Homework 6

YIMIN LI

In [113]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.svm import SVC
from sklearn.model_selection import train_test_split, cross_val_score
from sklearn.linear_model import LogisticRegression
import warnings
warnings.filterwarnings('ignore')
```

1.

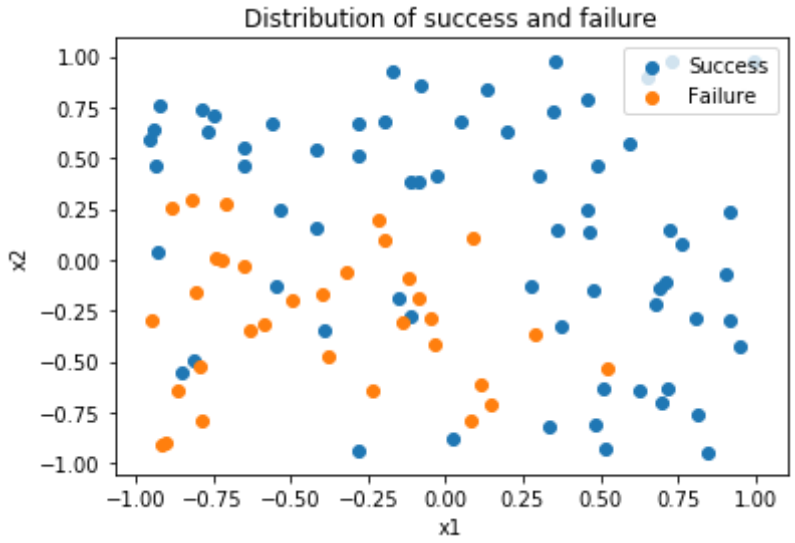
In [50]:
```
x1 = np.random.uniform(-1, 1, 100)
x2 = np.random.uniform(-1, 1, 100)
err = np.random.normal(0, 0.5, 100)
```

In [51]:
```
y = x1 + x1 * x1 + x2 + x2 * x2 + err
```

In [52]:
```
success = y >= 0
failure = y < 0
```

In [53]:
```
plt.scatter(x1[success], x2[success])
plt.scatter(x1[failure], x2[failure])
plt.xlabel('x1')
plt.ylabel('x2')
plt.legend(['Success', 'Failure'],loc=1)
```

```
plt.title('Distribution of success and failure')
plt.show()
```



In [54]:
```
X = np.column_stack((x1, x2))
X = pd.DataFrame(X)
x_train, x_test, y_train, y_test = train_test_split(X, success, test_si
ze=0.2)
```

In [55]:
```
svm_radial = SVC().fit(x_train, y_train)
svm_linear = SVC(kernel='linear').fit(x_train, y_train)
```

In [56]:
```
print("Training Error:")
print("SVM with radial kernel:" + str(1 - svm_radial.score(x_train, y_t
rain)))
print("SVM with linear kernel:" + str(1 - svm_linear.score(x_train, y_t
rain)))
print("Test Error:")
print("SVM with radial kernel:" + str(1 - svm_radial.score(x_test, y_te
st)))
print("SVM with linear kernel:" + str(1 - svm_linear.score(x_test, y_te
st)))
```

```
Training Error:
SVM with radial kernel:0.15000000000000002
SVM with linear kernel:0.22499999999999998

Test Error:
SVM with radial kernel:0.25
SVM with linear kernel:0.30000000000000004
```

As is shown above, we would see that SVM with radial kerneling performs better than linear kernel SVM in both training and testing error.

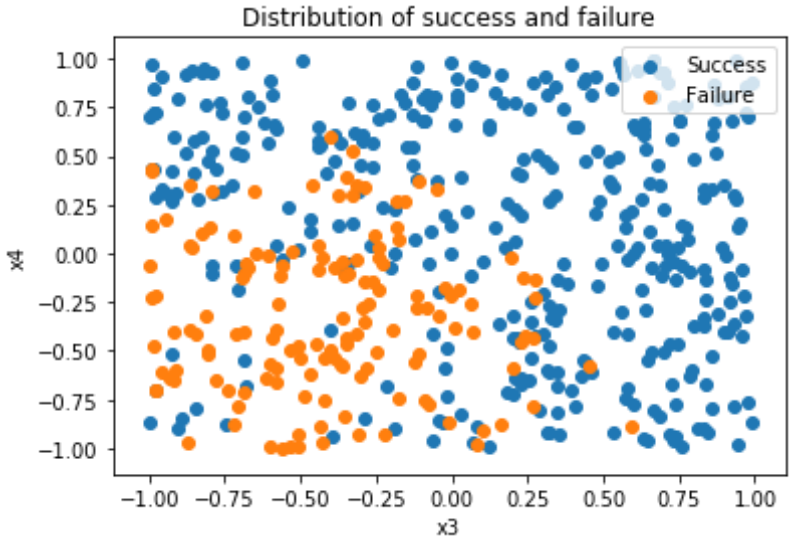
SVM vs. logistic regression

2.

In [60]:
```
x3 = np.random.uniform(-1, 1, 500)
x4 = np.random.uniform(-1, 1, 500)
err2  = np.random.normal(0, 0.5, 500)
y2 = x3 + x3 * x3 + x4 + x4 * x4 + err2
success2 = y2 >= 0
failure2 = y2 < 0
```

3.

In [63]:
```
plt.scatter(x3[success2], x4[success2])
plt.scatter(x3[failure2], x4[failure2])
plt.xlabel('x3')
plt.ylabel('x4')
plt.legend(['Success', 'Failure'], loc=1)
plt.title('Distribution of success and failure')
plt.show()
```

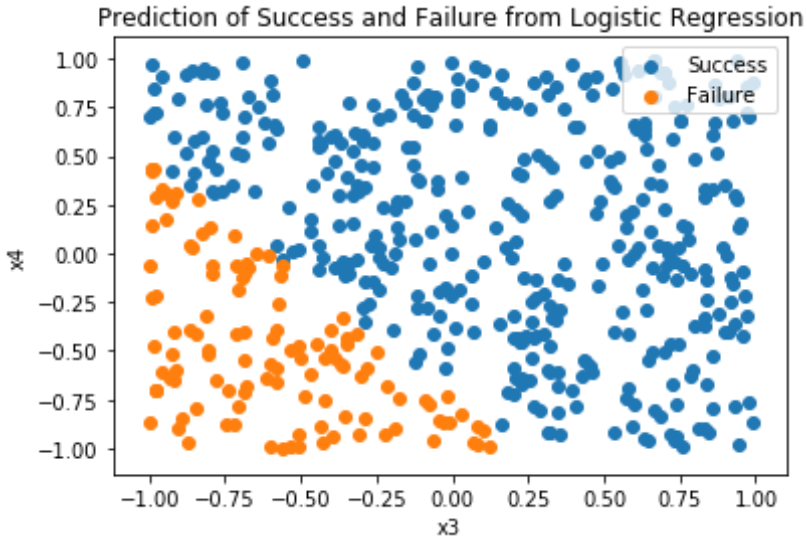


4.

In [71]:
```
X2 = np.column_stack((x3, x4))
X2 = pd.DataFrame(X2)
lr = LogisticRegression().fit(X2, success2)
```

5.

In [72]:
```
lr_predict = lr.predict(X2)
plt.scatter(x3[lr_predict], x4[lr_predict])
plt.scatter(x3[~lr_predict], x4[~lr_predict])
plt.xlabel('x3')
plt.ylabel('x4')
plt.legend(['Success', 'Failure'], loc=1)
plt.title('Prediction of Success and Failure from Logistic Regression')
plt.show()
```



6.

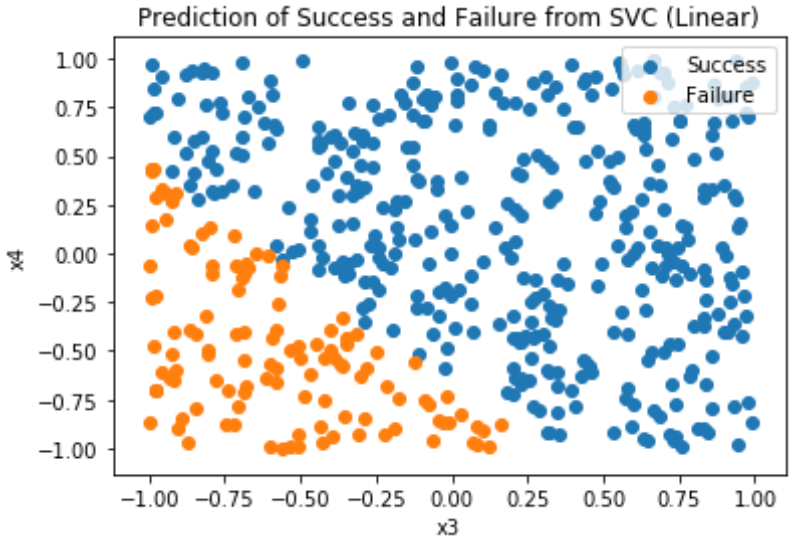
In [75]:
```
X3 = np.column_stack((x3 * x3 - 1, x4 * x4 - 0.5))
X3 = pd.DataFrame(X3)
lr2 = LogisticRegression().fit(X3, success2)
```

7.

In [76]:
```
lr_predict_3 = lr.predict(X3)
plt.scatter(x3[lr_predict_3], x4[lr_predict_3])
plt.scatter(x3[~lr_predict_3], x4[~lr_predict_3])
plt.xlabel('x3')
plt.ylabel('x4')
plt.legend(['Success', 'Failure'], loc=1)
plt.title('Prediction of Success and Failure from Logistic Regression
 (non-linear)')
plt.show()
```

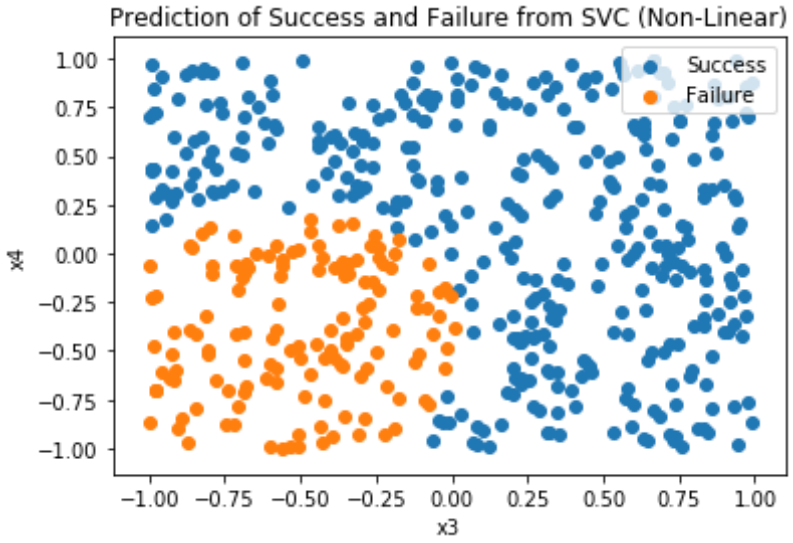
8.

In [78]:
```
svm = SVC(kernel='linear').fit(X2, success2)
svm_predict = svm.predict(X2)
plt.scatter(x3[svm_predict], x4[svm_predict])
plt.scatter(x3[~svm_predict], x4[~svm_predict])
plt.xlabel('x3')
plt.ylabel('x4')
plt.legend(['Success', 'Failure'], loc=1)
plt.title('Prediction of Success and Failure from SVC (Linear)')
plt.show()
```



9.

In [79]:
```
svm2 = SVC(gamma='scale').fit(X2, success2)
svm_predict2 = svm2.predict(X2)
plt.scatter(x3[svm_predict2], x4[svm_predict2])
plt.scatter(x3[~svm_predict2], x4[~svm_predict2])
plt.xlabel('x3')
plt.ylabel('x4')
plt.legend(['Success', 'Failure'], loc=1)
plt.title('Prediction of Success and Failure from SVC (Non-Linear)')
plt.show()
```



10.

In [81]:
```
print("Accuracy Score:")
print("Logistic Regression (linear):" + str(lr.score(X2, success2)))
print("Logistic Regression (non-linear):" + str(lr2.score(X3, success2
)))
print("SVM (linear):" + str(svm.score(X2, success2)))
print("SVM (non-linear):" + str(svm2.score(X2, success2)))
```

```
Accuracy Score:
Logistic Regression (linear):0.81
Logistic Regression (non-linear):0.712
SVM (linear):0.816
SVM (non-linear):0.854
```

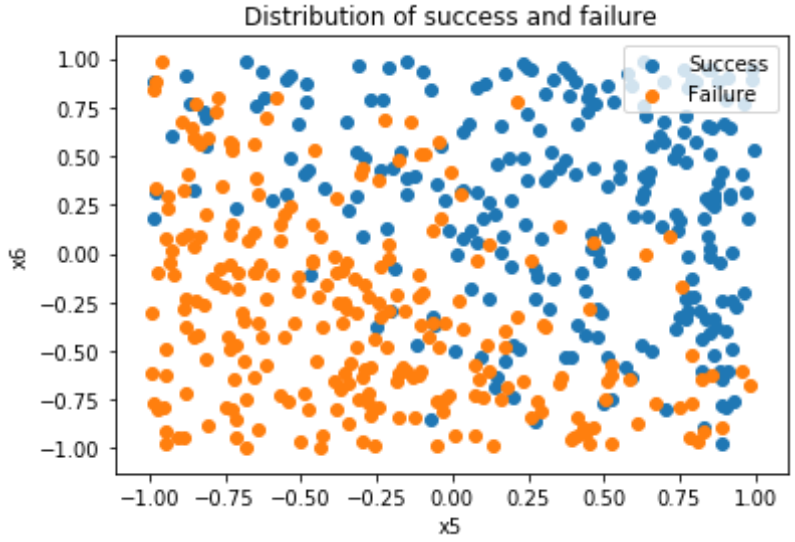
From the accuracy score, we would see that SVM model performs better than logisitic regression in both linear and non-linear models and datasets. We would see that there is no apparent difference in linear model between logistic regression and SVM. However, SVM non-linear model performs much better than linear model, and Logistic Regression non-linear model performs the worst.

Tuning cost

11.

In [85]:
```
x5 = np.random.uniform(-1, 1, 500)
x6 = np.random.uniform(-1, 1, 500)
err3  = np.random.normal(0, 0.5, 500)
y3 = x5 + x6 + err3
success3 = y3 >= 0
failure3 = y3 < 0
```

In [86]:
```
plt.scatter(x5[success3], x6[success3])
plt.scatter(x5[failure3], x6[failure3])
plt.xlabel('x5')
plt.ylabel('x6')
plt.legend(['Success', 'Failure'], loc=1)
plt.title('Distribution of success and failure')
plt.show()
```



12.

In [87]:
```
X4 = np.column_stack((x5, x6))
X4 = pd.DataFrame(X4)
x_train, x_test, y_train, y_test = train_test_split(X4, success3, test_
size=0.2)
```

In [95]:
```
ls_costs = np.arange(1, 11, 1)
cv_errors = []
training_errors = []
for cost in ls_costs:
    svc = SVC(C=cost, kernel='linear')
    cv_error = 1 - np.mean(cross_val_score(svc, x_train, y_train))
    cv_errors.append(cv_error)
    training_error = 1 - svc.fit(x_train, y_train).score(x_train, y_tra
in)
    training_errors.append(training_error)
```

In [99]:
```
df_error = pd.DataFrame({'Cost': ls_costs, 'CV Error': cv_errors, 'Trai
ning Error': training_errors})
df_error.set_index('Cost', inplace=True)
df_error
```

Out[99]:

Cost	CV Error	Training Error
1	0.155014	0.1550
2	0.157521	0.1575
3	0.160027	0.1575
4	0.162533	0.1575
5	0.162514	0.1550
6	0.160008	0.1550
7	0.157502	0.1550

Cost	CV Error	Training Error
8	0.157502	0.1550
9	0.157502	0.1575
10	0.159990	0.1575

From the dataframe, we would see that CV error reaches maximum when cost is between 4 and 5, and it reaches minimum when cost approaches to 1. We would also see that training value reaches minimum at 1 and 5 to 8. However, both CV error and training error fluctuates very small between 0.15 to 0.16.

13.

In [103]:
```
ls_costs = np.arange(1, 11, 1)
test_errors = []
for cost in ls_costs:
    svc = SVC(C=cost, kernel='linear')
    test_error = 1 - svc.fit(x_train, y_train).score(x_test, y_test)
    test_errors.append(test_error)
```

In [104]:
```
df_test_error = pd.DataFrame({'Cost': ls_costs, 'Test Error': test_erro
rs})
df_test_error.set_index('Cost', inplace=True)
df_test_error
```

Out[104]:

Cost	Test Error
1	0.2
2	0.2
3	0.2
4	0.2

Cost	Test Error
5	0.2
6	0.2
7	0.2
8	0.2
9	0.2
10	0.2

As is shown in the above dataframe, the test error is consistent among different cost values.

14.

Our results show that for linear model, test error remains consistent for different cost values. It shows that the error for the model only depends on the training and cv error since test error is always consistent.Based on our results on training and cv error, we would see a small cost (cost < 1) would be a best model since it has the smallest errors. As cost increases, the cv error would first increase then decrease while the training error would first increase then decrease and finally remaining consistent for a certain value.

Application: Predicting attitudes towards the racist college professors

In [109]:
```
df_train = pd.read_csv("https://raw.githubusercontent.com/macss-model20/problem-set-6/master/data/gss_train.csv")
df_test = pd.read_csv("https://raw.githubusercontent.com/macss-model20/problem-set-6/master/data/gss_test.csv")
x_train = df_train.drop(columns="colrac", axis=1)
y_train = df_train["colrac"]
```

In [112]:
```
ls_costs = np.arange(1, 11, 1)
cv_errors = []
for cost in ls_costs:
    svc = SVC(C=cost, kernel='linear')
    cv_error = 1 - np.mean(cross_val_score(svc, x_train, y_train))
    cv_errors.append(cv_error)
```

In [116]:
```
df_cv_error = pd.DataFrame({'Cost': ls_costs, 'CV Error': cv_errors})
df_cv_error.set_index('Cost', inplace=True)
df_cv_error
```

Out[116]:

Cost	CV Error
1	0.215386
2	0.214710
3	0.211332
4	0.214034
5	0.212681
6	0.212681
7	0.213360
8	0.213360
9	0.213357
10	0.212684

The lowest cv error reaches when cost = 3. Generally speaking, the CV error fluctuates very smally between 0.212 and 0.215 as cost changes. As cost increases, the CV error fluctuates between increasing and decreasing with no simple trend or patterns.

16.

In [117]:
```
# radial
ls_costs = np.arange(1, 5, 1)
ls_gammas = ['scale', 'auto']

cv_errors = []
for cost in ls_costs:
    for gamma in ls_gammas:
        svc = SVC(C=cost, kernel='rbf', gamma=gamma)
        cv_error = 1 - np.mean(cross_val_score(svc, x_train, y_train))
        cv_errors.append((cost, gamma, cv_error))
```

In [123]:
```
df_radial_error = pd.DataFrame(cv_errors, columns=['Cost', 'Gamma', 'CV
 Error'])
df_radial_error
```

Out[123]:

	Cost	Gamma	CV Error
0	1	scale	0.255212
1	1	auto	0.284267
2	2	scale	0.245084
3	2	auto	0.282923
4	3	scale	0.234969
5	3	auto	0.279542
6	4	scale	0.228227
7	4	auto	0.280894

As is shown in the dataframe, the CV error for radial SVMs reaches smallest when cost=4 and gamma=scale. Generally speaking, CV error is much smaller when gamma=scale, and it decreases as cost increases.

In [125]:
```
# polynomial
ls_costs = np.arange(1, 5, 1)
ls_degrees = np.arange(1, 5, 1)
```

```
cv_errors = []
for cost in ls_costs:
    for degree in ls_degrees:
        svc = SVC(C=cost, kernel='poly', degree=degree)
        cv_error = 1 - np.mean(cross_val_score(svc, x_train, y_train))
        cv_errors.append((cost, degree, cv_error))
```

In [126]:
```
df_polynomial_error = pd.DataFrame(cv_errors, columns=['Cost', 'Degree'
, 'CV Error'])
df_polynomial_error
```

Out[126]:

	Cost	Degree	CV Error
0	1	1	0.201860
1	1	2	0.230231
2	1	3	0.257903
3	1	4	0.271412
4	2	1	0.208616
5	2	2	0.235632
6	2	3	0.257903
7	2	4	0.271412
8	3	1	0.210642
9	3	2	0.230915
10	3	3	0.257903
11	3	4	0.271412
12	4	1	0.214701
13	4	2	0.233600
14	4	3	0.257903
15	4	4	0.271412

As is shown in the dataframe, the CV error for polynomial SVMs reaches smallest when cost=1 and degree=1. Generally speaking, CV error is much smaller when degree=1, and CV error would increase as degree increases. Also, CV would increase as well as cost increases although the influence of cost on cv error is not so dominant as degree does.