# ITCS227\_Lab06\_Assignment: Introduction to Statistical Learning (By Hand)

Objective: By the end of this worksheet you will know how a classification decision line works, how model errors are calculated and steps towards designing dataset data collection for modelling tasks.

# Making decision lines in data:

Test Data

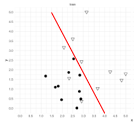
Training Data

1. Above is some data with two classes shown by shapes (**white triangles and black dots** ).   
   **With your hand, cover up the Test data** (the plot on the right) and then answer the question below.  
   1. **Draw a straight line across the** **training data** plot as a **good separating line for the two types of points**.   
      One side of the line should be for the white-triangles , and the other side should be for black-dots .

1. You can use Word’s line tool: **[Insert->Shapes->Line**] to draw a line.  
 Or use this red line (left). *2. There’s no perfect straight-line for our dataset, so you can choose any line. The ideal line will separate both classes perfectly.*

### Hint (expand for a helpful hint):

Here’s the kind of straight line you need to draw.   
It \*mostly\* separates the data-points of the two classes, but there are still some errors.

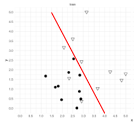
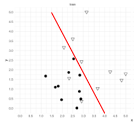


### Continue questions below:

* 1. Now, uncover the **Test data**, and **create the** **same line** on the righthand side plot.   
     *You can copy-paste your original line and drag it over.   
     The start and finish coordinates of your line should be the same on both plots.   
     This just means that your decision line will apply equally to both training data and test data.*

### Solution (Only open as a last resort!):

By example, here’s a straight line you might have drawn. You can choose any line.  
This line starts at [Y=0, X=4.0] and finishes at [Y=5.0, X=1.5].   
Copy-paste the line from the first plot to the second plot  
The X-Y coordinates (start and finish) will be the same in both plots.   
This is our decision line.

 -> 

### Continue questions below:

* 1. Next, compute the **confusion matrix** for the **training data**.  
       
     Count the matching and unmatched predictions by your decision line, and fill-in the counts for each colored cell in the table.  
     The sum of each “Actual” row should total 10, as there are 10 dots and 10 triangles in each plot.

|  |  |  |  |
| --- | --- | --- | --- |
|  | | Predicted | |
|  |  |
| Actual |  | ? | ? |
|  | ? | ? |

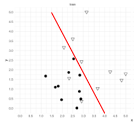
**Green boxes** mean the **Actual data-point** and the **Predicted decision data-point** (according to where you drew your line) are matched (i.e. correct decision)

* A dot is (Predicted) where a dot should be (Actual). (e.g. lower left in the plot)
* A triangle is (Predicted) where a triangle should be (Actual). (e.g. upper right in the plot)

**Orange boxes** means there’s an error (incorrect decisions). Either:

* A triangle is (Predicted) where a dot should be (Actual).
* A dot is (Predicted) where a triangle should be (Actual).

### Solution (Only open as a last resort!):

Here’s an example confusion matrix table for the red-line shown on the left.   
Your counts might be different, it depends on how you drew your line.

|  |  |  |  |
| --- | --- | --- | --- |
|  | | Predicted | |
|  |  |
| Actual |  | 10 | 0 |
|  | 3 | 7 |

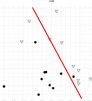
### Continue questions below:

* 1. Now, compute the **confusion matrix** for the **Test data:**

|  |  |  |  |
| --- | --- | --- | --- |
|  | | Predicted | |
|  |  |
| Actual |  | ? | ? |
|  | ? | ? |

### Solution (Only open as a last resort!):

Here’s the confusion matrix table for the Validation Data for the red-line shown on the left.   
Your counts might be different, it depends on how you drew your line.



|  |  |  |  |
| --- | --- | --- | --- |
|  | | Predicted | |
|  |  |
| Actual |  | 9 | 1 |
|  | 2 | 8 |

### Continue questions below:

* 1. Recall that the **number of errors** is simply the sum of incorrect decisions.   
       
     Calculate the error rate (the fraction of decision errors and its percentage) for the **training data**:

|  |  |
| --- | --- |
| Training error rate: | \_\_ / 20 = \_\_ = \_\_ % |

* 1. Calculate the error rate (the fraction of errors and its percentage) for the **Test data**:

|  |  |
| --- | --- |
| Test error rate: | \_\_ / 20 = \_\_ = \_\_ % |

### Solution (Only open as a last resort! Or after you have an answer):

The training error rate is 0+3 / 20 = 3/20 = 0.15 = 15 %.

The validation error rate is 1+2 / 20 = 3/20 = 0.15 = 15 %.

This model is nicely balanced – **a good “generalized fit”**. We know this because both training and validation error rates are equal.

If the training error was higher, we would say the model is “**overfitted**” (to the training data).

If the validation error was higher, we would say the model is “**underfitted**” (to the validation data).

More on this later

### Continue questions below:

# Designing a Dataset for Classification Modelling tasks:

Three examples of predictive classification modelling tasks are described below. For each, identify:

(i) the **classes** of the prediction outcomes.

(ii) what is an **observation** / what thing you would observe to measure this data.

(iii) **two** (possible) **numeric features** that will help to predict the outcome classes.

(iv) pick one class as a **reference class** (the class you are trying to predict). Estimate whether the value of each numeric feature will be **positively or negatively** **associated with the reference class**.

You can choose any two numeric features, as long as you can measure them, and they help you make the decision.

### Hint (expand/collapse for helpful hint):

For example - Will customers buy our new laptop product?

*Observation:*  
**Each time a customer asks a staff member about a laptop in the store and they buy or don’t buy that laptop**

*Two Classes:*  
**Buy / Not Buy***Reference Class:*  
**Buy**

Two numeric features:  
**1 cost price ( - )  
2 gigabytes of RAM ( + )**

Will customers buy our new laptop product?

* (Classes) – “Buys Laptop” or “Does not Buy Laptop”:
* (Observation) - Observe each time a customer asks a staff member about a laptop in the store and they buy or don’t buy that laptop.
* (Two numeric features): (1) cost price and (2) gigabytes of RAM.
* (Reference Class): “Buys Laptop”.
* Expected association to “Buys Laptop” -> lower price (-) and higher RAM (+).

### Continue questions below:

1. Create a model to predict whether a basketball player makes a free-throw shot.   
   (*e.g. numeric features about the player or court conditions, etc. , like number of \_\_\_\_ or amount of \_\_\_\_\_ )*

*Two Classes:*  
**1  
2***Reference Class:*  
**-**

*Two numeric features:*  
**1  
2**

Predict whether basketball player makes a free-throw shot

Two numeric features:

*Observation:***-**



1. Create a model to determine if a photo was taken indoors or outside.   
   (*e.g. numeric features from the captured camera image, like number of \_\_\_\_ or amount of \_\_\_\_\_ )*

*Two Classes:*  
**1  
2***Reference Class:*  
**-**

Predict whether a photo was taken indoors or outside. 

*Two numeric features:*  
**1  
2**

*Observation:***-**

### Solutions (Only open as a last resort!):

You might have chosen different options (for observations, numeric features and reference class) and that’s okay. This is a guideline:

Basketball

|  |  |
| --- | --- |
| Class 1 | Makes shot |
| Class 2 | Miss |
| Reference Class | Makes shot |
| Numeric Feature 1 | Height of player in meters (+) |
| Numeric Feature 2 | Distance from basket in meters (-) |
| Observation | Past free-throw shots from videos of basketball games. |

Photo is taken indoor or outside

|  |  |
| --- | --- |
| Class 1 | Indoor |
| Class 2 | Outdoor |
| Reference Class | Indoor |
| Numeric Feature 1 | Percentage of the picture with clouds or sky (+) |
| Numeric Feature 2 | Number of pixels with natural light color (+) |
| Observation | Collect past images of indoors and outdoors from public website with Creative Commons licensed images. |

Camera [Image credit: Robsonbillponte666, CC BY-SA 3.0](https://commons.wikimedia.org/wiki/File:Camera-icon.png)

### Continue questions below:

# Detecting Spam - Predicting Modelling tasks (by hand):

Now, we have a prediction task to check whether an **SMS message is spam or a legitimate message**. We trained a logistic regression model to make the decision.   
We set the target value spam=1 and legitimate=0; with the reference class as spam.  
The model has two input features:

* ***x1 =* the count of the number of numeric digits 0-9.**
* ***x2 =* the count of the number of words with two or more capital letters.**

Our training data and model fitting procedure found the following model: f(X).

**f(X) = -1.2 + 0.3 \* x*1* + 0.2 \* x*2***

The f(X) function returns a number indicating Spam, but it is not quite a probability for Spam.

So (in the logistic regression model algorithm) we send the returned number into the logistic function, which returns a probability value ([0.0-1.0]) for Spam (our reference class):

**P(Spam) = ypred = exp( f(X) ) / (exp( f(X) )+1)**

Commonly in code, we use **y\_pred** to store the predicted result from the model. So, in our case, **y\_pred** is the same as the probability for spam.

Answer the following questions, based on this model.

1. For each of the following messages, compute the probability that the SMS message is spam:

Example: “Congratulations, you have won 10,000 Baht. Text us back ASAP to claim!”

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  |  |  | *Apply fitted function:* | *Apply Logistic Function:* | *Probability of Spam  from the Logistic Function:* |
| *#* | *x1* | *x2* | **f(X)** | *~* | **P(Spam) =** |
| Example | 5 | 1 | 0.5 | exp(0.5) / (exp(0.5)+1) | 0.62 |

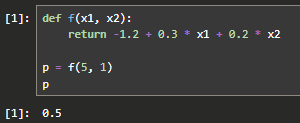
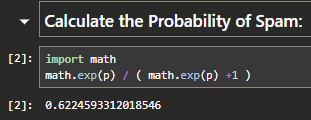
1. “OMG, our house just got invaded by BEES! I’m going to be 15-20 minutes LATE!”
2. “Hi Joe, you still owe me 20.00 dollars for dinner last night. Can you send Promptpay?”
3. “I am taking MATH-299 this semester. It is by far the WORST class I have ever taken! :-)”
4. “For 600 THB FREE! Go here: WWW.\*\*\*\*\*\*\*\*\*.COM/123”

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  |  |  | *Apply function:* | *Apply Logistic Function:* | *Probability of Spam  from the Logistic Function:* |
| *#* | *x1* | *x2* | **f(X)** | *~* | **P(Spam) =** |
| i. | ? | ? | ? | exp( ? ) / (exp( ? )+1) | ? |
| ii. |  |  |  | exp( ) / (exp( )+1) |  |
| iii. |  |  |  | exp( ) / (exp( )+1) |  |
| iv. |  |  |  | exp( ) / (exp( )+1) |  |

### Hint #1 (expand/collapse for helpful hint):

1. Count x1 and x2 numbers for each example, and fill-in those table.

Then we will calculate the answers using Jupyter Notebook code.

1. Open an Anaconda Terminal. CD to our lab directory.
2. Start `jupyter notebook`.
3. Open the CalculateSpam.ipynb notebook.
4. Type in the values of x1 and x2 into the f(x) function call and Run the cell to get the f(x) result.  
   
5. Run the next cell, to get the P(Spam) answer. Repeat Step 5. and 6. for each row in the table.  
   

### Solution (Only open as a last resort!):

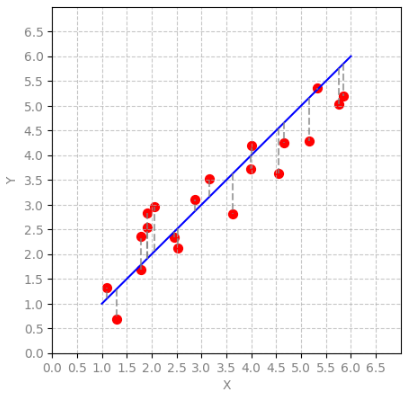
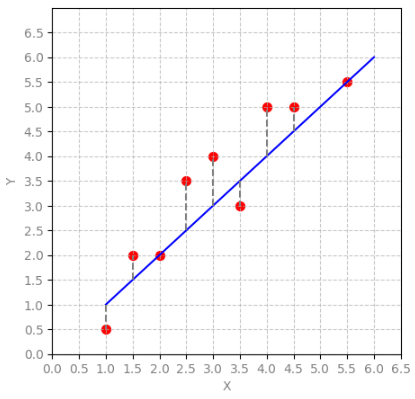
|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  |  |  | *Apply function:* | *Apply Logistic Function:* | *Probability of Spam  from the Logistic Function:* |
| *#* | *x1* | *x2* | **f(X)** | *~* | **P(Spam) =** |
| i. | 4 | 3 | 0.6 | exp( 0.6 ) / (exp( 0.6 )+1) | 0.645 |
| ii. | 4 | 0 | 0.0 | exp( 0.0 ) / (exp( 0.0 )+1) | 0.5 |
| iii. | 3 | 2 | 0.09 | exp( 0.09 ) / (exp( 0.09 )+1) | 0.52 |
| iv. | 6 | 4 | 1.4 | exp( 1.4 ) / (exp( 1.4 )+1) | 0.802 |

### Continue questions below:

# Find the Error Rate of a Regression Model (with numeric target):

Training Data

Test Data

1. Above are some data-points with **an already fitted** **estimator function (model)** from X to Y.   
     
   The **blue line** represents the simple fitted model (which is **X=Y**).

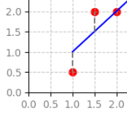
* There is 1 numeric input feature X (we will call **x1**), and 1 numeric output feature Y (target).
* The output target is numeric, therefore it’s a **regression** **task**!
* The dataset was split with **20 records** in the **training data**, and **9** **records** in the **test data**.

The **Sci-kit Learn** **Linear Regression** algorithm fitted the model to the training data, which added aslope **intercept at 0.0**, and found the best **coefficient for x1 was** **1.0**.

**f(X) = 0.0 + 1.0 \* x1**

The **training data** was already measured with a **mean absolute error (MAE) of 0.494.**

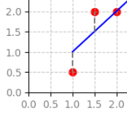
1. **With your hand, cover the training data (left plot),** and answer the following questions:

* **First, measure the (vertical) distance** from each **test data-point** to the **blue line** (**predicted value**), along the dashed-black line.  
    
  All the distances are in [0.0, 0.5, 1.0] and either plus or minus.  
    
  The first 3 data points are filled-in, now measure and fill in the remaining **6 points**:

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **- 0.5** | **0.5** | **0.0** | **?** | **?** | **?** | **?** | **?** | **?** |

### Hint (expand for a helpful hint):

For the first 3 points, we measure from the red-dot to the blue line, along the black-dashed line.

* All the distances are in [0.0, 0.5, 1.0] and either plus or minus.
* **From the picture below, we see the difference for the first (left-most) red-dot is**
* From 1.0 (blue) to 0.5 (red) is 1.0 – 0.5 = - 0.5 == 0.5 – 1.0 = - 0.5  
    
   

### Solution (Only open as a last resort!):

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **- 0.5** | **0.5** | **0.0** | **1.0** | **1.0** | **-0.5** | **1.0** | **0.5** | **0.0** |

### Continue questions below:

* Now, compute the **mean absolute error (MAE)** for your 9 measured distances of the **test data-points**:

|  |
| --- |
|  |

### Hint (expand for a helpful hint):

There are two steps to calculate Mean Absolute Error, from your error distances:

1. Change each number to its **absolute value** (only positive values).  
   In Python you can use **abs( -1.0 );** which will return 1.0  
   It is also simply removing the minus sign.
2. Next, with your nine (only positive) absolute values, **calculate the mean**.
3. That’s all! Enter that value into field above.

Btw - if the Error Rate is the same or similar between **Training Set Error Rate** and **Test Set Error Rate, then you have a “good” generalized fit of your model to the dataset.**

### Solution (Only open as a last resort!):

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **- 0.5** | **0.5** | **0.0** | **1.0** | **1.0** | **-0.5** | **1.0** | **0.5** | **0.0** |

Convert to absolute:

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **0.5** | **0.5** | **0.0** | **1.0** | **1.0** | **0.5** | **1.0** | **0.5** | **0.0** |

Calculate Mean:

|  |
| --- |
| e.g.  distances = [0.5,0.5,0.0,1.0,1.0,0.5,1.0,0.5,0.0]  np.mean( [ abs(d) for d in distances] )  or  np.mean(distances)  **0.555** |

### End of questions.

# Finished! Well done!

That’s it! Congratulations!

Now Right-Click on the “4. Finished!” heading -> Expand/Collapse -> Click “Expand All Headings” to check you have answered all questions and to read the ***Solutions*** to check your answers.

When you are happy, call for an LA to check your solution. Then, upload your DOCX file on MyCourses.