Fouille de données – TD 7

On va travailler sur des meta-données de cellules potentiellement cancéreuses: dataset, description.
Faites tout votre TD dans un fichier `td7.py`, que vous rendrez via Moodle avant 23h59 ce soir.

Exercice 1: Lecture du dataset, encodage

Implémenter cette fonction dans votre fichier `td7.py` :

```
def read_data(filename):
  """Reads a breast-cancer-diagnostic dataset, like wdbc.data.

  Args:
    filename: a string, the name of the input file.

  Returns:
    A pair (X, Y) of lists:
    - X is a list of N points: each point is a list of numbers
      corresponding to the data in the file describing a sample.
      N is the number of data points in the file (eg. lines).
    - Y is a list of N booleans: Element #i is True if the data point
      #i described in X[i] is "cancerous", and False if "Benign".
  """
```

Se référer aux TDs précédents si vous avez oublié comment lire un fichier. On pourra utiliser `line.split(',')` pour convertir un CSV string `line` en une liste de strings.
Pour convertir un string représentant un nombre en `float`, rien de plus simple: la fonction `float()`.
Exemple:
Si le fichier `/tmp/tmp.txt` contient le texte suivant:

```
1234,M,2.3,1.0,0.5
1235,B,1.1,3.2,0.9
1235,B,0.2,0.1,0.23
1236,M,4.1,1.9,4
```

Alors `read_data('/tmp/tmp.txt')` doit renvoyer:

```
([[2.3, 1.0, 0.5], [1.1, 3.2, 0.9], [0.2, 0.1, 0.23], [4.1, 1.9, 4.0]],
 [True, False, False, True])
```

Exercice 2: Distance euclidienne

Implémenter cette fonction dans votre fichier `td7.py` :

```
def simple_distance(data1, data2):
  """Computes the Euclidian distance between data1 and data2.

  Args:
    data1: a list of numbers: the coordinates of the first vector.
    data2: a list of numbers: the coordinates of the second vector (same length as data1).

  Returns:
    The Euclidian distance: sqrt(sum((data1[i]-data2[i])^2)).
  """
```

Exemple: `simple_distance([1.0, 0.4, -0.3, 0.15], [0.1, 4.2, 0.0, -1]) = 4.081972562377166`

Exercice 3: K Nearest Neighbors

Implémenter cette fonction dans votre fichier `td7.py` :

```
def k_nearest_neighbors(x, points, dist_function, k):
  """Returns the indices of the k elements of "points" that are closest to "x".

  Args:
    x: a list of numbers: a N-dimensional vector.
    points: a list of list of numbers: a list of N-dimensional vectors.
    dist_function: a function taking two N-dimensional vectors as
       arguments and returning a number. Just like simple_distance.
    k: an integer. Must be smaller or equal to the length of "points".

  Returns:
    A list of integers: the indices of the k elements of "points" that are
    closest to "x" according to the distance function dist_function.
    IMPORTANT: They must be sorted by distance: nearest neighbor first.
  """
```

Exemple:

```
k_nearest_neighbors([1.2, -0.3, 3.4],
                    [[2.3, 1.0, 0.5], [1.1, 3.2, 0.9], [0.2, 0.1, 0.23], [4.1, 1.9, 4.0]],
                    simple_distance, 2)
```

doit renvoyer `[2, 0]`

Exercice 4: Prédiction

Séparer le dataset, grâce à la fonction `split_lines` d'un TD précédent (corrigé), en un fichier `'train'` et un fichier `'test'`.

Puis implémenter la fonction suivante dans `td7.py` :

```
def is_cancerous_knn(x, train_x, train_y, dist_function, k):
  """Predicts whether some cells appear to be cancerous or not, using KNN.

  Args:
    x: A list of floats representing a data point (in the cancer dataset,
       that's 30 floats) that we want to diagnose.
    train_x: A list of list of floats representing the data points of
       the training set.
    train_y: A list of booleans representing the classification of
       the training set: True if the corresponding data point is
       cancerous, False if benign. Same length as 'train_x'.
    dist_function: A function taking two N-dimensional vectors as
       arguments and returning a number. Just like simple_distance.
    k: Same as in k_nearest_neighbors().

  Returns:
    A boolean: True if the data point x is predicted to be cancerous, False
        if it is predicted to be benign.
  """
```

Exemple:

```
is_cancerous_knn([1.2, -0.3, 3.4],
                 [[2.3, 1.0, 0.5], [1.1, 3.2, 0.9], [0.2, 0.1, 0.23], [4.1, 1.9, 4.0]],
                 [True, False, True, False], simple_distance, 2)
```

Doit renvoyer `True`. Si vous changez le 3ème argument en `[False, False, True, False]` il doit renvoyer `True` encore. Si vous changez en `[False, False, False, False]` il doit renvoyer `False`.

Exercice 5: Évaluation

Implémenter cette fonction dans votre fichier `td7.py`:

```
def eval_cancer_classifier(test_x, test_y, classifier):
  """Evaluates a cancer KNN classifier.

  This takes an already-trained classifier function, and a test dataset, and evaluates
  the classifier on that test dataset: it calls the classifier function for each x in
  test_x, compares the result to the corresponding expected result in test_y, and
  computes the average error.

  Args:
    test_x: A list of lists of floats: the test/validation data points.
    test_y: A list of booleans: the test/validation data class (True = cancerous,
       False = benign)
    classifier: A classifier, i.e. a function x→y whose sole argument x is of the
       Same type as an element of train_x or test_x, and whose return value y is
       the same type as train_y or test_y. For example:
       lambda x: is_cancerous_knn(x, train_x, train_y, dist_function=simple_distance, k=5)

  Returns:
    A float: the error rate of the classifier on the test dataset. This is
    a value in [0,1]: 0 means no error (we got it all correctly), 1 means
    we made a mistake every time. Note that choosing randomly yields an error
    rate of about 0.5, assuming that the values in test_y are all Boolean.
  """
```

Essayons sur notre dataset en utilisant l'exemple donné dans le commentaire de 'classifier' pour diverses valeurs de k, et pour `train_x` etc on les extraira des fichiers `'train' et 'test'` via `read_data()`).

- Quel taux d'erreur obtenez-vous pour k=1? Pour k=10? Pour k=100?
    - Ca devrait être entre 5% et 15%, sinon vous avez sans doute un bug.
- **Vérifiez** qu'en injectant le fichier 'train' à la fois en argument de training et de test, on obtient bien un taux d'erreur de zéro si on prend k=1 (comprenez-vous pourquoi?).

Exercice 6: Validation Croisée 1 / 2: Évaluation sur l'ensemble d'entraînement

Implémenter cette fonction dans votre fichier `td7.py`.

```
def cross_validation(train_x, train_y, untrained_classifier):
  """Uses cross-validation (with 5 folds) to evaluate the given classifier.

  Args:
    train_x: Like above.
    train_y: Like above.
    untrained_classifier: Like above, but also needs training data:
       untrained_classifier should be a function taking 3 arguments (train_x, train_y, x).
       For example:
       untrained_classifier = lambda train_x, train_y, x: is_cancerous_knn(x, train_x,
           train_y, dist_function=simple_distance, k=5)
  Returns:
    A float, like above (the average error rate evaluated across all folds).
  """
```

On pourra utiliser KFold ou StratifiedKFold de sklearn, ou ré-implémenter l'algo soi-même.

Exercice 7: Validation Croisée 2 / 2: Optimisation de paramètre

Implémenter cette fonction dans votre fichier `td7.py`.

```
def find_best_k(train_x, train_y, untrained_classifier_for_k):
  """Uses cross-validation (10 folds) to find the best K for the given classifier.

  Args:
    train_x: Like above.
    train_y: Like above.
    untrained_classifier_for_k: A function that takes FOUR arguments: train_x, train_y, k
       and x. Example:
       lambda train_x, train_y, k, x: is_cancerous_knn(x, train_x, train_y,
          dist_function=simple_distance, k)

  Returns:
    An integer: the ideal value for K in a K-nearest-neighbor classifier.
  """
```

Il faudra bien sûr réutiliser cross_validation() faite ci-dessus.
De plus, on pourra s'aider de la fonction suivante pour ne pas tester vraiment tous les K (**trop lent!**), mais plutot un échantillon "bien réparti" parmi les valeurs possibles:

```
import math
def sampled_range(mini, maxi, num):
  if not num:
    return []
  lmini = math.log(mini)
  lmaxi = math.log(maxi)
  ldelta = (lmaxi - lmini) / (num - 1)
  out = [x for x in set([int(math.exp(lmini + i * ldelta)) for i in range(num)])]
  out.sort()
  return out
```

Exemple: `sampled_range(1, 1000, 10) = [1, 2, 4, 9, 21, 46, 99, 215, 464, 999]`

Quel est le meilleur K sur notre dataset? Quel taux d'erreur obtient-on en test?

Exercice 8 (*): Distance pondérée.

Les 30 coordonnées de nos points n'ont pas toutes la même importance, et d'ailleurs elles n'ont pas toutes la même amplitude. Il est important de **pondérer** les coordonnées lors du calcul de la distance!

L'idée de base est de pondérer (pour chaque i) le terme (x_i-x'_i)^2 dans la distance euclidienne en le divisant par la variance des x_i sur l'ensemble du dataset (n'hésitez pas a demander une explication au tableau).

Vous pourriez avoir des idées plus poussées (à essayer si vous le souhaitez!). Par exemple, on peut essayer de favoriser les coordonnées qui semblent plus discriminatoires par rapport au diagnostic: vous pourriez évaluer la moyenne d'une certaine coordonnées dans le sous-dataset "cancéreux", la moyenne dans le sous-dataset "bénin", faire la différence, et comparer cette différence à l'écart-type: si la différence

est significativement plus grande que l'écart-type, on a une coordonnée discriminante, dont on voudrait sans doute "booster" le coefficient. Inversement pour une coordonnée non discriminante.

Vous êtes libre! La fonction ci-dessous peut faire les calculs qu'elle veut, et doit renvoyer une fonction de distance (c'est possible -- et facile -- en python). L'idée étant que cette fonction de distance sera meilleure que simple_distance pour la classification KNN, en tout cas c'est le but.

```
def get_weighted_dist_function(train_x, train_y):
  """Returns a (hopefully) good distance function for KNN classification.

  Args:
    train_x: List of lists of floats. As usual, see above.
    train_y: List of booleans. As usual, see above.

  Returns:    A distance function that seems suitable for KNN classification.
    For example, you could write "return lambda x, y: simple_distance(x, y)"
    and it would return a function equivalent to 'simple_distance'.
  """
```

Ré-essayer eval_cancer_classifier avec la fonction de distance renvoyée par `get_weighted_dist_function` (lancée sur l'ensemble de training correspondant au fichier train), avec le K qui était optimal pour simple_distance. Est-ce mieux? Ré-essayer de trouver le K optimal avec cette nouvelle fonction.

Exercice 9 (**): Optimisation des poids (de la distance pondérée)
Faites-le pour vous, si vous êtes motivés!
Idée générale: faire de la validation croisée *itérative* pour optimiser un par un les paramètres (les poids) de la distance pondérée.
Utilisation de RepeatedStratifiedKFold ?