German University in Cairo
Media Engineering and Technology
Prof. Dr. Slim Abdennadher

Concepts of Programming Languages, Spring term 2017
Project Description
"kNN Classifier"

In this project, you will be introduced to a very basic Machine Learning algorithm, namely the k-Nearest-Neighbors classifier. You will use this algorithm to predict whether a student will pass or fail in their final exam, given their midterm and quiz scores. The word classifier means that we're trying to develop an algorithm, that given previous students (from previous years) and their labels (student grades and pass/fail as a label in this case), we want to predict the labels for new students (from this year), given only the new students' grades.

A kNN classifier in this case will take as an input two main parameters: a list of students who we wish to classify whether they pass, and a list of old students who we know their midterm and quiz scores, and we also know if the passed/failed from their previous exams. So for example, we know for the year 2013 the grades of the students and whether they passed or not, and we're trying to predict whether some students from 2017 will pass or not.

When predicting whether a new student will pass or not, the algorithm works as follows:

a) Computes the distance between a new student and all the old students (using any distance metric, such as the Euclidean distance).

b) Selects the top `K` closest old students based on the previously computed distances.

c) Chooses the most repeated label (pass/fail) in top K as the predicted label (pass/fail) for the new student.

Used Types

There should be three main types:

```
data Ex = Ex Float Float String String deriving Show
data NewSt = NewSt Float Float String deriving Show
data Dist = Dist Float NewSt Ex deriving Show
```

The `Ex` type has 4 parameters: the value of the midterm, value of the quizzes, name of the student, and pass/fail as a label. The `NewSt` type has the same parameters except the pass/fail label, since this is what we're trying to predict. The `Dist` type has three parameters, the first is the distance between the last two parameters (A new student and an old student).

As an example: a new student can be represented as `(NewSt 50 30 "alex")`, where `NewSt` is the name of the constructor, 50 is the midterm grade, 30 is the quizes grade, and "alex" is the student name. A list of old students can be represented as `[(Ex 20 30 ‘‘carmen’’ ‘‘pass’’), (Ex 20 5 ‘‘julia’’ ‘‘fail’’)]`, where `Ex` is the name of the constructor, and the last parameter indicates whether this student has passed/failed a previous exam, which we call the "label".

An example for a `Dist` object would be: `Dist 1.0 NewSt(2 3 "yasser") Ex(2 2 "peter" "pass")`. It indicates that the distance between the new student named "yasseränd the ex student named ppeterïs equal to 1.0.

Example

```
classify_all euclidean 2
[(Ex 1 3 "john" "pass"), (Ex 2 2 "peter" "pass"),
(Ex 2 3 "clark" "fail"), (Ex 3 3 "douglas" "fail")]
[(NewSt 1 2 "mostafa"), (NewSt 3 4 "yasser")] = [("mostafa","pass"),("yasser","fail")]
```

Here we use a euclidean distance metric, with `n=2`, so we should select the top 2 closest old students. Given these parameters and the input lists, "mostafa" is closest to "john" and "peter", and both their labels is "pass", so the resultant label for him is the also "pass".

Functions to be added

The rest of the description will guide you through the functions that you need to implement to be able to solve the problem.

euclidean

`euclidean :: NewSt -> Ex -> Dist`
Takes as input a `NewSt` type and an `Ex` type, and returns an object of type `Dist`, which holds the euclidean distance between the two inputs.

manhattan

`manhattan :: NewSt -> Ex -> Dist`
Takes as input a `NewSt` type and an `Ex` type, and returns an object of type `Dist`, which holds the manhattan distance between the two inputs.

dist

`dist :: (a -> b -> c) -> a -> b -> c`
Takes as an input a generic distance function (euclidean/manhattan) and two points, and returns a `Dist` object holding the distance function applied on the two points.

all_dists

`all_dists :: (a -> b -> c) -> a -> [b] -> [c]`
Takes as an input a distance function, a data-point `x` (for example a NewSt), and a list of data-points `lst` (for example a list of Ex). Returns a list of `Dist`s, which is all the distances from `x` to elements in `lst`.

takeN

`takeN :: Num a => a -> [b] -> [b]`
Given a number `n` and a list `l`, the function should return the first `n` elements from the list.

In the following functions, you may find it useful to use the functions `sortBy`, `maximumBy`, `groupBy`. **They can be used after importing** `Data.List`. Otherwise, you can write your own implementation for `sort, group, maximum`.

closest

`Num a => (b -> c -> Dist) -> a -> [c] -> b -> [Dist]`
Given a generic distance function, a number `n`, a list of `Ext` and a `St` object to be classified, returns a list of `Dist` objects. The list should be the closest `n` objects to the `St` object. You can use `sortBy` from Data.List, or implement your own sorting function.

Example:

```
closest euclidean 3 [(Ex 1 3 "john" "pass"), (Ex 2 2 "peter" "pass"),
(Ex 2 3 "clark" "fail"), (Ex 3 3 "douglas" "fail")] (NewSt 1 1 "ted")
=
[Dist 1.414214 (NewSt 1.0 1.0 "ted") (Ex 2.0 2.0 "peter" "pass"),
Dist 2.0 (NewSt 1.0 1.0 "ted") (Ex 1.0 3.0 "john" "pass"),
Dist 2.236068 (NewSt 1.0 1.0 "ted") (Ex 2.0 3.0 "clark" "fail")].
```

grouped_dists

`grouped_dists :: Num a => (b -> c -> Dist) -> a -> [c] -> b -> [[Dist]]`
Given a generic distance function, a number `n`, a list of `Ext` and a `St` object to be classified, returns a list of lists of `Dist` objects. There should be 2 lists in the the returned list, the first for "pass" and second for "fail", filled with the correct elements from the function `closest`. You can use `groupBy` from Data.List, or implement your own grouping function.

Example for custom grouping function (without groupBy):

```
myGroup [Dist 1.414214 (NewSt 1.0 1.0 "ted") (Ex 2.0 2.0 "peter" "pass"),
Dist 2.0 (NewSt 1.0 1.0 "ted") (Ex 1.0 3.0 "john" "pass"),
Dist 2.236068 (NewSt 1.0 1.0 "ted") (Ex 2.0 3.0 "clark" "fail")]
=
[
[Dist 1.414214 (NewSt 1.0 1.0 "ted") (Ex 2.0 2.0 "peter" "pass"),
Dist 2.0 (NewSt 1.0 1.0 "ted") (Ex 1.0 3.0 "john" "pass")],
[Dist 2.236068 (NewSt 1.0 1.0 "ted") (Ex 2.0 3.0 "clark" "fail")]
]
```

Example for `grouped_dists`:

```
grouped_dists euclidean 3 [(Ex 1 3 "john" "pass"), (Ex 2 2 "peter" "pass"),
(Ex 2 3 "clark" "fail"), (Ex 3 3 "douglas" "fail")] (NewSt 1 1 "ted") =
[
[ Dist 1.414214 (NewSt 1.0 1.0 "ted") (Ex 2.0 2.0 "peter" "pass"),
Dist 2.0 (NewSt 1.0 1.0 "ted") (Ex 1.0 3.0 "john" "pass") ],
[ Dist 2.236068 (NewSt 1.0 1.0 "ted") (Ex 2.0 3.0 "clark" "fail") ]
].
```

mode

`mode :: Num a => (b -> c -> Dist) -> a -> [c] -> b -> [Dist]`
Given a generic distance function, a number `n`, a list of `Ext` and a `St` object to be classified, returns a list of `Dist` objects. The list should contain the `Dist` objects of only the most frequent label; for example if there are two `Dist` objects with their `Ex` objects having the label "pass", and only one object with the label "fail", only the first two objects should be kept. You can use `maximumBy` from Data.List, or implement a function that returns the list (containing pass/fail) that has the greater length between two lists (as returned from `grouped_dists`).

Example for custom implementation for maximumLength (without using maximumBy):

```
maximumLength
[
[Dist 1.414214 (NewSt 1.0 1.0 "ted") (Ex 2.0 2.0 "peter" "pass"),
Dist 2.0 (NewSt 1.0 1.0 "ted") (Ex 1.0 3.0 "john" "pass")],
[Dist 2.236068 (NewSt 1.0 1.0 "ted") (Ex 2.0 3.0 "clark" "fail")]
] =
[Dist 1.414214 (NewSt 1.0 1.0 "ted") (Ex 2.0 2.0 "peter" "pass"),
Dist 2.0 (NewSt 1.0 1.0 "ted") (Ex 1.0 3.0 "john" "pass")]
```

Example for mode:

```
mode euclidean 3 [(Ex 1 3 "john" "pass"), (Ex 2 2 "peter" "pass"),
(Ex 2 3 "clark" "fail"), (Ex 3 3 "douglas" "fail")] (NewSt 1 1 "ted") =
[Dist 1.414214 (NewSt 1.0 1.0 "ted") (Ex 2.0 2.0 "peter" "pass"),
Dist 2.0 (NewSt 1.0 1.0 "ted") (Ex 1.0 3.0 "john" "pass")]
```

label_of

`label_of :: [Dist] -> ([Char],[Char])`
Given a list of `Dist` objects, extracts a tuple from the first element of the list containing the name of the `NewSt` object and the label of the `Ex` object.

```
labelOf [Dist 1.414214 (NewSt 1.0 1.0 "ted") (Ex 2.0 2.0 "peter" "pass"),
Dist 2.0 (NewSt 1.0 1.0 "ted") (Ex 1.0 3.0 "john" "pass"),
Dist 2.236068 (NewSt 1.0 1.0 "ted") (Ex 2.0 3.0 "clark" "fail")]
= ("ted","pass")
```

classify

`classify :: Num a => (b -> c -> Dist) -> a -> [c] -> b -> ([Char],[Char])`
Given a generic distance function, a number `k`, a list of `Ex` and a `NewSt` object to be classified, returns a tuple containing the name of the `NewSt` object and the predicted label (most repeated class in the k-neighborhood).

```
classify euclidean 3 [(Ex 1 3 "john" "pass"),
(Ex 2 2 "peter" "pass"), (Ex 2 3 "clark" "fail")
, (Ex 3 3 "douglas" "fail")] (NewSt 1 1 "ted") =
("ted","pass")
```

classify_all

`classify_all :: (a -> b -> Dist) -> Int -> [b] -> [a] -> [([Char],[Char])]`

Given a generic distance function, a number `k`, a list `old` of `Ex` objects, and a list `new` of `NewSt` objects, returns the predictions for every element in `new` using the k-neighborhood from the list `old`. The result should be a list of tuples, containing the name and label of each element.

```
classify_all euclidean 3
[(Ex 1 3 "john" "pass"), (Ex 2 2 "peter" "pass"),
(Ex 2 3 "clark" "fail"), (Ex 3 3 "douglas" "fail")]
[(NewSt 1 1 "ted"), (NewSt 3 4 "frank")]
=
[("ted","pass"),("frank","fail")]
```

Submission Guidelines

- The deadline of the project will be on **26th of April**.
- Each team should submit a single *.hs* file, via the MET website submission link, containing the team's full project implementation. The submitted file **must** abide by the following rules:
  a) The file should be named according to your team's *Team Name* (from the to-be-posted teams list).
  b) You should include a clear documentation per implemented function. You should write each function's documentation above the function's implementation.

- It is the team's full responsibility to successfully submit a valid .hs file before the project's deadline.