fisher_lda_from_scratch

September 12, 2025

1 Fisher's Linear Discriminant (LDA) — from scratch

Implement Fisher's criterion and LDA step-by-step. We'll compute projection vector w that maximizes class separation, visualize projections and classification, and discuss common pitfalls.

1.1 What this notebook contains

- Derivation in code: within-class scatter, between-class scatter
- Fit/predict functions implemented in numpy
- Examples on synthetic data and (optionally) Iris dataset
- FAQs and practical notes

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

np.random.seed(0)
```

1.2 Implementation notes

Fisher's LDA for two classes finds direction w that maximizes (w^T S_B w) / (w^T S_W w) where S_B is between-class scatter and S_W is within-class scatter. For two classes w is proportional to S_W^{-1}(m1 - m2). We'll implement the two-class case from scratch.

[ ]:
```
def fisher_lda_fit(X, y, regularize=1e-6):
    """Fit two-class Fisher LDA.
    X: (n_samples, d)
    y: labels (n_samples,) with exactly two distinct classes
    returns:
      w: projection vector (d,)
      threshold: scalar on projected axis (midpoint of class projected means)
      stats: dict with class means and scatter matrices
    regularize: small diagonal addition to S_W for numerical stability
    """
    X = np.asarray(X)
    y = np.asarray(y)
    classes = np.unique(y)
    if classes.shape[0] != 2:
        raise ValueError('This simple implementation handles exactly two␣
↪classes.')
```

```
    c0, c1 = classes
    X0 = X[y == c0]
    X1 = X[y == c1]
    m0 = X0.mean(axis=0)
    m1 = X1.mean(axis=0)
    # within-class scatter
    S_w = np.cov(X0, rowvar=False, bias=True) * X0.shape[0] + np.cov(X1,␣
 ↪rowvar=False, bias=True) * X1.shape[0]
    # regularize
    S_w += regularize * np.eye(S_w.shape[0])
    # direction
    w = np.linalg.solve(S_w, (m1 - m0))
    # scale doesn't matter for direction, but keep it normalized for convenience
    w = w / np.linalg.norm(w)
    # projections and threshold (midpoint between projected means)
    proj0 = X0.dot(w)
    proj1 = X1.dot(w)
    mean0 = proj0.mean(); mean1 = proj1.mean()
    threshold = 0.5 * (mean0 + mean1)
    stats = {'m0': m0, 'm1': m1, 'S_w': S_w, 'proj_mean0': mean0, 'proj_mean1':␣
 ↪mean1}
    return w, threshold, stats

def fisher_predict(X, w, threshold):
    proj = X.dot(w)
    # predict class 1 if projection > threshold
    return (proj > threshold).astype(int), proj
```

1.2.1 Synthetic demo (two Gaussian clouds)

Create two 2D Gaussian clusters, fit LDA, visualize the separating direction and projected histograms.

[ ]:
```
# synthetic data
np.random.seed(10)
n = 200
X0 = np.random.multivariate_normal([-1.5, 0], [[0.6, 0.2],[0.2, 0.6]], size=n//
 ↪2)
X1 = np.random.multivariate_normal([1.0, 0.5], [[0.5, -0.15],[-0.15, 0.5]],␣
 ↪size=n//2)
X = np.vstack([X0, X1])
y = np.array([0]*(n//2) + [1]*(n//2))

w, thr, stats = fisher_lda_fit(X, y)
preds, proj = fisher_predict(X, w, thr)

# plot data and line for direction w
```

```python
plt.figure(figsize=(8,4))
plt.subplot(1,2,1)
plt.scatter(X0[:,0], X0[:,1], label='class 0', alpha=0.6)
plt.scatter(X1[:,0], X1[:,1], label='class 1', alpha=0.6)
# plot direction through overall mean
mean_all = X.mean(axis=0)
line_x = np.linspace(mean_all[0]-4, mean_all[0]+4, 50)
line_y = mean_all[1] + (w[1]/w[0])*(line_x - mean_all[0])
plt.plot(line_x, line_y, color='k', linestyle='--', label='Fisher direction')
plt.legend()
plt.title('Data and Fisher direction')

# projections histogram
plt.subplot(1,2,2)
plt.hist(proj[y==0], bins=25, alpha=0.6, label='class 0')
plt.hist(proj[y==1], bins=25, alpha=0.6, label='class 1')
plt.axvline(thr, color='k', linestyle='--', label='threshold')
plt.legend()
plt.title('Projections on w')
plt.tight_layout()
plt.show()

# accuracy
acc = (preds == y).mean()
print(f'Classification accuracy (simple threshold on projection): {acc:.3f}')
```

1.2.2 Optional: Iris dataset (multi-class)

Fisher's LDA generalizes to multiple classes; one common approach is to compute projection vectors from generalized eigenvalue problem S_B v = lambda S_W v. In this notebook we implemented two-class explicitly. Below we provide a short sketch code: try extending to multiple classes as an exercise.

[ ]:
```python
# quick attempt at using sklearn's iris if available (optional)
try:
    from sklearn import datasets
    iris = datasets.load_iris()
    X_iris = iris.data[:, :2]  # take first two features for easy plotting
    y_iris = iris.target
    print('Iris loaded: demo left as exercise - this notebook focuses on␣
 ↪two-class LDA.')
except Exception as e:
    print('sklearn not available in this environment or failed to import:', e)
```

1.3 Common questions (FAQ)

Q: Why regularize S_W? A: To avoid singular matrices when classes have fewer samples than dimensions or data is nearly collinear. Add a small diagonal term.

Q: How does LDA differ from PCA? A: PCA is unsupervised and seeks directions of maximum variance. LDA is supervised and seeks directions that maximize class separability.

Q: Multi-class LDA? A: Solve generalized eigenproblem S_B v = lambda S_W v and pick top eigenvectors; dimensionality limited to (C-1) where C is number of classes.

1.4 Exercises / next steps

- Implement the multi-class generalized eigenvalue LDA
- Add cross-validation for regularization strength
- Compare Fisher LDA classification to logistic regression on the same dataset