Project 2: Fisher's Linear Discriminant Analysis

1 Fisher's LDA via a toy problem

1.1 Introduction

The aim of the Fisher's Linear Discriminant Analysis is to find a separation vector v to classify data points. This vector can be found by solving the maximization problem:

$$\max \frac{(v^T m_A - v^T m_B)}{v^T(\Sigma_A + \Sigma_B)v} \tag{1}$$

where Σ_C is the covariance matrix and m_C are the sampled means for $C \in \{A, B\}$. We can rewrite (1) as

$$R = \frac{v^T S v}{v^T M v} \tag{2}$$

where $S = (m_A - m_B)(m_A - m_B)^T$ and $M = \Sigma_A + \Sigma_B$. From the exercises we know that the vector v maximizes (2) if and only if $Sv = \lambda M v$ is satisfied. Since S has rank one, we know Sv has the direction $m_A - m_B$. This implies Mv has to be in the direction to satisfy $Sv = \lambda M v$.
Together we can conclude $v = M^{-1}(m_A - m_B) = (\Sigma_A + \Sigma_B)^{-1}(m_A - m_B)$ maximizes (1). The separation vector v will then be used to classify data points x based on its projection. We classify x to class A, if $v^T x > c$ for a constant c, and to class B, if $v^T x \leq c$.

1.2 Building a classifier for the toy problem

For the given toy problem in the task sheet, we aim to build a classifier. First, for given datasets *DataA* and *DataB* we split the data points into training and testing data for each dataset, where each column represents a sample.
We will now create the mean value vector m_C and the covariance matrix Σ_C for $C \in \{A, B\}$ to create the separation vector v.

```
% sample mean & covariance
mA = mean(TrainA')';
mB = mean(TrainB')';
sA = cov(TrainA');
sB = cov(TrainB');
```

```
% separation vector
v = (sA+sB)\(mA-mB);
v = v/norm(v);
```

Now we can classify the data points, i.e the column vectors, for the data set *DataA* and *DataB*, while counting the false classifications, as described before:

```
c = v'*(mA+mB)/2;
classifyA = 0;
classifyB = 0;
missA = 0;
missB = 0;
for i=1:size(TestA,2)
    if v'*TestA(:,i) > c
        classifyA = classifyA+1;

    else
        missA = missA + 1;
    end
end

for i=1:size(TestB,2)
    if v'*TestB(:,i) <= c
        classifyB = classifyB+1;

    else
        missB = missB + 1;
    end
end
```

From the vectors *MissA* and *MissB* we can compute the miss rate and the success rate of the classifier.

```
missRate = (missA+missB)/(size(TestA,2)+size(TestB,2));
successRate = 1 - missRate
```

In this case we get the $successRate = 1$ for our toy problem.

2 UCI Benchmark Problems

2.1 Building a classifier for different data sets

We will now build the classifier for the data set *sonar.mat* provided by the UCI Machine Learning Repository. The classifier for *ionosphere.mat* works analogously (apart from the singularity problem we will discuss later).

The dataset *sonar.mat* consists of the 208×60-matrix *sonar_ data*, where 208 data points are stored in the rows, each with 60 features stored in the columns. Further there is the vector *sonar_ label* with 0-1 entries representing the classes of each row of the data matrix (0 means class A and 1 means class B).

First we will separate *sonar_ data* into the vector *SonarA* and *SonarB*, where the data points with label 0 are stored in *SonarA* and with label 1 in *SonarB*.

```
load sonar.mat
sizeOfSonarA = sum(sonar_label == 0);
sizeOfSonarB = sum(sonar_label == 1);

sonarA = zeros(sizeOfSonarA, 60);
sonarB = zeros(sizeOfSonarB, 60);

%Store all sonar_data == 0 in SonarA

for i = 1:size(sonar_label)
    if sonar_label(i) == 0
        sonarA(i,:) = sonar_data(i,:);
    end
end

%Deleting the 0 rows
ind = find(sum(sonarA,2)==0) ;
sonarA(ind,:) = [] ;

%Same for SonarB (sonar_data == 1)

for i = 1:size(sonar_label)
    if sonar_label(i) == 1
```

```
        sonarB(i,:) = sonar_data(i,:);
    end
end


ind = find(sum(sonarB,2)==0) ;
sonarB(ind,:) = [] ;
```

This will give us 2 matrices *SonarA* and *SonarB* with data points in class A and B. We now can split these matrices into training and testing data:

```
%Create test and train data of SonarA and SonarB
m = ceil(size(sonarA,1)*(7/10));

trainSonarA = sonarA(1:m,:);
testSonarA = sonarA(m+1:end,:);

m = ceil(size(sonarB,1)*(7/10));

trainSonarB = sonarB(1:m, :);
testSonarB = sonarB(m+1:end,:);
```

From the test vectors *testSonarA* and *testSonarB* we can create the mean value vector and the covariance matrix to create the separation vector *vSonar* and the constant *cSonar*.

```
%Create test and train data of SonarA and SonarB
mSonarA = mean(trainSonarA)';
mSonarB = mean(trainSonarB)';
sSonarA = cov(trainSonarA);
sSonarB = cov(trainSonarB);

%Create the separation vector and the constant for the classification
vSonar = (sSonarA+sSonarB)\(mSonarA-mSonarB);
vSonar = vSonar/norm(vSonar);

cSonar = vSonar'*(mSonarA + mSonarB)/2;
```

Like in the toy problem from the first part, we have all the tools to test and report the success rate for classification by the linear discriminant analysis.

```
classifySonarA = 0;
classifySonarB = 0;
missSonarA = 0;
missSonarB = 0;

for i=1:size(testSonarA,1)
    if vSonar'*testSonarA(i,:)' > cSonar
        classifySonarA = classifySonarA+1;

    else
        missSonarA = missSonarA + 1;
    end
end

for i=1:size(testSonarB,1)
    if vSonar'*testSonarB(i,:)' <= cSonar
        classifySonarB = classifySonarB+1;

    else
        missSonarB = missSonarB + 1;
    end
end

missRateSonar = (missSonarA+missSonarB)/(size(testSonarA,1)+
    size(testSonarB,1));
successRateSonar = 1 - missRateSonar
```

This gives us following output:

```
successRateSonar =

    0.7742
```

For the data set *ionosphere.data* we follow the same procedure. One thing we notice is that for the covariance matrices Σ_A and Σ_B of *trainIonoA* and *trainIonoB* the sum $\Sigma_A + \Sigma_B$

is singular. Most likely one feature has such small values that it leads to rounding errors and the values are stored as 0, instead of very small numbers. This leads to a 0 column, which means the $det(\Sigma_A + \Sigma_B) = 0$, i.e the matrix is singular.
If we disregard that $\Sigma_A + \Sigma_B$ is singular, we get following success rate for ionosphere classification:

```
successRateIono =

    0
```

To avoid the singularity problem, instead of the inverse, we will now compute the pseudo inverse of $\Sigma_A + \Sigma_B$ to compute the separation vector *vIono*:

```
vIono = pinv(sIonoA+sIonoB)*(mIonoA-mIonoB);
vIono = vIono/norm(vIono);
```

This leads to following success rate:

```
successRateIono =

    0.8544
```

2.2 Trying different thresholds for constant c

In this part we are trying out different constants between $v^T m_A$ and $v^T m_B$ to see how sensitive the classification rate is to the choice of the constant c.
Computing the boundaries of the *sonar.mat* data set we get:

$$v^T m_A = -0,0150 \text{ and } v^T m_B = -0,0353$$

For $c = -0.02$ we get following output:

```
successRateSonar =

    0.7581
```

and for $c = -0.033$ we get:

```
successRateSonar =

    0.7097
```

After trying out several different constants we can conclude that the closer the constant c gets to the boundaries $v^T m_A$ and $v^T m_B$, the worse the LDA classification, i.e the success rate, gets.

The same phenomenon we can observe while changing the constant c in the *ionosphere.data* data set. Here we notice, the further the boundaries $v^T m_A$ and $v^T m_B$ are apart, the worse the LDA classification gets, when c is close to the boundaries.