

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

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

```
from google.colab import files
files=files.upload()
```

task_d.csv

- **task_d.csv**(application/vnd.ms-excel) - 14099 bytes, last modified: 9/2/2019 - 100% done
Saving task_d.csv to task_d.csv

```
data = pd.read_csv('task_d.csv')
```

```
data.head()
```

	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

```
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

- 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

- Train Logistic regression on data(X,Y) that we have created in the above cell
- Find the best hyper parameter alpha with hyper parameter tuning using k-fold cross random search CV make sure you choose the alpha in log space)
- Create a new Logistic regression with the best alpha
(search for how to get the best hyper parameter value), name the best model as 'best

3. Getting the weights with the original data

- train the 'best_model' with X, Y
- Check the accuracy of the model 'best_model_accuracy'
- Get the weights W using best_model.coef_

4. Modifying original data

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

5. Checking deviations in metric and weights

- find the difference between 'best_model_accuracy_edited' and 'best_model_accuracy'
- find the absolute change between each value of W and W' ==> $|(W-W')|$
- print the top 4 features which have higher % change in weights compare to the other feature

Task: 2 Linear SVM

- 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

▼ 1. Finding the Correlation between the features:

- Check the correlation between the features
- Plot heat map of correlation matrix using seaborn heatmap

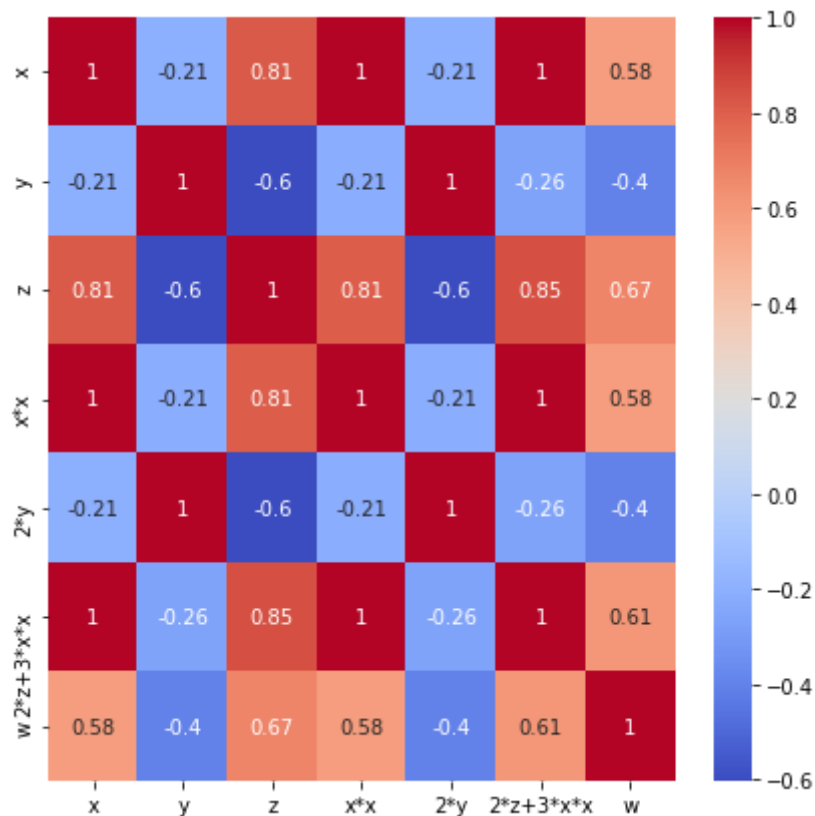
```
data[['x', 'y', 'z', 'x*x', '2*y', '2*z+3*x*x', 'w']].corr()
```

	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

```
plt.figure(figsize=(7,7))
```

```
sns.heatmap(data[['x', 'y', 'z', 'x*x', '2*y', '2*z+3*x*x', 'w']].corr(), annot=True, cmap
```

<matplotlib.axes._subplots.AxesSubplot at 0x7fa37e4a5ed0>



2. Finding the best model for the given data:

1. Train Logistic regression on data(X,Y) that we have created in the above cell
2. Find the best hyper parameter 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)
3. Create 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'

```
clf = SGDClassifier(loss = 'log', random_state = 30)
param = {'alpha': [0.00001, 0.0001, 0.001, 0.01, 0.1, 1, 10, 100, 1000]}
grid_search = GridSearchCV(clf, param, cv=3)
grid_search.fit(X,Y)
```

```
grid_search.best_params_
```

```
{'alpha': 0.001}
```

```
best_model = SGDClassifier(loss = 'log', alpha = 0.001)
```

```
best_model.fit(X,Y)
```

```
best_model
```

```
SGDClassifier(alpha=0.001, average=False, class_weight=None,
              early_stopping=False, epsilon=0.1, eta0=0.0, fit_intercept=True,
              l1_ratio=0.15, learning_rate='optimal', loss='log', max_iter=1000,
              n_iter_no_change=5, n_jobs=None, penalty='l2', power_t=0.5,
              random_state=None, shuffle=True, tol=0.001,
              validation_fraction=0.1, verbose=0, warm_start=False)
```

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_

```
from sklearn.metrics import accuracy_score
```

```
y_pred = best_model.predict(X)
```

```
best_model_accuracy = accuracy_score(Y, y_pred)
```

```
best_model_accuracy
```

```
1.0
```

```
weights = best_model.coef_
```

```
weights
```

```
array([[ 1.47032425, -1.97639369,  2.92045048,  1.23418187, -1.97639369,
         1.46336743,  0.7252211 ]])
```

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_

```
X_noise = [i + 0.01 for i in X]
```

```
best_model_ed = SGDClassifier(loss = 'log', alpha = 0.001)
```

```
best_model_ed.fit(X_noise,Y)
```

```
best_model_ed
```

```
SGDClassifier(alpha=0.001, average=False, class_weight=None,
              early_stopping=False, epsilon=0.1, eta0=0.0, fit_intercept=True,
```

```
l1_ratio=0.15, learning_rate='optimal', loss='log', max_iter=1000,
n_iter_no_change=5, n_jobs=None, penalty='l2', power_t=0.5,
random_state=None, shuffle=True, tol=0.001,
validation_fraction=0.1, verbose=0, warm_start=False)
```

```
y_pred = best_model_ed.predict(X_noise)
best_model_accuracy_edited = accuracy_score(Y, y_pred)
best_model_accuracy_edited
```

```
1.0
```

```
weights_of_best_model_accuracy_edited = best_model_ed.coef_
weights_of_best_model_accuracy_edited
```

```
array([[ 1.459417, -1.50846005,  3.06412059,  1.25902788, -1.50846005,
         1.50301719,  0.93530803]])
```

*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

```
acc_diff = best_model_accuracy - best_model_accuracy_edited
acc_diff
```

```
0.0
```

```
w_change = abs(weights - weights_of_best_model_accuracy_edited)
w_change = w_change[0]
```

```
percentage_change = []
```

```
for i in range(len(w_change)):
    percentage_change.append((w_change[i] / weights[0][i]) * 100)
```

```
change_lr = np.argsort(np.array(percentage_change))[:, -1]
features = data.columns.drop("target").values
for i in change_lr[0:4]:
    print(weights[0][i], "--->", features[i])
```

```
0.7252210982973472 ---> w
2.920450478374889 ---> z
1.4633674307855913 ---> 2*z+3*x*x
1.234181871482873 ---> x*x
```

Observations:

In Logistic Regression while performing perturbation test to check the features are of collinear or not, weights of the original and perturbed features are significantly different, Edited

Weights(W) will not be considered for the feature importance.

Task: 2 Linear SVM

```

clf = SGDClassifier(random_state = 30)
param = {'alpha': [0.00001, 0.0001, 0.001, 0.01, 0.1, 1, 10, 100, 1000]}
grid_search = GridSearchCV(clf, param, cv=3)
grid_search.fit(X,Y)
grid_search.best_params_

{'alpha': 0.01}

best_model_svm = SGDClassifier(alpha = 0.01)
best_model_svm.fit(X,Y)
best_model_svm

SGDClassifier(alpha=0.01, average=False, class_weight=None,
              early_stopping=False, epsilon=0.1, eta0=0.0, fit_intercept=True,
              l1_ratio=0.15, learning_rate='optimal', loss='hinge',
              max_iter=1000, n_iter_no_change=5, n_jobs=None, penalty='l2',
              power_t=0.5, random_state=None, shuffle=True, tol=0.001,
              validation_fraction=0.1, verbose=0, warm_start=False)

from sklearn.metrics import accuracy_score
y_pred_svm = best_model_svm.predict(X)
best_model_accuracy_svm = accuracy_score(Y, y_pred)
best_model_accuracy_svm

1.0

w_svm = best_model.coef_
w_svm

array([[ 1.47032425, -1.97639369,  2.92045048,  1.23418187, -1.97639369,
         1.46336743,  0.7252211 ]])

X_noise_svm = [i + 0.01 for i in X]

best_model_svm_ed = SGDClassifier(alpha = 0.01)
best_model_svm_ed.fit(X_noise_svm,Y)
best_model_svm_ed

SGDClassifier(alpha=0.01, average=False, class_weight=None,
              early_stopping=False, epsilon=0.1, eta0=0.0, fit_intercept=True,
              l1_ratio=0.15, learning_rate='optimal', loss='hinge',
              max_iter=1000, n_iter_no_change=5, n_jobs=None, penalty='l2',
              power_t=0.5, random_state=None, shuffle=True, tol=0.001,
              validation_fraction=0.1, verbose=0, warm_start=False)

y_pred_svm_ed = best_model_svm_ed.predict(X_noise)
acc_svm_ed = accuracy_score(Y, y_pred_svm_ed)

```

```
acc_svm_ed
```

```
1.0
```

```
w_svm_ed = best_model_svm_ed.coef_  
w_svm_ed
```

```
array([[ 0.50594224, -0.37885177,  1.24647823,  0.46908487, -0.37885177,  
        0.57279916,  0.00868479]])
```

```
acc_diff_svm = best_model_accuracy_svm - acc_svm_ed  
acc_diff_svm
```

```
0.0
```

```
w_change_svm = abs(w_svm - w_svm_ed)  
w_change_svm
```

```
array([[0.96438201, 1.59754192, 1.67397225, 0.765097 , 1.59754192,  
        0.89056827, 0.71653631]])
```

```
w_change_svm = w_change_svm[0]  
percentage_change_svm = []
```

```
for i in range(len(w_change_svm)):  
    percentage_change_svm.append((w_change_svm[i] / w_svm[0][i]) * 100)
```

```
change_svm = np.argsort(np.array(percentage_change_svm))[:-1]  
features = data.columns.drop("target").values
```

```
for i in change_svm[0:4]:  
    print(w_svm[0][i], "--->", features[i])
```

```
0.7252210982973472 ---> w  
1.4703242523523743 ---> x  
1.234181871482873 ---> x*x  
1.4633674307855913 ---> 2*z+3*x*x
```

Observations:

- From the percentage value difference between the original & perturbation study, resulted change in weights in Logistic Regression is slightly higher than the resulted change in weights in SVM. So multi-collinearity is less stable in Logistic regression than linear SVM whereas, resulting change in weights in Logistic Regression varies alot.

✓ 0s completed at 9:09 PM

● ✕