Fisher Linear Discriminant

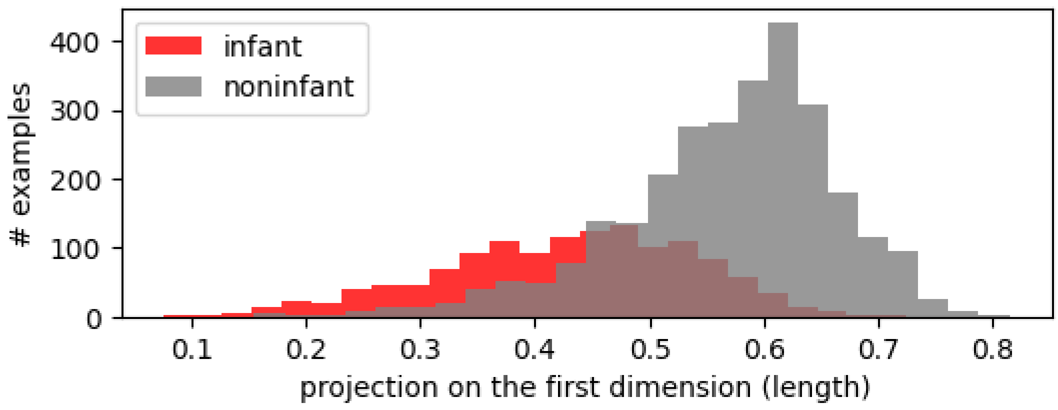
In this exercise, we apply Fisher Linear Discriminant as described in Chapter 3.8.2 of Duda et al. on the UCI Abalone dataset. A description of the dataset is given at the page https://archive.ics.uci.edu/ml/datasets/Abalone. The following two methods are provided for your convenience:

- `utils.Abalone.__init__(self)` reads the Abalone data and instantiates two data matrices corresponding to: *infant (I)*, *non-infant (N)*.
- `utils.Abalone.plot(self,w)` produces a histogram of the data when projected onto a vector `w`, and where each class is shown in a different color.

Sample code that makes use of these two methods is given below. It loads the data, looks at the shape of instantiated matrices, and plots the projection on the first dimension of the data representing the length of the abalone.

In [ ]:
```
%matplotlib inline
import utils,numpy

# Load the data
abalone = utils.Abalone()

# Print dataset size for each class
print(abalone.I.shape, abalone.N.shape)

# Project data on the first dimension
w = numpy.array([1,0,0,0,0,0,0])
abalone.plot(w,'projection on the first dimension (length)')
```

```
(1342, 7) (2835, 7)
```



Implementation (10 + 5 + 5 = 20 P)

- **Create a function `w = fisher(X1,X2)` that takes as input the data for two classes and returns the Fisher linear discriminant.**

- **Create a function `objective(X1,X2,w)` that evaluates the objective defined in Equation 96 of Duda et al. for an arbitrary projection vector `w`.**

- **Create a function `z = phi(X)` that returns a quadratic expansion for each data point `x` in the dataset. Such expansion consists of the vector `x` itself, to which we concatenate the vector of all pairwise products between elements of `x`.** In other words, letting $x = (x_1, \dots, x_d)$ denote the d-dimensional data point, the quadratic expansion for this data point is a $d \cdot (d+3)/2$ dimensional vector given by $\phi(x) = (x_i)_{1 \le i \le d} \cup (x_i x_j)_{1 \le i \le j \le d}$. For example, the quadratic expansion for $d = 2$ is $(x_1, x_2, x_1^2, x_2^2, x_1 x_2)$.

In [ ]:
```python
def fisher(X1,X2):
    """

    """
    ##### Replace by your code
    # calculate m1,m2
    m1 = numpy.mean(X1, axis=0).reshape(-1, 1)
    m2 = numpy.mean(X2, axis=0).reshape(-1, 1)
    # 计算类间散布矩阵 Sb
    mean_diff = m1 - m2

   # coveriance matrix
    S1 = numpy.cov(X1, rowvar=False)
    S2 = numpy.cov(X2, rowvar=False)

    # whin class scatter matrix
    Sw = S1 + S2

    # Sw is invertible
    try:
        Sw_inv = numpy.linalg.inv(Sw)
    except numpy.linalg.LinAlgError:
        Sw_inv = numpy.linalg.pinv(Sw)

    #  w = Sw^{-1} (m1 - m2), get the projection vector
    w = Sw_inv @ mean_diff

    return w
    #####


def objective(X1,X2,w):
    m1 = numpy.mean(X1, axis=0).reshape(-1, 1)
    m2 = numpy.mean(X2, axis=0).reshape(-1, 1)
    # print(f'm1 shape = {m1.shape}')
    # print(f'm2 shape = {m2.shape}')

    w = w.reshape(-1, 1)
    # print(f'w shape = {w.shape}')
    m1_proj = numpy.dot(w.T, m1)
    m2_proj = numpy.dot(w.T, m2)

    # Compute the scatter (variance) of each projected class
    s1_sq = numpy.sum((X1 @ w - m1_proj) ** 2)
    s2_sq = numpy.sum((X2 @ w - m2_proj) ** 2)
    # Compute the objective function
    J = (m1_proj - m2_proj) ** 2 / (s1_sq + s2_sq)

    # keep only the 5th decimal digit
    return numpy.round(J[0, 0], 5)
    ####

def expand(X):
    ##### Replace by your code
    n_samples, d = X.shape # 7
    num_quad = d * (d + 1) // 2
    Z = numpy.zeros((n_samples, d + num_quad))
    Z[:, :d] = X
    idx = d
    for i in range(d):
        for j in range(i, d):
            Z[:, idx] = X[:, i] * X[:, j]
            idx += 1
    return Z
    #####
```

Analysis (5 + 5 = 10 P)

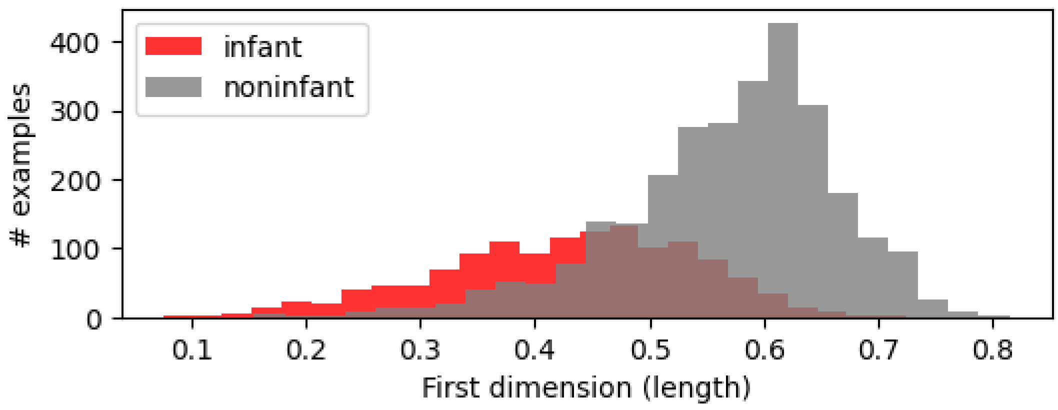
- **Print value of the objective function and the histogram for several values of `w`:**
    - `w` is a canonical coordinate vector for the first feature (length).
    - `w` is the difference between the mean vectors of the two classes.
    - `w` is the Fisher linear discriminant.
    - `w` is the Fisher linear discriminant (after quadratic expansion of the data).

In [ ]:

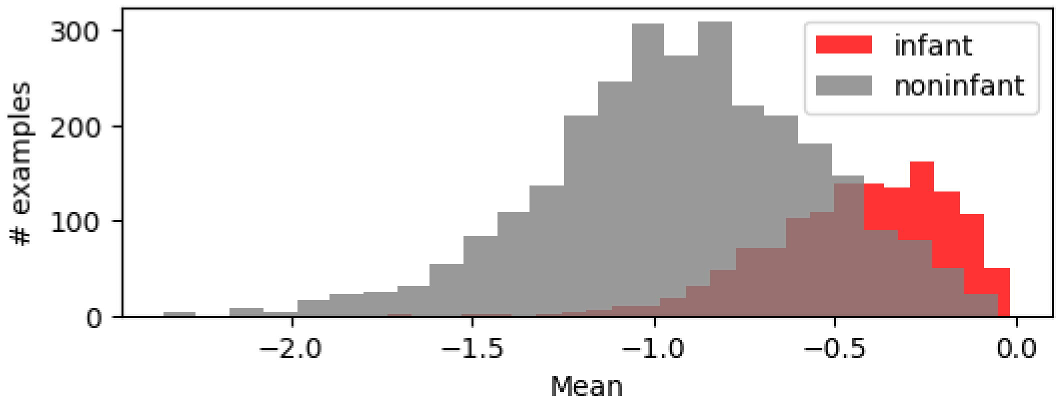
```
##### REPLACE BY YOUR CODE

# Project data on the first dimension
w = numpy.array([1,0,0,0,0,0,0])
print('First dimension (length):', objective(abalone.I, abalone.N, w))
abalone.plot(w,'First dimension (length)')

# Mean
m1 = numpy.mean(abalone.I, axis=0).reshape(-1, 1)
m2 = numpy.mean(abalone.N, axis=0).reshape(-1, 1)
w = m1-m2
print('Means Linear:', objective(abalone.I, abalone.N, w))
abalone.plot(w,'Mean')


w = fisher(abalone.I, abalone.N)
print('Fisher:', objective(abalone.I, abalone.N, w))
abalone.plot(w,'Fisher')


# Expand the data
I = expand(abalone.I)
N = expand(abalone.N)
w = fisher(I, N)
print('Fisher after expansion:', objective(I, N, w))
```

```
abalone.N = N
abalone.I = I
abalone.plot(w,'Fisher after expansion')
#####

# scale the data
```

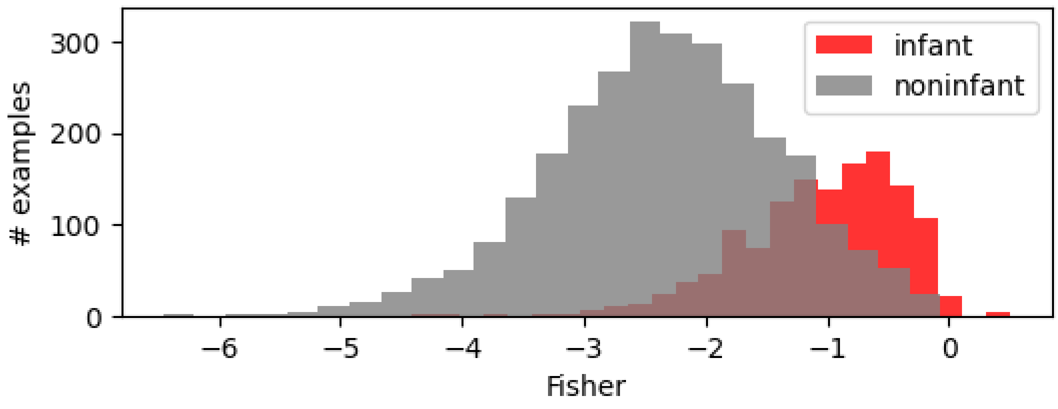
```
First dimension (length): 0.00048
```



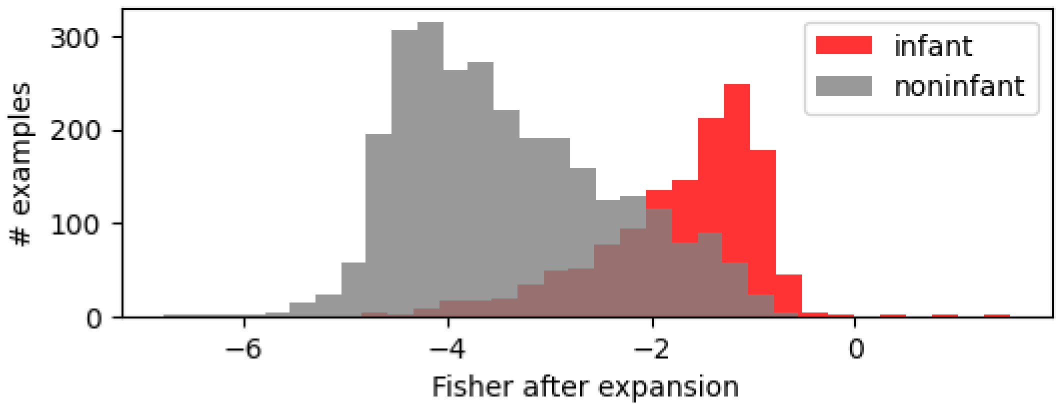
```
Means Linear: 0.0005
```



```
Fisher: 0.00055
```



```
Fisher after expansion: 0.00074
```



300
200
100
0
# examples
−6
−4
−2
0
Fisher after expansion
infant
noninfant