Lab Assignment solution key

1.

```
import numpy as np

# Register number digits: e.g., a=7, b=9
a, b = 7, 9
c1 = float(input("Enter c1: "))
c2 = float(input("Enter c2: "))

dist = abs(c1 - c2) / np.sqrt(a**2 + b**2)
print(f"Distance between parallel lines: {dist:.4f}")
```

2.

```
import numpy as np


def is_orthonormal(vectors):
    V = np.column_stack(vectors)
    n = V.shape[1]
    # Check if V^T * V is identity matrix
    identity_check = np.dot(V.T, V)
    return np.allclose(identity_check, np.eye(n))


v1 = np.array([1, 0, 0])
v2 = np.array([0, 1, 0])
print("Is Orthonormal?", is_orthonormal([v1, v2]))
```

3.

```
import numpy as np

v = np.array([1, 2, 3])
w = np.array([4, 6, 8])

dist = np.linalg.norm(v - w)
print("Euclidean Distance:", dist)
```

4.

```
from scipy.sparse import csr_matrix

A = csr_matrix([[1, 0, 0], [0, 2, 0]])
B = csr_matrix([[0, 3], [4, 0], [0, 0]])
```

```
C = A.dot(B)
print("Sparse Product:\n", C.toarray())
```

5.

```
import numpy as np

X = np.array([1, 2, 3, 4, 5])
y = np.array([2, 4, 5, 4, 5])

exp_X, exp_Y = np.mean(X), np.mean(y)
var_X, var_Y = np.var(X), np.var(y)
cov_XY = np.cov(X, y)[0, 1]
corr_XY = np.corrcoef(X, y)[0, 1]

print(f"Covariance: {cov_XY:.4f}, Correlation: {corr_XY:.4f}")
if cov_XY == 0:
    print("No linear relationship between the two features")
```

6.

```
import numpy as np

# D features (columns), N data points (rows)
X = np.array([[1, 2, 3], [4, 5, 6], [7, 8, 10]])

cov_matrix = np.cov(X, rowvar=False)
print("Covariance Matrix:\n", cov_matrix)
```

7.

```
import sympy as sp

x, y, lam = sp.symbols("x y lambda")
f = x**2 + y**2
g = x + y - 1

# Lagrangian
L = f - lam * g

# System of equations: grad(L) = 0
eq1 = sp.diff(L, x)
eq2 = sp.diff(L, y)
eq3 = sp.diff(L, lam)

sol = sp.solve([eq1, eq2, eq3], (x, y, lam))
print("Optimal Solution (x, y, lambda):", sol)
```

8.

```
import numpy as np

# Coin tosses
tosses = np.array([1, 0, 1, 1, 1, 0, 1, 0, 1, 1])

p_mle = np.mean(tosses)
print("MLE estimate for p:", p_mle)
```

9.

```
import numpy as np

data = [1.2, 0.9, 1.5, 2.0, 1.7]

mu_mle = np.mean(data)
sigma2_mle = np.var(data)  # Biased sample variance = MLE

print(f"MLE Mean (mu): {mu_mle:.4f}")
print(f"MLE Variance (sigma^2): {sigma2_mle:.4f}")
```

10.

```
import numpy as np

A = np.array([[1, 2], [3, 4], [5, 6]])

U, S, VT = np.linalg.svd(A)

print("U:\n", U)
print("Singular Values (S):", S)
print("VT:\n", VT)
```

11.

```
import matplotlib.pyplot as plt
from sklearn.decomposition import PCA
from sklearn.datasets import load_iris

iris = load_iris()
X, y = iris.data, iris.target

pca = PCA(n_components=2)
X_pca = pca.fit_transform(X)

plt.scatter(X_pca[:, 0], X_pca[:, 1], c=y, cmap="viridis")
```

```
plt.xlabel("PC 1")
plt.ylabel("PC 2")
plt.title("PCA on Iris")
plt.show()
```

12.

```
import numpy as np
from sklearn.linear_model import LogisticRegression

X = np.array([[34, 78], [45, 85], [50, 43], [65, 72], [70, 90]])
y = np.array([0, 0, 1, 1, 1])

clf = LogisticRegression()
clf.fit(X, y)

print("Coefficients:", clf.coef_)
print("Intercept:", clf.intercept_)
```

13.

```
import numpy as np

# Synthetic Dataset
X = np.array([[1], [2], [3], [4], [5]], dtype=float)
y = np.array([2, 3, 5, 6, 8], dtype=float)

X_b = np.c_[np.ones((len(X), 1)), X]

# 1. Linear Regression
beta_lr = np.linalg.inv(X_b.T @ X_b) @ X_b.T @ y

# 2. Ridge Regression (L2)
alpha_ridge = 1.0
I = np.eye(X_b.shape[1])
I[0, 0] = 0  # Do not penalize bias
beta_ridge = np.linalg.inv(X_b.T @ X_b + alpha_ridge * I) @ X_b.T @ y

# 3. Lasso (Coordinate Descent)
beta_lasso = np.zeros(X_b.shape[1])
alpha_lasso = 0.5
lr_rate = 0.01

for _ in range(1000):
    grad = -X_b.T @ (y - X_b @ beta_lasso)
    beta_lasso -= lr_rate * (grad + alpha_lasso * np.sign(beta_lasso))
```

```
print("Linear Coeffs:", beta_lr)
print("Ridge Coeffs: ", beta_ridge)
print("Lasso Coeffs: ", beta_lasso)
```

14.

```
import matplotlib.pyplot as plt
import numpy as np
from sklearn.datasets import load_iris

iris = load_iris()
X, y = iris.data, iris.target

mean_overall = np.mean(X, axis=0)
SW = np.zeros((4, 4))
SB = np.zeros((4, 4))

for c in np.unique(y):
    X_c = X[y == c]
    mean_c = np.mean(X_c, axis=0)
    SW += (X_c - mean_c).T @ (X_c - mean_c)
    n_c = X_c.shape[0]
    mean_diff = (mean_c - mean_overall).reshape(-1, 1)
    SB += n_c * (mean_diff @ mean_diff.T)

# Eigenvalue decomposition of inv(SW) * SB
eigvals, eigvecs = np.linalg.eig(np.linalg.inv(SW) @ SB)
idx = np.argsort(eigvals)[::-1]
W = eigvecs[:, idx[:2]]

X_lda = X @ W

plt.scatter(X_lda[:, 0], X_lda[:, 1], c=y, cmap="rainbow")
plt.title("Manual LDA Projection")
plt.show()
```

15.

```
import matplotlib.pyplot as plt
import numpy as np
from sklearn.svm import SVC

X = np.array(
    [
        [10, 5],
```

```
        [1, 1],
        [5, 3],
        [3, 2],
        [15, 7],
        [2, 1],
        [7, 4],
        [9, 6],
        [20, 10],
        [0, 0],
    ]
)
y = np.array([1, 0, 0, 0, 1, 0, 0, 1, 1, 0])

model = SVC(kernel="linear")
model.fit(X, y)

print("Accuracy:", model.score(X, y))

# Decision boundary plot
plt.scatter(X[:, 0], X[:, 1], c=y, cmap="bwr")
w = model.coef_[0]
b = model.intercept_[0]
x_line = np.linspace(0, 20, 100)
y_line = -(w[0] * x_line + b) / w[1]

plt.plot(x_line, y_line, color="black", linestyle="--")
plt.xlabel("Buy Words")
plt.ylabel("Hyperlinks")
plt.show()
```