Hyun-Joon Yang ¶

yanghyun@usc.edu

CSCI 360

Lab 3

In [1]:

```
import numpy as np
from lab3_utils import edit_distance, feature_names

# Hint: Consider how to utilize np.unique()
def preprocess_data(training_inputs, testing_inputs, training_labels, testing_labels):
    processed_training_inputs, processed_testing_inputs = ([], [])
    processed_training_labels, processed_testing_labels = ([], [])
    # VVVVV YOUR CODE GOES ERE VVVVV $

    # raw & processed inputs
    inputs = [training_inputs, testing_inputs]
    labels = [training_labels, testing_labels]
    processed = [[], []]
    # indices of ordinal vs discrete feature
    ordinals = {0, 2, 3, 5}
    discretes = {1, 4, 6, 7, 8}

    # pre-processing
    for i in range(len(inputs)):
        # basic constants
        m = inputs[i].shape[0]
        n = inputs[i].shape[1]

        # replace ?, deal with categorical
        for j in range(n):
            # convert to right data type
            feature = np.array(inputs[i][:,j], dtype=str)
            # find mode
            vals, occurs = np.unique(feature, return_counts=True)
            mode = vals[np.argmax(occurs)]
            # print(vals, occurs, mode)
            # replace ?
            if '?' in vals:
                feature = np.char.replace(feature, '?', mode)

            # deal with categorical
            if j in ordinals:
                mapper = {}
                # age
                if j == 0:
                    for k in range(9):
                        key = str(k+1)+'0-'+str(k+1)+'9'
                        mapper[key] = str(k+1)
                        feature = np.char.replace(feature, key, mapper[key])
                # tumor size
                elif j == 2:
                    for k in range(12):
                        lower = k*5
                        upper = (k+1)*5-1
                        key = str(lower)+'-'+str(upper)
                        mapper[key] = str(k+1)
                        feature = np.char.replace(feature, key, mapper[key])
                # inv-nodes
                elif j == 3:
                    for k in range(13):
                        lower = (k*3)
                        upper = lower + 2
                        if k == 12:
                            upper = lower + 3
                        key = str(lower)+'-'+str(upper)
                        mapper[key] = str(k+1)
                        feature = np.char.replace(feature, key, mapper[key])
                # deg-malig
                else:
                    for k in range(3):
                        key = str(k+1)
                        mapper[key] = str(k+1)
                        feature = np.char.replace(feature, key, mapper[key])
            elif j in discretes:
                # multiple levels
                if j in {1, 7}:
                    # menopause
                    if j == 1:
                        keys = ['lt40', 'ge40', 'premeno']
                    # breast-quad
                    else:
                        keys = ['left_up', 'left_low', 'right_up', 'right_low', 'central']
                    new_feature = np.zeros((m, len(keys)))
                    # create dict
                    indices = {}
                    for k in range(len(keys)):
                        indices[keys[k]] = k
```

```
                    # fill in auxilliary
                    for k in range(m):
                        new_feature[k, indices[feature[k]]] = 1
                    feature = new_feature
                # 2 levels
                else:
                    # breast
                    if j == 6:
                        keys = ['left', 'right']
                    # node-caps, irradiat
                    else:
                        keys = ['yes', 'no']
                    feature[feature == keys[0]] = 1
                    feature[feature == keys[1]] = 0

            # make processed matrix
            feature = np.array(feature, dtype=int)
            if feature.ndim < 2:
                feature = np.expand_dims(feature, axis=1)
            if type(processed[i]) == list:
                processed[i] = feature
            else:
                processed[i] = np.concatenate((processed[i], feature), axis=1)

        # turn labels into 0 or 1
        labels[i] = np.array(labels[i], dtype=str)
        labels[i] = np.char.replace(labels[i], 'no-recurrence-events', '0')
        labels[i] = np.char.replace(labels[i], 'recurrence-events', '1')
        labels[i] = np.array(labels[i], dtype=int)

    processed_training_inputs = processed[0]
    processed_testing_inputs = processed[1]
    processed_training_labels = labels[0]
    processed_testing_labels = labels[1]

    # print(processed_training_inputs)
    # print(processed_training_labels)
    # print(training_inputs[0,:], processed_training_inputs[0,:])
    # print(len(training_labels) == len(processed_training_labels))
    # print(processed_testing_inputs)
    # print(processed_testing_labels)
    # print(testing_inputs[0, :], processed_testing_inputs[0, :])
    # print(len(testing_labels) == len(processed_testing_labels))

    # ^^^^^ YOUR CODE GOES ERE ^^^^^ $
    return processed_training_inputs, processed_testing_inputs, processed_training_labels, processed_testing_labels



# Hint: consider how to utilize np.argsort()
def k_nearest_neighbors(predict_on, reference_points, reference_labels, k, l, weighted):
    assert len(predict_on) > 0, f"parameter predict_on needs to be of length 0 or greater"
    assert len(reference_points) > 0, f"parameter reference_points needs to be of length 0 or greater"
    assert len(reference_labels) > 0, f"parameter reference_labels needs to be of length 0 or greater"
    assert len(reference_labels) == len(reference_points), f"reference_points and reference_labels need to be the" \
                                                           f" same length"
    predictions = []
    # VVVVV YOUR CODE GOES ERE VVVVV $

    # get distances for every x, x*
    m_2 = predict_on.shape[0]
    m_1 = reference_points.shape[0]
    distances = np.zeros((m_1, m_2))
    for i in range(m_1):
        for j in range(m_2):
            distances[i, j] = edit_distance(reference_points[i, :], predict_on[j, :], l)
    # nonmatching or matching elements for when l = -1??

    # get k neighbors for each x*
    orders = np.argsort(distances, axis=0)
    neighbors = orders[0:k, 0:m_2]

    # find mode & add to predictions for each x*
    if weighted:
        scores = np.zeros(m_2)
        epsilon = 0.0001
        # calculate score
        for i in range(m_2):
            num = 0
            denom = 0
            for j in range(k):
                order = orders[j,i]
```

```
                y = reference_labels[order]
                dist = edit_distance(reference_points[order,:], predict_on[i,:], l)
                num += (y / dist)
                denom += (1/(dist + epsilon))
                # print(y, dist, num, denom)
            scores[i] = num / denom
        # classify
        scores[scores >= 0.5] = 1
        scores[scores < 0.5] = 0
        predictions = list(scores)
    else:
        for i in range(m_2):
            vals, occurs = np.unique(neighbors[:,i], return_counts=True)
            maxoccur = occurs[0]
            ind = 0
            for i in range(len(occurs)):
                if occurs[i] > maxoccur:
                    maxoccur = occurs[i]
                    ind = i
                # break tie in favor of recurrence
                elif occurs[i] == maxoccur:
                    if reference_labels[i] == 1:
                        ind = i
            mode = vals[ind]
            # mode = vals[np.argmax(occurs)]
            predictions.append(reference_labels[mode])

    # ^^^^^ YOUR CODE GOES ERE ^^^^^ $
    return predictions
```

In [2]:

```
from lab3_utils import (
    accuracy_score,
    load_data
)
from lab3 import k_nearest_neighbors, preprocess_data

k_max = 30
l_norms = [-1, 1, 2, 3, 4, 5, 6, np.inf]
weighting_options = [True, False]
inaccuracies = np.zeros((k_max, len(l_norms), len(weighting_options)))

raw_data = load_data()
processed_training_inputs, processed_testing_inputs, processed_training_labels, processed_testing_labels =\
    preprocess_data(*raw_data)

for k in range(1, k_max+1):
    for l_i, l_norm in enumerate(l_norms):
        for w, weighting in enumerate(weighting_options):
            predicted_labels = k_nearest_neighbors(
                predict_on=processed_testing_inputs,
                reference_points=processed_training_inputs,
                reference_labels=processed_training_labels,
                k=k,
                l=l_norm,
                weighted=weighting
            )
            inaccuracies[k-1, l_i, w] = 1-accuracy_score(processed_testing_labels, predicted_labels)
```

```
C:\Users\Hyun-\OneDrive\바탕 화면\Classes\FALL 2020\CSCI 360\Labs\Lab3\l3\lab3.py:167: RuntimeWarning: invalid value encoun
tered in true_divide
  num += (y / dist)
C:\Users\Hyun-\OneDrive\바탕 화면\Classes\FALL 2020\CSCI 360\Labs\Lab3\l3\lab3.py:172: RuntimeWarning: invalid value encoun
tered in greater_equal
  scores[scores >= 0.5] = 1
C:\Users\Hyun-\OneDrive\바탕 화면\Classes\FALL 2020\CSCI 360\Labs\Lab3\l3\lab3.py:173: RuntimeWarning: invalid value encoun
tered in less
  scores[scores < 0.5] = 0
C:\Users\Hyun-\OneDrive\바탕 화면\Classes\FALL 2020\CSCI 360\Labs\Lab3\l3\lab3_utils.py:26: RuntimeWarning: invalid value e
ncountered in power
  return np.power(
C:\Users\Hyun-\OneDrive\바탕 화면\Classes\FALL 2020\CSCI 360\Labs\Lab3\l3\lab3.py:167: RuntimeWarning: invalid value encoun
tered in long_scalars
  num += (y / dist)
```

In [3]:

```
print('SUMMARY')
print('-'*20)
print('Row -> k (1 to 30)')
print('Col -> l (-1, 1, 2, 3, 4, 5, 6, inf)')
print('-'*10)
print('Weighted')
print(inaccuracies[:,:,0])
best_accuracy = np.min(inaccuracies[:,:,0])
best_k, best_metric, best_weighting = np.where(inaccuracies == best_accuracy)
print(f"Lowest error achieved: {best_accuracy:0.4f}")
for k, l, w in zip(best_k, best_metric, best_weighting):
    print(f"\t k= {k+1} - l= {l_norms[l]} - weighting= {weighting_options[w]}]")
print('-'*10)
print('Unweighted')
print(inaccuracies[:,:,1])
best_accuracy = np.min(inaccuracies[:,:,1])
best_k, best_metric, best_weighting = np.where(inaccuracies == best_accuracy)
print(f"Lowest error achieved: {best_accuracy:0.4f}")
for k, l, w in zip(best_k, best_metric, best_weighting):
    print(f"\t k= {k+1} - l= {l_norms[l]} - weighting= {weighting_options[w]}]")
# print(inaccuracies[inaccuracies[:,:,0] != inaccuracies[:,:,1]])
```

```
SUMMARY
--------------------
Row -> k (1 to 30)
Col -> l (-1, 1, 2, 3, 4, 5, 6, inf)
----------
Weighted
[[0.61111111 0.43055556 0.47222222 0.44444444 0.47222222 0.47222222
  0.47222222 0.47222222]
 [0.65277778 0.5        0.5        0.5        0.51388889 0.51388889
  0.51388889 0.48611111]
 [0.65277778 0.44444444 0.38888889 0.36111111 0.375      0.375
  0.375      0.43055556]
 [0.65277778 0.44444444 0.38888889 0.40277778 0.40277778 0.40277778
  0.38888889 0.43055556]
 [0.72222222 0.36111111 0.375      0.36111111 0.34722222 0.34722222
  0.34722222 0.41666667]
 [0.68055556 0.38888889 0.38888889 0.375      0.375      0.375
  0.375      0.38888889]
 [0.65277778 0.38888889 0.40277778 0.34722222 0.34722222 0.34722222
  0.34722222 0.27777778]
 [0.63888889 0.40277778 0.375      0.36111111 0.36111111 0.36111111
  0.36111111 0.36111111]
 [0.65277778 0.375      0.34722222 0.30555556 0.30555556 0.30555556
  0.31944444 0.30555556]
 [0.59722222 0.36111111 0.34722222 0.31944444 0.30555556 0.30555556
  0.30555556 0.34722222]
 [0.625      0.34722222 0.36111111 0.33333333 0.33333333 0.33333333
  0.34722222 0.36111111]
 [0.59722222 0.36111111 0.34722222 0.31944444 0.30555556 0.30555556
  0.30555556 0.38888889]
 [0.56944444 0.33333333 0.33333333 0.29166667 0.30555556 0.30555556
  0.30555556 0.29166667]
 [0.56944444 0.36111111 0.31944444 0.30555556 0.31944444 0.31944444
  0.31944444 0.30555556]
 [0.55555556 0.34722222 0.30555556 0.29166667 0.29166667 0.29166667
  0.29166667 0.33333333]
 [0.58333333 0.34722222 0.30555556 0.31944444 0.30555556 0.30555556
  0.30555556 0.33333333]
 [0.55555556 0.34722222 0.29166667 0.30555556 0.30555556 0.30555556
  0.30555556 0.34722222]
 [0.58333333 0.33333333 0.31944444 0.30555556 0.29166667 0.29166667
  0.29166667 0.34722222]
 [0.56944444 0.34722222 0.27777778 0.30555556 0.30555556 0.30555556
  0.30555556 0.34722222]
 [0.58333333 0.33333333 0.31944444 0.30555556 0.30555556 0.30555556
  0.30555556 0.34722222]
 [0.52777778 0.31944444 0.29166667 0.30555556 0.30555556 0.30555556
  0.30555556 0.34722222]
 [0.54166667 0.33333333 0.31944444 0.30555556 0.31944444 0.31944444
  0.31944444 0.33333333]
 [0.52777778 0.33333333 0.29166667 0.30555556 0.30555556 0.30555556
  0.30555556 0.31944444]
 [0.54166667 0.34722222 0.27777778 0.30555556 0.30555556 0.30555556
  0.30555556 0.33333333]
 [0.54166667 0.34722222 0.29166667 0.29166667 0.29166667 0.29166667
  0.29166667 0.31944444]
 [0.54166667 0.34722222 0.27777778 0.29166667 0.29166667 0.29166667
  0.29166667 0.31944444]
 [0.56944444 0.34722222 0.27777778 0.27777778 0.27777778 0.27777778
  0.27777778 0.31944444]
 [0.52777778 0.33333333 0.27777778 0.27777778 0.27777778 0.27777778
  0.27777778 0.33333333]
 [0.52777778 0.34722222 0.26388889 0.27777778 0.27777778 0.27777778
  0.27777778 0.31944444]
 [0.52777778 0.33333333 0.29166667 0.25       0.26388889 0.26388889
  0.26388889 0.33333333]]
Lowest error achieved: 0.2500
         k= 25 - l= inf - weighting= False]
         k= 30 - l= 3 - weighting= True]
         k= 30 - l= inf - weighting= False]
----------
Unweighted
[[0.61111111 0.375      0.41666667 0.38888889 0.41666667 0.41666667
  0.41666667 0.41666667]
 [0.63888889 0.38888889 0.41666667 0.40277778 0.38888889 0.40277778
  0.38888889 0.44444444]
 [0.58333333 0.40277778 0.45833333 0.44444444 0.44444444 0.45833333
  0.44444444 0.51388889]
 [0.625      0.375      0.38888889 0.40277778 0.38888889 0.38888889
  0.38888889 0.375     ]
 [0.51388889 0.34722222 0.44444444 0.38888889 0.40277778 0.40277778
  0.40277778 0.43055556]
```

```
 [0.65277778 0.33333333 0.38888889 0.375      0.36111111 0.36111111
  0.36111111 0.23611111]
 [0.51388889 0.44444444 0.43055556 0.40277778 0.41666667 0.38888889
  0.41666667 0.29166667]
 [0.59722222 0.36111111 0.375      0.40277778 0.41666667 0.43055556
  0.43055556 0.31944444]
 [0.47222222 0.51388889 0.40277778 0.40277778 0.375      0.375
  0.40277778 0.375     ]
 [0.5        0.45833333 0.36111111 0.38888889 0.375      0.375
  0.38888889 0.36111111]
 [0.44444444 0.38888889 0.41666667 0.45833333 0.45833333 0.45833333
  0.45833333 0.40277778]
 [0.54166667 0.34722222 0.375      0.33333333 0.33333333 0.33333333
  0.33333333 0.29166667]
 [0.54166667 0.33333333 0.34722222 0.29166667 0.29166667 0.29166667
  0.29166667 0.36111111]
 [0.65277778 0.375      0.34722222 0.30555556 0.31944444 0.31944444
  0.31944444 0.41666667]
 [0.5        0.45833333 0.41666667 0.41666667 0.41666667 0.41666667
  0.41666667 0.36111111]
 [0.69444444 0.31944444 0.29166667 0.33333333 0.31944444 0.31944444
  0.31944444 0.40277778]
 [0.69444444 0.33333333 0.30555556 0.31944444 0.31944444 0.31944444
  0.31944444 0.34722222]
 [0.48611111 0.52777778 0.44444444 0.38888889 0.40277778 0.40277778
  0.40277778 0.41666667]
 [0.65277778 0.29166667 0.31944444 0.29166667 0.29166667 0.29166667
  0.29166667 0.33333333]
 [0.54166667 0.52777778 0.40277778 0.44444444 0.45833333 0.45833333
  0.45833333 0.51388889]
 [0.5        0.43055556 0.375      0.33333333 0.34722222 0.36111111
  0.34722222 0.41666667]
 [0.55555556 0.43055556 0.5        0.47222222 0.45833333 0.43055556
  0.40277778 0.56944444]
 [0.45833333 0.33333333 0.41666667 0.36111111 0.33333333 0.375
  0.33333333 0.43055556]
 [0.51388889 0.45833333 0.5        0.51388889 0.48611111 0.43055556
  0.44444444 0.40277778]
 [0.36111111 0.34722222 0.30555556 0.34722222 0.31944444 0.375
  0.33333333 0.25      ]
 [0.34722222 0.36111111 0.40277778 0.36111111 0.36111111 0.31944444
  0.36111111 0.45833333]
 [0.45833333 0.31944444 0.40277778 0.36111111 0.34722222 0.38888889
  0.38888889 0.33333333]
 [0.63888889 0.38888889 0.30555556 0.27777778 0.29166667 0.29166667
  0.30555556 0.30555556]
 [0.51388889 0.44444444 0.52777778 0.51388889 0.51388889 0.47222222
  0.5        0.44444444]
 [0.40277778 0.44444444 0.38888889 0.40277778 0.41666667 0.45833333
  0.43055556 0.25      ]]
Lowest error achieved: 0.2361
         k= 6 - l= inf - weighting= False]
```

In [ ]: