LA_PolynomialClassifier

October 21, 2021

[ ]:
```python
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
```

[ ]:
```python
N = 500

def load_data():
    x_train = np.random.randn(N, 2)
    y_train = (x_train[:, 0]*x_train[:, 1] >= 0.) * 2 - 1
    return x_train, y_train

def least_squares(A, b, reg=1.0):
    return np.linalg.inv((A.T @ A) + reg * np.eye(A.shape[1])) @ (A.T @ b)
```

[ ]:
```python
def confusion_matrix(y_true, y_pred, labels=[]):
    """
    Computes the confusion matrix for a given set of labels.
    Args:
        y_true: The true labels.
        y_pred: The predicted labels.
        labels: The list of labels to consider.
    Returns:
        The confusion matrix. (np.ndarray)
    """
    matrix = np.zeros((len(labels), len(labels)), dtype=int)
    for i in range(len(y_pred)):
        x = labels.index(y_true[i])
        y = labels.index(y_pred[i])
        matrix[x, y] += 1
    return matrix
```

[ ]:
```python
def preprocess_data(x_train):
    a = np.empty((x_train.shape[0],6))
    a[:,0] = 1
    a[:,1:3] = x_train
    a[:,3] = x_train[:,0]*x_train[:,1]
    a[:,4:6] = x_train**2
```

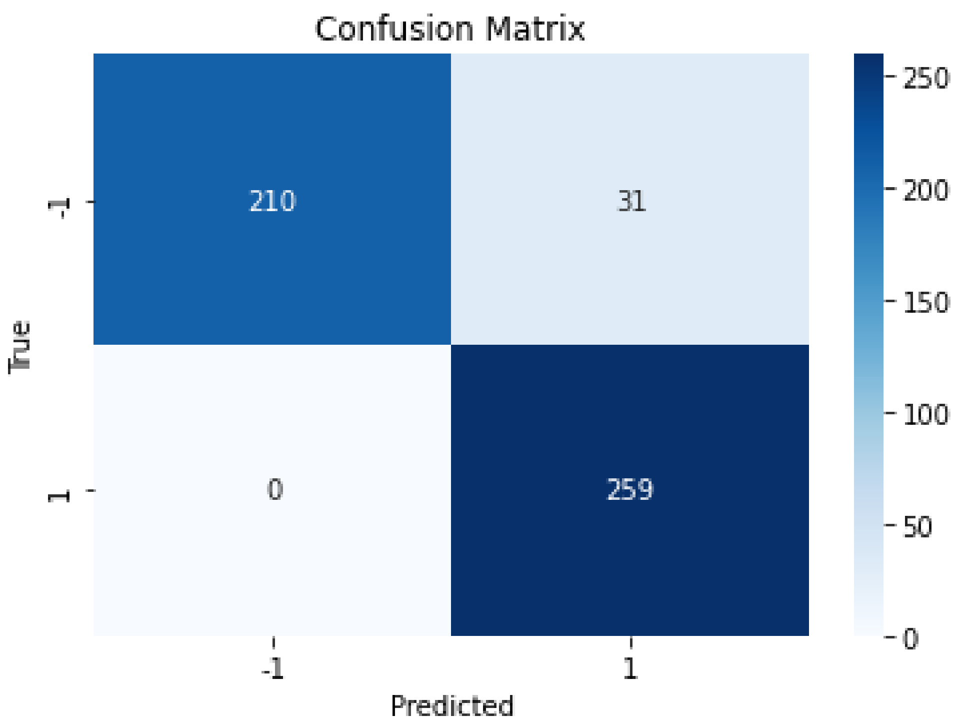
```
    return a
```

[ ]:
```
x_train, y_train = load_data()
A = preprocess_data(x_train)
x_hat = least_squares(A, y_train)
y_pred = A @ x_hat
y_pred = np.sign(y_pred).astype(np.int32)
```

[ ]:
```
err = np.mean(y_pred != y_train)
print(f"Error: {err*100:.2f}%")
```

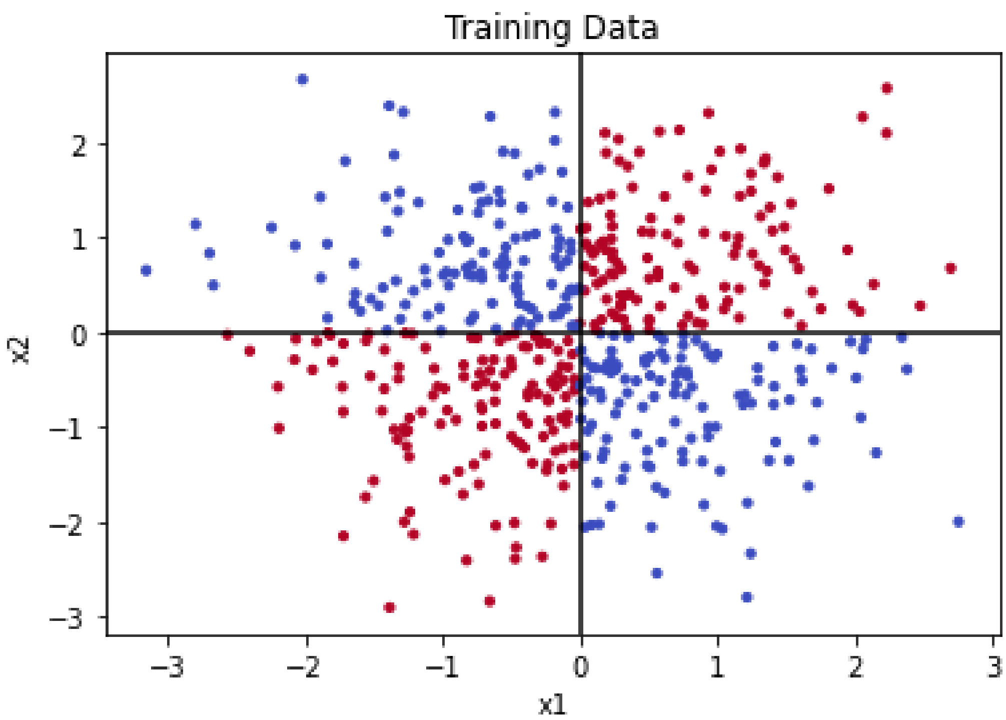
```
Error: 6.20%
```

[ ]:
```
cnf_matrix = confusion_matrix(y_train, y_pred, labels=[-1, 1])
sns.heatmap(cnf_matrix, xticklabels=[
                '-1', '1'], yticklabels=['-1', '1'], annot=True, fmt="d",cmap =␣
 ↪'Blues')
plt.title("Confusion Matrix")
plt.xlabel("Predicted")
plt.ylabel("True")
plt.show()
```



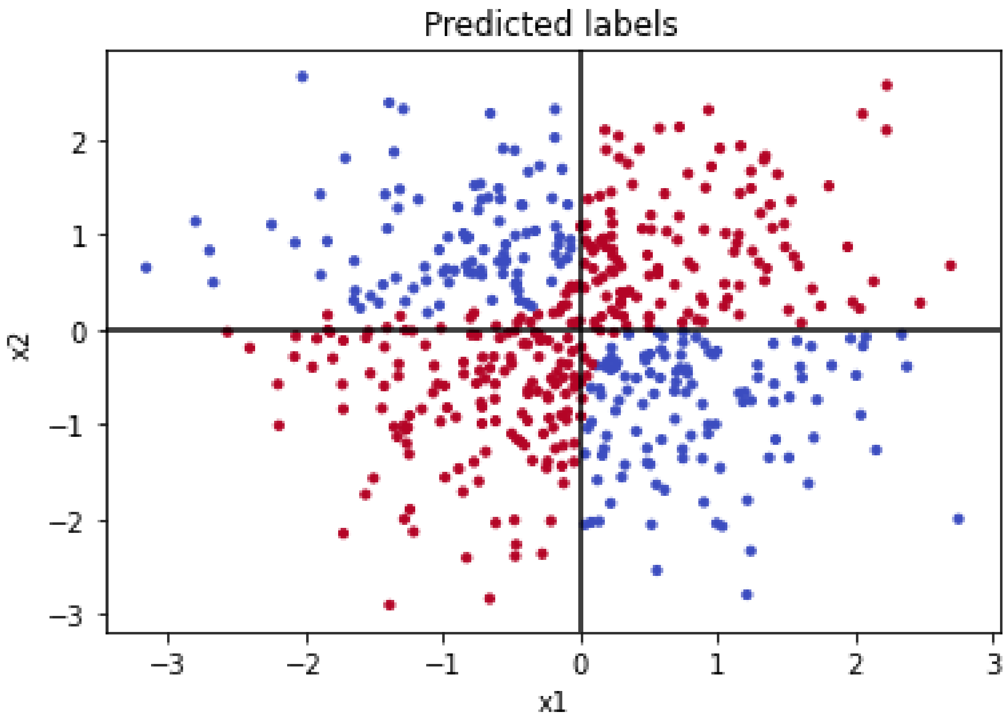
[ ]:
```
fig,ax = plt.subplots()
ax.scatter(x_train[:,0], x_train[:,1], c=y_train, s=10, cmap='coolwarm')
ax.axhline(y=0, color='k')
ax.axvline(x=0, color='k')

ax.set_title("Training Data")
ax.set_xlabel("x1")
ax.set_ylabel("x2")
plt.show()
```



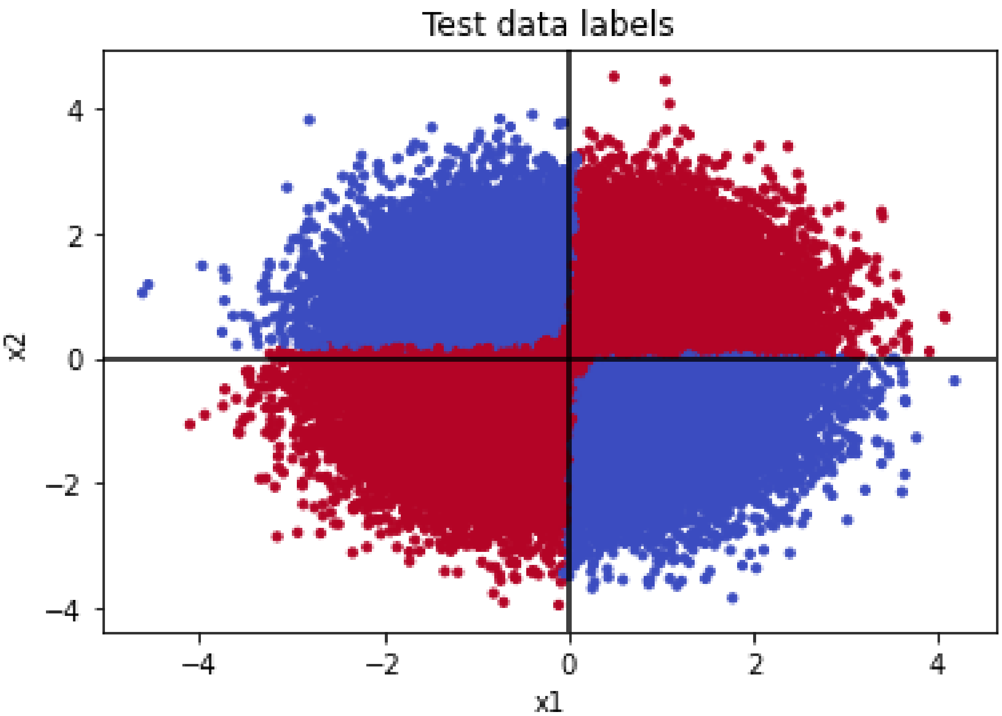
[ ]:
```
fig,ax = plt.subplots()
ax.scatter(x_train[:,0], x_train[:,1], c=y_pred, s=10, cmap='coolwarm')
ax.axhline(y=0, color='k')
ax.axvline(x=0, color='k')

ax.set_title("Predicted labels")
ax.set_xlabel("x1")
ax.set_ylabel("x2")
plt.show()
```




```
[ ]: POINTS = 100000
     x_test = np.random.randn(POINTS, 2)
     y_test = (x_test[:, 0]*x_test[:, 1] >= 0.) * 2 - 1
     A_test = preprocess_data(x_test)
     y_hat = A_test @ x_hat
     y_hat = np.sign(y_hat).astype(np.int32)
```

```
[ ]: fig,ax = plt.subplots()
     ax.scatter(x_test[:,0], x_test[:,1], c=y_hat, s=10, cmap='coolwarm')
     # ax.grid(True,which = 'both')
     ax.axhline(y=0, color='k')
     ax.axvline(x=0, color='k')
     ax.set_title("Test data labels")
     ax.set_xlabel("x1")
     ax.set_ylabel("x2")
     plt.show()
```



[ ]:
```
x_hat
```

[ ]:
```
array([ 0.02725639, -0.0662813 , -0.01267758,  0.58076163,  0.01154289,
       -0.01519336])
```

1. Yes, the second degree-polynomial, $g = x_1 * x_2$ classifies the generated points with zero error.
2. If $g(x_i) > 0. \implies y_i = 1$ else $y_i = -1$.
3. Looking at the least-squared solution produced ($\hat{x}$), the coefficient for $x_1 x_2$ is the largest although other coefficients are not zero

[ ]: