0.1,1,10,100,1000]}

# Initialize the estimator with the parameters:
estimator = SGDClassifier()

clf = GridSearchCV(estimator,param_grid = params,cv = 10,
                   scoring = 'accuracy',return_train_score=True)
```

```
clf = clf.fit(X,Y)

best_params = clf.best_params_
print("The best parameter :",best_params) # ->we got 0.01 as our best alpha

# cv results for visually finding the best parameter alpha:
cv_results = pd.DataFrame(clf.cv_results_)
cv_score = list(cv_results['mean_test_score'])
print("cv scores :-\n",cv_score)
train_score = list(cv_results['mean_train_score'])
print("train score :-\n",train_score)

# lets plot the results:
plt.plot(alphas,cv_score,c = 'r',label = 'CV accuracy',marker = 'o')
plt.plot(alphas,train_score,c = 'b',label = 'Train accuracy',marker = 'o')
plt.xlabel("Alpha-Hyperparameter")
plt.ylabel("Accuracy")
plt.title("Hyperparameter vs Accuracy Plot")
plt.legend()
plt.show()
```

```
The best parameter : {'alpha': 0.001, 'loss': 'hinge'}
cv scores :-
 [0.99, 1.0, 1.0, 1.0, 1.0, 0.59, 0.5, 0.5]
train score :-
 [1.0, 1.0, 1.0, 1.0, 1.0, 0.5766666666666667, 0.5, 0.5]
```



OBSERVATION :

Here, we can infer that for both training data and as well as CV data, the best parameters of the model indicates overfitting which could be the cause of some collinearity present in the data.

In [39]:

```
# Train SVM with the best alpha :

from sklearn.metrics import accuracy_score

best_model = SGDClassifier(loss = 'hinge',alpha = 0.001).fit(X,Y) # fit the best model with best parameter alpha

ypredicted = best_model.predict(X) # obtain the prediction value

best_model_accuracy = accuracy_score(Y,ypredicted) # obtain the accuracy value

W = best_model.coef_

print("Best model training accuracy :-",best_model_accuracy)
print("Optimal Weights of the best model :-\n",W[0])
```

```
Best model training accuracy :- 1.0
Optimal Weights of the best model :-
```

```
[ 1.78622886 -2.15482417  3.32111991  1.49565149 -2.15482417  1.7470662
 0.1822998 ]
```

In [40]:

```
# Modifying original data for a perturbation test:

X_dash = pd.DataFrame(X) + 0.01  # adding a small noise
X_dash.head()

best_model = SGDClassifier(loss = 'hinge',alpha = 0.001).fit(X_dash,Y) # fitting the best model to the new data

ypredicted = best_model.predict(X_dash) # obtain prediction value
best_model_accuracy_edited = accuracy_score(Y,ypredicted)# accuracy scores
W_dash = best_model.coef_ # weights learnt

print("Best model training accuracy after perturbation :-",best_model_accuracy_edited)
print("Optimal Weights of the best model after perturbation :-\n",W_dash[0])
```

```
Best model training accuracy after perturbation :- 1.0
Optimal Weights of the best model after perturbation :-
 [ 2.30145113 -2.35167269  6.56801264  1.90842941 -2.35167269  2.51036222
  1.68144053]
```

In [42]:

```
# Checking deviation in metric and weights:

accuracy_diff = (best_model_accuracy - best_model_accuracy_edited)
weights_diff = abs(W[0] - W_dash[0])

print("(A) The difference in accuracy : \n",accuracy_diff)
print("\n")
print("(B) The difference in weights : \n",weights_diff)
print("\n")

# percentage change in weights before and after adding a small noise:
percent_change = [(weights_diff[j]/abs(W_dash[0][j]))*100 for j in range(len(weights_diff))]
print("(C) The %change in weights : \n",percent_change)
print("\n")

# Top 4 features that has high percentage change in weights:
print("(D) The Top 4 features with high %change in weights : \n")
print("(i)   feature (w)")
print("(ii)  feature (z)")
```

```
print("(iii) features (2*z+3*x*x)")
print("(iv)  feature (x)")
```

```
(A) The difference in accuracy :
 0.0


(B) The difference in weights :
 [0.51522227 0.19684852 3.24689273 0.41277791 0.19684852 0.76329601
 1.49914073]


(C) The %change in weights :
 [22.386843819049503, 8.370574806966612, 49.43493426795867, 21.629194782481616, 8.370574806966612, 30.40581192687493
2, 89.15811804625909]


(D) The Top 4 features with high %change in weights :

(i)   feature (w)
(ii)  feature (z)
(iii) features (2*z+3*x*x)
(iv)  feature (x)
```

OBSERVATION :

Here, its evident that the weights change drastically for the collinear features and that is because of various dependencies between the features which is nullifying the actual effect of the optimal model. Thus, we can opt for feature forward selection to obtain useful features.