TP. Introduction to Machine Learning

Goal

In this TP we are going to work on the classification method of Logistic Regression. You are going to:

1. Describe a set of images by means of a set region descriptors.

2. Using the computed descriptors, we are going to simulate a classification experiment and calculate some performance metrics from the results of the Logistic Regression classifier. In this experiment, we are going to divide the dataset into two disjoint sets: training and test.

The general script of this TP is called `main_TP_LogisticReg.m`. You will have to read the code and comments, follow it, and complete the missing parts to implement both the description and the classification parts.

The set of region descriptors that we are going to use are called ShapeFeat, and they are explained in the paper **"Combining shape and contour features to improve tool wear monitoring in milling processes"**, that is provided along with this TP, and also can be found in the link: https://doi.org/10.1080/00207543.2018.1435919.

As a convention, once the feature vector of each cutting edge is described, it will be stored as a row in the matrix `X`. Therefore, each row will be a different pattern, and each column will be a variable.

1) PART 1: Description of the images

The first step in this TP is to describe the images.

The images are binary regions corresponding to cutting edges of insert tools used in milling processes. A diagram depicted how these have been obtained is shown in Figure 1.

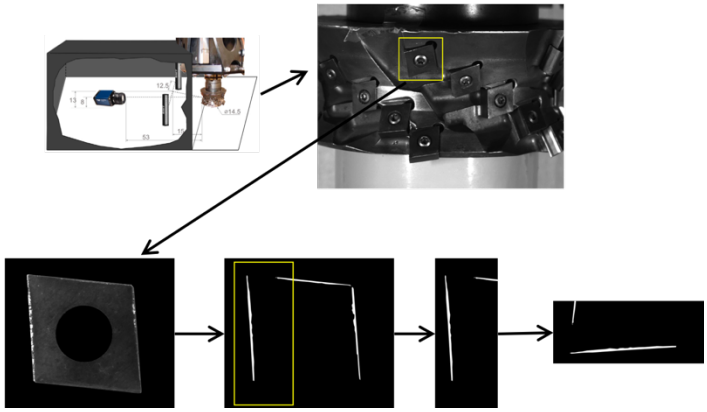

Figure 1. Diagram of the extraction of the cutting edge masks

From these binary regions you will extract 10 features, which form the descriptor called ShapeFeat, explained on pages 3-4 of the paper mentioned above. They are also explained in comments of the file `fGetShapeFeat.m`, which you will have to complete.

TASK 1: Finish the function `fGetShapeFeat` which, given an image with a binary region, extracts the ShapeFeat feature vector.

TASK 2: Take a look at **regionprops** function in Matlab's help. What does it do? How can it help for extracting the ShapeFeat descriptor? What additional information does it provide?

2) Division of the data between Train and Test sets.

In this second part of the main script you will not have to change anything from the script. In this block of code, the descriptors are loaded and this data is normalized, so that all variables of the dataset have mean 0 and standard deviation 1. It usually helps classification because it makes the ranges of the different variables to be more uniform.

Afterwards, the data is randomly split in two disjoint sets: one will be used for training the classifier and the other one will be used for making the test and, therefore, assess how good the classifier is.

When designing a classifier, we need to know the classification error to **see if it is able to generalize well with data that it has not seen before** (i.e., the test set). Therefore, the patterns of the test set are classified with the classifier trained with the training set, and its output is compared with the real classes of the classified test set samples. If we used the training data for this task, we would be favouring hypothesis functions that are too focused on the training data, so it might happen that the classifier is not able to generalize predictions for new, unknown data.

3) PART 2: Classification

This second part of the TP will be divided into two sub-parts: First, the classifier will be trained using the training set and then it will be used for classifying the test set. Secondly, the results will be assessed by means of the confusion matrix, the accuracy and the F-Score.

3.1) Training of the classifier and classification of the test set

On this first sub-part, the classifier is trained using the training set and, once it is trained, the elements of the test set are classified.

The goal of the **training** of the Logistic Regression classifier is to estimate the parameters $\boldsymbol{\theta} = [\theta_0, \theta_1, \ldots, \theta_n]$ used to estimate the probability that a certain pattern $\mathbf{x}^{(i)} = [x_1^{(i)}, x_2^{(i)}, \ldots, x_n^{(i)}]$ belongs to the positive class. In logistic regression, the **hypothesis function** h is the sigmoid function, whose output is bounded between 0 and 1, as shown in Figure 2. The equation of such function is:

$$h_{\boldsymbol{\theta}}(\mathbf{x}) = g(\boldsymbol{\theta}^T \mathbf{x}) = \frac{1}{1 + e^{-\boldsymbol{\theta}^T \mathbf{x}}}$$

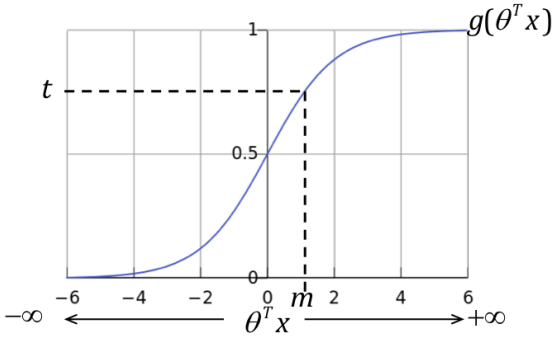

Figure 2. Sigmoid function

During the training phase, the parameters $\boldsymbol{\theta}$ will be modified so that they minimise a **cost function**, which will yield the average error between the outputs of the function h and for the training elements and their real classes. In Logistic Regression, this cost function is given by the following equation:

$$J(\boldsymbol{\theta}) = -\frac{1}{m}\left[\sum_{i=1}^{m} y^{(i)} \ln(h_{\boldsymbol{\theta}}(\mathbf{x}^{(i)})) + (1 - y^{(i)}) \ln\left(1 - h_{\boldsymbol{\theta}}(\mathbf{x}^{(i)})\right)\right]$$

This optimization is carried out by means fo the **gradient descent** algorithm, which is an iterative procedure in which each of the components of the vector $\boldsymbol{\theta}$ are updated on each iteration:

$$\theta_j^{(k)} = \theta_j^{(k-1)} - \alpha \frac{1}{m} \sum_{i=1}^{m} (h_{\boldsymbol{\theta}}(\mathbf{x}^{(i)}) - y^{(i)}) \cdot x_j^{(i)},$$

where the element $x_j^{(i)}$ is the j-th feature of the feature vector that represents the i-th object (i.e. the i-th pattern). Be aware that the value of the element x_0 is always 1. In practice, it will be obtained adding the value 1 at the beginning of the feature vector.

For the sake of simplicity, in this practice the **stop condition** of the gradient descent algorithm is reaching a maximum number of iterations (which are indicated in the parameter `max_iter` in the code).

On each iteration of the gradient descent algorithm (implemented in the function `fTrain_LogisticReg`) **three steps** are carried out:

1. The value of the **output of the function *h*** is calculated **for each feature vector** of the training set.
2. The components of $\boldsymbol{\theta} = [\theta_0, \theta_1, \ldots, \theta_n]$ are **updated**.
3. Calculation of the **value of the cost function (*J*)** with the new values of $\boldsymbol{\theta}$.

If the training has been carried out correctly, the cost should be smaller on each iteration, as depicted in Figure 3.

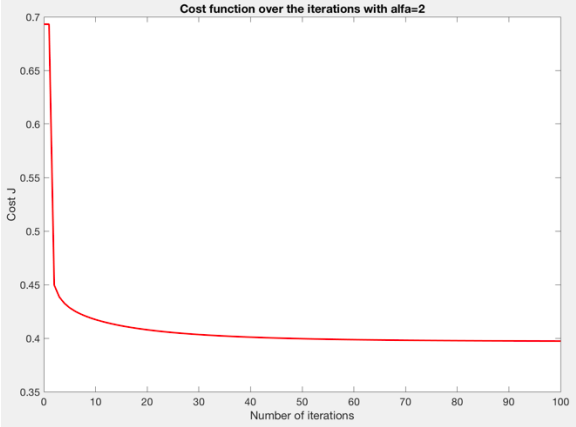

Figure 3. Value of the cost function in the training phase during 100 iterations of the gradient descent algorithm

Once the training has finished, these parameters will be used to classify new elements, i.e., to estimate the probability that a pattern belongs to the positive class.

a. Training

In this TP, the training phase is implemented in the function **`fTrain_LogisticReg`**. The **input parameters** to this function are the following:

- **`X_train`**: Matrix with dimensions `(m x n)` with the training data, where m is the number of training patterns (i.e. elements) and n is the number of features (i.e. the length of the feature vector which characterizes the object).

- **`Y_train`**: Vector that contains the classes of the training patterns. Its length is `n`.

- **`alpha`**: Learning rate for the gradient descent algorithm.

As output parameter this function returns **`theta`**, a vector whose length is `(n+1)` (one more element than the number of features on each pattern). It contains the parameters `theta` of the function *h* obtained after the training.

TASK: Open the file **`fTrain_LogisticReg.m`**, study the code and finish the implementation by completing the parts in the code indicated by:

```
% ====================== YOUR CODE HERE ======================
```

```
% ============================================================
```

To carry out this task you will also have to **complete and use**, the following functions:

- **`fun_sigmoid`** to calculate the result of the function *h* for **just one input pattern**.

- **`fCalculateCostLogReg`** to calculate the cost **of just one** output of the function *h*.

b. Classification

Once the classifier is trained (i.e., the vector $\boldsymbol{\theta}$ for which the cost reaches a minimum has been obtained), the classification of **elements that have not been used in the classification** is carried out using $\boldsymbol{\theta}$. In this TP, it is done on the function **`fClassify_LogisticReg`**, which returns the probability for each element on the test set to belong to the positive class. Its input parameters are:

- **`X_test`**: Matrix with dimension `(m_t x n)` containing the test data, where `m_t` is the number of test patterns and `n` is the number of features (i.e. the length of the feature vector that define each element).

- **`theta`**: Vector with length `n` (i.e., the number of features of each pattern along with the parameters theta of the estimated function *h* function after the training.

TASK: Open the file **`fClassify_LogisticReg.m`**, study the code and finish the implementation by completing the parts in the code indicated by:

```
% ====================== YOUR CODE HERE ======================


% ============================================================
```

At the end of the classification, the evolution of the cost function is shown in a separate figure. The learning rate α is set in the main script. Change the value of the learning rate α and see how the cost function graph looks like depending on the you have set. **Mention what you have seen and try to provide an explanation.**

3.2) Assessment of the performance of the classifier

The logistic regression classifier returns the probability of a pattern to belong to the positive class. However, in order to assign a pattern to one or another class it is necessary to determine a value, called decision threshold, such that **when the output of the classifier is higher than that threshold, the assigned class is positive** (see Figure 2). Normally the value of the decision threshold is 0.5.

Once the estimated classes of all the elements of the test set, we can calculate performance metrics, based on the confusion matrix. The most basic one is the **accuracy**. Which is the **percentage of the elements that have been well classified**. However, this is not always the best possible metric, especially when the test set is very imbalanced (i.e. there are more elements from one class than from the other one). For example, let us suppose that we want to build a classifier to diagnose a cancer which is known to be suffered by 1% of the population.

If the classifier would predict that everybody is healthy, it would have an accuracy of 99%. However, **this does not mean that the classifier is performing well**.

There are other metrics "fairer" to measure the performance of a classifier, such as precision, recall, or the F1-Score. Given the confusion matrix of the classifier:

		Predicted class	
		1	0
Real class	1	TP	FN
	0	FP	TN

where TP, TN, FP and FN are the number of **true positives, true negatives, false positives and false negatives**, respectively. For example, if FN=10, it means that the classifier has predicted 10 test elements as negatives (i.e., class 0) whereas they really belonged to the positive class (i.e., class 1).

$$accuracy = \frac{TP + TN}{TP + FN + FP + TN}$$

- **Precision:** It is the ratio of the true positives and the **total number of elements that the classifier has predicted as positives**.

$$precision = \frac{TP}{pred\_positive} = \frac{TP}{TP + FP}$$

- ***Recall*:** Ratio of the true positives and the elements that the number of elements **whose real class is positive**.

$$recall = \frac{TP}{P} = \frac{TP}{TP + FN}$$

- ***F-Score*:** It is the harmonic mean between the precision and the recall. Therefore, it gives an idea of the balance between both measurements.

$$F_1 = 2 \cdot \frac{precision \cdot recall}{precision + recall}$$

TASK: The calculation of the performance metrics is carried out in the final part of the main script, **`main_TP_LogisticReg`**. It shows the confusion matrix of the classifier in a new figure. You will have to complete the code to calculate the TP, FP, FN and TN by comparing the true classes and the classes assigned by the classifier. With these, you will have to calculate the accuracy and F1-score, which are shown in the screen.

```
% ====================== YOUR CODE HERE ======================


% ============================================================
```

Now, looking at the confusion matrix, accuracy and F1-score, what conclusion can you extract from the classification of this dataset?

TASK: You have seen in the theoretical part what is a ROC curve and how it can be calculated. Once you have finished the previous tasks, **you will have to write a function to calculate the ROC curve to evaluate the output of the classifier.**

FOR FUTURE SESSIONS: You will have to try with other classifiers, such as Support Vector Machines (SVM) or Neural Networks. They have not been explained in theory, so you have to make a little bit of research about how they work. Find below a few tips:

- **SVM**: There are built-in functions in Matlab to train an SVM and use it for classification. I also recommend the use of the library libsvm (https://www.csie.ntu.edu.tw/~cjlin/libsvm/), which is very easy to install and use in Matlab. You can use either the Matlab or the libsvm functions. **The parameters that make the most difference in the classification performance are the soft margin parameter C of the classifier and the kernel type. Therefore, try using different values of C and different kernel types, realize and report the differences in classification performance.**

- **Neural Networks**: We are referring to the classical shallow Neural Networks (i.e. not the Neural Networks you would use in Deep Learning, although they share similar principles). In order to understand what they are and how they work, there is a nice introduction in this website: https://www.doc.ic.ac.uk/~nd/surprise_96/journal/vol4/cs11/report.html. In order to understand how you can use Neural Networks in Matlab (both in GUI tools and **from the command line in a program**), you can follow the links in this website: https://www.mathworks.com/help/deeplearning/gs/shallow-networks-for-pattern-recognition-clustering-and-time-series.html, particularly "Fit Data with a Shallow Neural Network" for the **training** and "Classify Patterns with a Shallow Neural Network" for **classifying the elements in the test set**. The network will have a maximum of two hidden layers, and you can choose the number of neurons on each hidden layer. Try with different numbers of neurons, realize and report the differences in classification performance.

TO THINK ABOUT:

1 - Is there any difference between the accuracy and the F-Score? If there is, can you explain the difference and why there is such difference?

2 - Apart from using a different classifier, is there another way to improve the classification performance?

3 - Can you cite and explain other geometrical (i.e. shape, contour,…) descriptors that you would use to describe the regions of the wear? If you have time enough, you can extract them from the images and try to classify the wear level using them.