Tyler Singleton
February 2, 2025

Problem 1:

First, let the response vector y be generated according to a Gaussian sampling model with mean $X\beta$ and covariance matrix $\sigma^2 I$, such that:

$$p(y \mid \beta) \propto \exp\Big(-\frac{1}{2\sigma^2} \|y - X\beta\|^2\Big)$$

If we impose a Gaussian prior on the coefficients β with mean 0 and covariance τI, we have:

$$p(\beta) \propto \exp\Big(-\frac{1}{2\tau} \|\beta\|^2\Big)$$

By Bayes' rule, the posterior distribution for β is proportional to the product of the likelihood and the prior, so we can write:

$$p(\beta \mid y) \propto p(y \mid \beta)\, p(\beta) \propto \exp\Big(-\frac{1}{2\sigma^2} \|y - X\beta\|^2 - \frac{1}{2\tau} \|\beta\|^2\Big)$$

Since the exponential function is monotonic, maximizing the posterior probability is equivalent to minimizing the exponent:

$$\frac{1}{2\sigma^2} \|y - X\beta\|^2 + \frac{1}{2\tau} \|\beta\|^2$$

Multiplying the entire expression by the positive constant $2\sigma^2$ does not affect the location of the minimum, so:

$$\|y - X\beta\|^2 + \frac{\sigma^2}{\tau} \|\beta\|^2$$

By definition, the ridge regression estimator is the minimizer of the penalized least-squares problem:

$$\min_{\beta} \|y - X\beta\|^2 + \lambda \|\beta\|^2$$

Clearly, we see that the ridge estimator coincides with the MAP estimate when the regularization parameter satisfies:

$$\lambda = \frac{\sigma^2}{\tau}$$

Because the posterior distribution is Gaussian, its mean, mode, and median all coincide. So, the ridge regression estimator is not only the MAP estimate but also the mean of the posterior distribution.

Problem 2:

To show that the ridge regression estimates can be obtained by ordinary least squares regression on an augmented data set, we first begin with the ridge regression problem, which seeks the coefficient vector β that minimizes:

$$\|y - X\beta\|^2 + \lambda\|\beta\|^2$$

Differentiating with respect to β and setting the derivative to zero yields:

$$(X^\top X + \lambda I)\beta = X^\top y$$

Which can be rewritten:

$$\beta_{\text{ridge}} = (X^\top X + \lambda I)^{-1} X^\top y$$

Now, consider forming an augmented data set. Define the augmented design matrix and response vector by:

$$X_{\text{aug}} = \begin{bmatrix} X \\ \sqrt{\lambda}\, I \end{bmatrix} \quad \text{and} \quad y_{\text{aug}} = \begin{bmatrix} y \\ 0 \end{bmatrix}$$

Ordinary least squares regression on this augmented data involves minimizing:

$$\|y_{\text{aug}} - X_{\text{aug}}\beta\|^2$$

The normal equations for this problem are:

$$X_{\text{aug}}^\top X_{\text{aug}} \beta = X_{\text{aug}}^\top y_{\text{aug}}$$

Since:

$$X_{\text{aug}}^\top X_{\text{aug}} = X^\top X + (\sqrt{\lambda}\, I)^\top (\sqrt{\lambda}\, I) = X^\top X + \lambda I$$

And:

$$X_{\text{aug}}^\top y_{\text{aug}} = X^\top y + (\sqrt{\lambda}\, I)^\top 0 = X^\top y$$

Then, the normal equations are:

$$(X^\top X + \lambda I)\beta = X^\top y$$

So, the least squares solution on the augmented data set is identical to the equation obtained for ridge regression:

$$\beta_{\text{aug}} = (X^\top X + \lambda I)^{-1} X^\top y = \beta_{\text{ridge}}$$

This confirms that by augmenting the data with p artificial observations (each with a response of zero and a design matrix contribution of $\sqrt{\lambda}\, I$), ordinary least squares yields the same estimates as ridge regression, effectively shrinking the coefficients toward zero.

Problem 3:

To show that the elastic-net optimization problem can be transformed into a lasso problem, we begin with the following objective:

$$\min_{\beta} \|y - X\beta\|^2 + \lambda\big[\alpha\|\beta\|_2^2 + (1-\alpha)\|\beta\|_1\big]$$

The squared error term $\|y - X\beta\|^2$ together with the ℓ_2 penalty $\lambda\alpha\|\beta\|_2^2$ can be merged into a single least squares term by augmenting both the design matrix X and the response vector y. Specifically, we can define:

$$\tilde{X} \;=\; \begin{pmatrix} X \\ \sqrt{\lambda\alpha}\, I \end{pmatrix} \quad \text{and} \quad \tilde{y} \;=\; \begin{pmatrix} y \\ 0 \end{pmatrix}$$

where I is the $p \times p$ identity matrix (with p denoting the number of predictors), and 0 is the p-dimensional zero vector. So, that this construction yields:

$$\|\tilde{y} - \tilde{X}\beta\|^2 \;=\; \|y - X\beta\|^2 + \|\sqrt{\lambda\alpha}\,\beta\|^2 \;=\; \|y - X\beta\|^2 + \lambda\alpha\,\|\beta\|_2^2$$

Substituting this expression into the elastic-net objective give:

$$\min_{\beta} \|\tilde{y} - \tilde{X}\beta\|^2 \ + \ \lambda(1-\alpha)\|\beta\|_1$$

Then, this is now in the standard lasso form:

$$\min_{\beta} \|\tilde{y} - \tilde{X}\beta\|^2 \ + \ \tilde{\lambda}\,\|\beta\|_1, \quad \text{with} \quad \tilde{\lambda} = \lambda(1-\alpha)$$

So, by suitably augmenting the design matrix X and the response vector y, the original elastic-net optimization problem can be recast as a lasso problem.

Problem 4:

To show that Linear Discriminant Analysis (LDA) using $\hat{Y}$ is identical to LDA in the original space, we begin by defining the linear transformation of the predictors. Given an indicator response matrix Y and predictors X, we form the fitted values via linear regression as:

$$\hat{Y} \ = \ X(X^TX)^{-1}X^TY \ = \ XB$$

We can set:

$$B \ = \ (X^TX)^{-1}X^TY$$

For any input $x \in \mathbb{R}^p$, this transformation yields:

$$\hat{y} \ = \ B^Tx \ \in \ \mathbb{R}^K$$

Under the standard LDA model in the original space, we assume observations from class k follow a multivariate normal distribution with a common within-class covariance Σ and class-specific mean μ_k. The discriminant function for class k is:

$$\delta_k(x) \ = \ x^T\Sigma^{-1}\mu_k \ - \ \tfrac{1}{2}\,\mu_k^T\Sigma^{-1}\mu_k \ + \ \log\pi_k$$

Where π_k is the prior probability of class k. An observation is assigned to the class with the highest value of $\delta_k(x)$. Next, consider LDA applied to the transformed predictors. The class means transform linearly:

$$\hat{\mu}_k \ = \ B^T\mu_k$$

The class covariance in the transformed space is:

$$\hat{\Sigma} \ = \ B^T\Sigma B$$

So, the discriminant function in the transformed space is:

$$\hat{\delta}_k(\hat{y}) \ = \ \hat{y}^T\hat{\Sigma}^{-1}\hat{\mu}_k \ - \ \tfrac{1}{2}\,\hat{\mu}_k^T\hat{\Sigma}^{-1}\hat{\mu}_k \ + \ \log\pi_k$$

Substitute $\hat{y} = B^Tx$ and $\hat{\mu}_k = B^T\mu_k$:

$$\hat{\delta}_k(B^Tx) \ = \ (B^Tx)^T\big(B^T\Sigma B\big)^{-1}(B^T\mu_k) \ - \ \tfrac{1}{2}\,(B^T\mu_k)^T\big(B^T\Sigma B\big)^{-1}(B^T\mu_k) \ + \ \log\pi_k$$

Note that:

$$(B^Tx)^T\big(B^T\Sigma B\big)^{-1}(B^T\mu_k) \ = \ x^TB\big(B^T\Sigma B\big)^{-1}B^T\mu_k$$

Likewise, similar algebra handles the μ_k-dependent quadratic terms.

$$P \;=\; B\big(B^T\Sigma B\big)^{-1}B^T$$

Clearly, for x in the column space of X:

$$x^T\Sigma^{-1}\mu_k \;=\; x^T P\,\mu_k \;=\; x^T B\big(B^T\Sigma B\big)^{-1}B^T\mu_k$$

And:

$$\mu_k^T\Sigma^{-1}\mu_k \;=\; \mu_k^T P\,\mu_k \;=\; \mu_k^T B\big(B^T\Sigma B\big)^{-1}B^T\mu_k$$

So, we have:

$$\hat{\delta}_k(B^T x) \;=\; x^T\Sigma^{-1}\mu_k \;-\; \tfrac{1}{2}\,\mu_k^T\Sigma^{-1}\mu_k \;+\; \log\pi_k$$

This coincides with the original discriminant function $\delta_k(x)$. So, LDA on the transformed predictors $\hat{y} = B^T x$ is identical to LDA in the original space.

Problem 6:

Python code to train a Quadratic Discriminant Analysis model using SciKit Learn on this dataset is provided below. This code yields the following outputs:

Test Accuracy: 0.4719

Misclassification Error: 0.5281

```
import numpy as np
import pandas as pd
from sklearn.discriminant_analysis import QuadraticDiscriminantAnalysis
from sklearn.metrics import accuracy_score

# load the data and set first column as the index
train_df = pd.read_csv("train.csv", index_col=0)
test_df = pd.read_csv("test.csv", index_col=0)

# for debugging
print("Train columns:", train_df.columns.tolist())
print("Test columns:", test_df.columns.tolist())

# target = "y" column, feature columns = start with "x"
feature_columns = [col for col in train_df.columns if col.startswith("x.")]
print("Feature columns:", feature_columns)

# set up the train / test data
X_train = train_df[feature_columns].values
y_train = train_df["y"].values
X_test = test_df[feature_columns].values
y_test = test_df["y"].values

# define and train a QDA model, then predict
qda = QuadraticDiscriminantAnalysis()
qda.fit(X_train, y_train)
y_pred = qda.predict(X_test)
```

```
# find the accuracy and error
accuracy = accuracy_score(y_test, y_pred)
misclassification_error = 1 - accuracy
print(f"Test Accuracy: {accuracy:.4f}")
print(f"Misclassification Error: {misclassification_error:.4f}")
```