1. Nearest Neighbours and the Curse of Dimensionality

1.a.

1.a.i. expected value

We have

$$Z = |X - Y|^2 = (X - Y)^2 = X^2 - 2XY + Y^2$$

and

$$E[Z] = E[X^2] - E[2XY] + E[Y^2]$$

For a function $f(x)$ where $x \sim \text{Uniform}(a, b)$, its expected value is

$$E[f(x)] = \int_a^b f(x) \cdot \frac{1}{b-a} dx$$

We have both X and Y are in uniform distribution from 0 and 1.

$$X, Y \sim \text{Uniform}(0, 1)$$

so

$$E[f(x)] = \int_0^1 f(x) dx$$

- for X^2 and Y^2:

$$E[X^2] = E[Y^2] = \int_0^1 x^2 dx = \frac{1^3}{3} - \frac{0^3}{3} = \frac{1}{3}$$

- for X and Y themselves:

$$E[X] = E[Y] = \frac{1}{2}$$

- so for $2XY$:

$$E[2XY] = 2 \cdot \frac{1}{2} \cdot \frac{1}{2} = \frac{1}{2}$$

  Therefore,

$$E[Z] = \frac{1}{3} - \frac{1}{2} + \frac{1}{3} = \frac{1}{6}$$

  .

1.a.ii. variance

The variance of Z

$$Var(Z) = E[(Z - \mu_Z)^2]$$

where μ_Z is the mean of Z, which is actually just $E[Z]$.

$$Var(Z) = E[(Z - E[Z])^2]$$

$$= E[(Z^2 - 2ZE[Z] + E[Z]^2)]$$

$$= E[Z^2] - 2E[Z]E[Z] + E[Z]^2$$

$$= E[Z^2] - 2E[Z]^2 + E[Z]^2$$

$$= E[Z^2] - E[Z]^2$$

So now we only need to calculate $E[Z^2]$.

$$E[Z^2] = E[(X - Y)^4]$$

$$= E[X^4 - 4X^3Y + 6X^2Y^2 - 4XY^3 + Y^4]$$

$$= E[X^4] - 4E[X^3]E[Y] + 6E[X^2]E[Y^2] - 4E[X]E[Y^3] + E[Y^4]$$

- $E[X^4] = \int_0^1 x^4 dx = \frac{1}{5}$
- $E[X^3] = \int +0^1 x^3 dx = \frac{1}{4}$ (X and Y has identical distributions, so their expected values (and to any power) are same too.)

$$E[Z^2] = \frac{1}{5} - 4(\frac{1}{4})(\frac{1}{2}) + 6(\frac{1}{3})(\frac{1}{3}) - 4(\frac{1}{2})(\frac{1}{4}) + \frac{1}{5}$$

$$= \frac{1}{15}$$

Therefore,

$$Var(Z) = \frac{1}{15} - (\frac{1}{6})^2$$

$$= \frac{7}{180}$$

.

1.b.

1.b.i. expectation

$$E[R] = E[(Z_1 + Z_2 + \ldots + Z_d)]$$

$$= E[Z_1] + E[Z_2] + \ldots + E[Z_d]$$

$$= \frac{1}{6} \cdot d$$

$$= \frac{d}{6}$$

.

1.b.ii. variance

$$Var(R) = Var(Z_1 + Z_2 + \ldots + Z_d)$$

With the following property:

> if random variables A and B are independent and have variances S_A, S_B, then the random variable A+B has variance $S_A + S_B$

$$Var(R) = Var(Z_1) + Var(Z_2) + \ldots + Var(Z_d)$$

$$= d \cdot \frac{7}{180}$$

$$= \frac{7d}{180}$$

.

1.c.

- The mean is $\frac{d}{6}$
- The variance is $\frac{7d}{180}$
- The maximal possible squared Euclidean distance between two points within the d-dimensional space would be $(\sqrt{1^2 + 1^2 + \ldots})^2 = d$.
- As dimension (d) increases, although both expectation and variance increase, expectation increase a lot more than variance. So in a high dimension, the distances between points are far away and do not vary as much as they would in lower dimensions (curse of dimensionality).

2. Limiting Properties of the Nearest Neighbour Algorithm

2.a.

- X_i (X_1 to X_n) is sampled uniformly and independently from the interval [0,2]
- $y = 1$
- $Z_i = |X_i - y| = |X_i - 1| \rightarrow$ (absolute value of) linear translation of a uniform distribution X_i
  - $\rightarrow Z_i$ is a uniform distributin
- the maximal possible (Z_i) would be when X_i is either 0 or 2.
  - if $X_i = 0 \rightarrow Z_i = |0 - 1| = 1$
  - if $X_i = 2 \rightarrow Z_i = |2 - 1| = 1$
- the minimal possible (Z_i) would be when $X_i = 1 \rightarrow |1 - 1| = 0$

Therefore, Z_i is a uniform distribution over [0,1].

2.b.

- calculate $\mathbb{P}\{Z_1 > t\}$
  - Z_1 is greather than some threshold t
  - Z_1 is smaller than any other Z_i
  - $\therefore t < Z_1 < Z_2 < \ldots < Z_n$, and

$$P\{Z_1 > t\} = P\{Z_1 > t, Z_2 > t, \ldots, Z_n > t\}$$

  - $\because Z_i$ are independent and identically distributed (iid),

$$P\{Z_1 > t\} = P\{Z_1 > t\} \cdot P\{Z_2 > t\} \cdot \cdots \cdot P\{Z_n > t\}$$

  - $\because$ each Z_i is uniformly distributed over [0,1],

$$P\{Z_i <= t\} = t; P\{Z_i > t\} = 1 - t$$

  - $\therefore$

$$P\{Z_1 > t\} = (1 - t)^n$$

- calculate $\mathbb{E}[Z_1]$

$$E[Z_1] = \int_0^1 P\{Z_1 > t\}dt$$

$$= \int_0^1 (1-t)^n dt$$

$$= \frac{1}{n+1}$$

.

2.c.

- the expected value of a continuous random variable X with PDF f(x) is

$$E[X] = \int_{-\infty}^{\infty} x f(x) dx$$

- so for this question,

$$E[Z_k] = \int_0^1 t f_{z_k}(t) dt$$

$$= \int_0^1 t \cdot \frac{n!}{(k-1)!(n-k)!} t^{k-1}(1-t)^{n-k} dt$$

$$= \frac{n!}{(k-1)!(n-k)!} \int_0^1 t^k (1-t)^{n-k} dt$$

- for the integral part, we use the beta function property:

$$B(a,b) = \int_0^1 t^{a-1}(1-t)^{b-1} dt = \frac{\Gamma(a)\Gamma(b)}{\Gamma(a+b)}$$

  - the gamma function $\Gamma(n) = (n-1)!$
  - for this question, $a = k+1, b = n-k+1$
  - $\therefore$

$$B(k+1, n-k+1) = \frac{k!(n-k)!}{(n+1)!}$$

- $\therefore$

$$E[Z_k] = \frac{n!}{(k-1)!(n-k)!} \cdot \frac{k!(n-k)!}{(n+1)!}$$

$$= \frac{k}{n+1}$$

.

2.d

- as $n \to \infty$, $E[Z_k] = \frac{k}{n+1} \approx \frac{k}{n}$
- and since $\frac{k}{n} \to 0$, we get $E[Z_k] \approx 0$.
- therefore, the expected distance to the k-th nearest neighbour approximates 0 in this case.

3. Information Theory

3.a.

$$H(X) = -\sum_{x} p(x) log_2 p(x)$$

To prove that the entropy H(X) is non negative:

- for $p(x)$:
  - $p(x)$ is a probability, which lies between (0,1]
  - $\therefore p(x) > 0.$
- for $log_2p(x):
  - since $0 < p(x) <= 1$, $\lim_{x \to 0^+} log_2(x) < log_2 p(x) <= log_2 1$
  - $log_2 x$ when x approaches $0^+ \approx -\infty$
  - $log_2 1 = 0$
  - $\therefore log_2 p(x) <= 0.$
- Combining these two points, $-\sum_x p(x) log_2 p(x) <= 0$
- $\therefore H(X) >= 0.$

3.b.

$$H(X,Y) = -\sum_{x \in X} \sum_{y \in Y} p(x,y) log_2 p(x,y)$$

- if X and Y are independent, p(x,y) = p(x)p(y)
- so we can say

$$H(X,Y) = -\sum_{x \in X} \sum_{y \in Y} p(x)p(y) log_2 [p(x)p(y)]$$

$$= -\sum_{x \in X} \sum_{y \in Y} p(x)p(y) [log_2 p(x) + log_2 p(y)]$$

$$= -\sum_{x \in X} \sum_{y \in Y} p(x)p(y) log_2 p(x) - \sum_{x \in X} \sum_{y \in Y} p(x)p(y) log_2 p(y)$$

- X and Y are independent, and p(X) = p(Y) = 1
- so

$$= -\sum_{x \in X} p(x) log_2 p(x) - \sum_{y \in Y} p(y) log_2 p(y)$$

$$= H(X) + H(Y)$$

.

3.c.

- if X and Y are dependent, p(x,y) = p(x)p(y|x)
- so

$$H(X,Y) = -\sum_{x \in X} \sum_{y \in Y} p(x,y) log_2 p(x,y)$$

$$H(X,Y) = -\sum_{x\in X}\sum_{y\in Y} p(x)p(y|x)log_2[p(x)p(y|x)]$$

$$= -\sum_{x\in X}\sum_{y\in Y} p(x)p(y|x)[log_2 p(x) + log_2 p(y|x)]$$

$$= -\sum_{x\in X}\sum_{y\in Y} p(x)p(y|x)log_2 p(x) - \sum_{x\in X}\sum_{y\in Y} p(x)p(y|x)log_2 p(y|x)$$

- in the first term, $log_2 p(x)$ only depends on x, so we can simplify it to $-\sum_{x\in X} p(x)log_2 2p(x) = H(X)$
- for the second term, that is basically $H(Y|X)$ (according to slide lec02 p26)

$$\therefore H(X,Y) = H(X) + H(Y|X)$$

3.d.

If we can rewrite everything under a negative sign, that can make our life a lot easier:

$$-KL(p||q) = -\sum_x p(x)log_2\frac{p(x)}{q(x)}$$

$$= \sum_x p(x)log_2\frac{q(x)}{p(x)};$$

This can be expressed as the expectation of $log_2\frac{q(x)}{p(x)}$ with respect to p(x)

$$-KL(p||q) = E_p[log_2\frac{q(X)}{p(X)}]$$

- a log function is concave -> the associated Jensen's inequality would be

  $$log(E[X]) >= E[log(X)]$$

  so

  $$log_2(E_p[\frac{q(X)}{p(X)}]) >= E_p[log_2\frac{q(X)}{p(X)}]$$

  For $log_2(E_p[\frac{q(X)}{p(X)}])$, let's rewrite it back to $log_2\sum_x p(x)\frac{q(x)}{p(x)}$.

  $$log_2\sum_x p(x)\frac{q(x)}{p(x)} = log_2\sum_x q(x)$$

  q(x) is another distribution, so it sums to 1. Therefore we get

  $$log_2\sum_x p(x)\frac{q(x)}{p(x)} = log_2 1 = 0$$

  Put it back into the Jensen's inequality,

  $$0 >= E_p[log_2\frac{q(X)}{p(X)}] = -KL(p||q)$$

  $$\therefore KL(p||q) >= 0$$

  .

3.e.

According to 3.c. H(Y|X)=H(X,Y)-H(X), so

$$I(Y;X) = H(Y) - H(Y|X) = H(Y) - H(X,Y) + H(X)$$

$$= -H(X,Y) + H(X) + H(Y)$$

$$= \sum_{x \in X} \sum_{y \in Y} p(x,y) log_2 p(x,y) - \sum_{x \in X} p(x) log_2 p(x) - \sum_{y \in Y} p(y) log_2 p(y))$$

While for KL,

$$KL(p(x,y)||p(x)p(y)) = \sum_{x,y} p(x,y) log_2 \frac{p(x,y)}{p(x)p(y)}$$

$$= \sum_{x,y} p(x,y) log_2 p(x,y) - \sum_{x,y} p(x,y) log_2 p(x) - \sum_{x,y} p(x,y) log_2 p(y)$$

$$= \sum_{x,y} p(x,y) log_2 p(x,y) - \sum_{x} p(x) log_2 p(x) - \sum_{y} p(y) log_2 p(y)$$

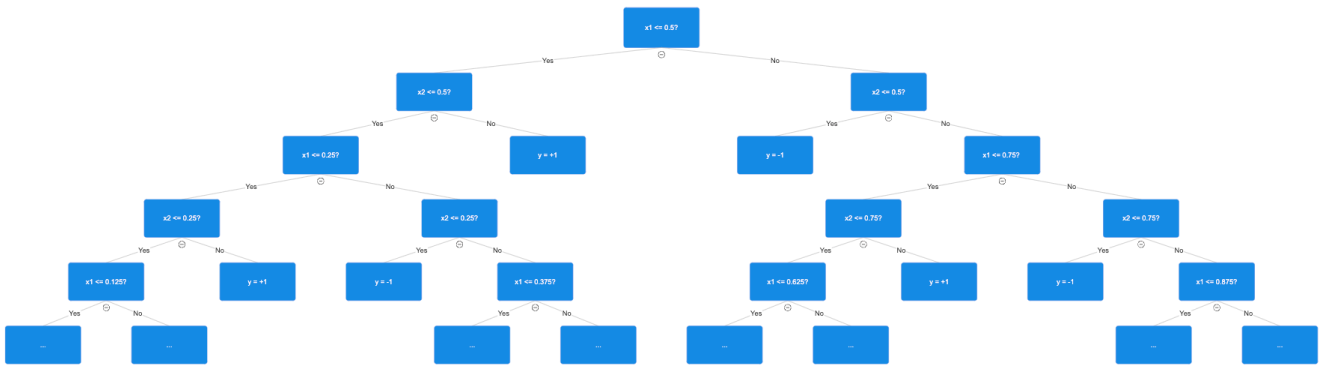
And we can see that the information gain is equal to the KL divergence.

4. Approximation Error in Decision Trees

4.a.

- the function f^* takes the value:
  - y = +1 for points where $x_2 - x_1 > 0$
  - y = -1 for points where $x_2 - x_1 < 0$
  - the decision boundary is $x_2 - x_1 = 0$
  - (let's not consider the points on the boundary now.)
- we see that this boundary is diagonal and takes in both parameters x_1 and x_2, while a decision tree only takes in one parameter at a time.
- If we have a decision tree f_d with depth d approaches infinity, we would be able to represent f^* because it can make arbitrary splits to approximate the diagonal line.
- However, with a finite d, the decision tree cannot represent f^* exactly. Instead, it approximates the diagonal boundary with a staircase-like split, which becomes closer to the true boundary as d increases, but never perfectly represents it.

4.b.



4.c.

- $e_2 = 0.25$
- $e_4 = 0.0625$

4.d.

$$e_d = \frac{1}{d^2}$$

where $d = 2k$ with $k \in \mathbb{N}$.

- Justification: An even d (d%2==0) indicates that the number of splits along x_1 is equal to the number of splits along x_2. Let's call it "a set of splits". With each set of splits, we obtain half of the areas where we are certain about the y values (i.e. the upper left quadrant has y = +1 whereas the lower right quadrant has y = -1). The other two quadrants (lower left and upper right) are regions where we are still unsure and need further separation. For each of these two quadrants, regardless of how we assign values to them, half of the quadrant will be classified erroneously. As a result, for each set of splits, half of the area in 2 out of 4 quadrants returns error--this is equivalent to 1/4 of the total area affected by the set of splits.
  - if d = 2, e = 1/4
  - d = 4, e = 1/4 * 1/4 = 1/16
  - therefore we can infer that for every even d, $e_d = \frac{1}{d^2}$.

4.e.

This tells us that as d increases, error decreases. If d approaches infinity, the error approaches 0, and we will eventually achieve a (nearly) perfect seperation.

5. Decision Trees and K-Nearest Neighbour

In [ ]:
```
import sys
import time
import matplotlib.pyplot as plt

%matplotlib inline
import numpy as np
import pandas as pd
from IPython.display import HTML

sys.path.append("code/.")

import mglearn
from IPython.display import display

from sklearn.pipeline import Pipeline, make_pipeline
from sklearn.compose import ColumnTransformer, make_column_transformer
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.model_selection import cross_val_score, cross_validate, train_test_split
from sklearn.metrics import accuracy_score
from sklearn.neighbors import KNeighborsClassifier, KNeighborsRegressor
from sklearn.tree import DecisionTreeClassifier, plot_tree

pd.set_option("display.max_colwidth", 200)
```

```
Intel MKL WARNING: Support of Intel(R) Streaming SIMD Extensions 4.2 (Intel(R) SSE4.2) enabled only
processors has been deprecated. Intel oneAPI Math Kernel Library 2025.0 will require Intel(R) Advanc
ed Vector Extensions (Intel(R) AVX) instructions.
Intel MKL WARNING: Support of Intel(R) Streaming SIMD Extensions 4.2 (Intel(R) SSE4.2) enabled only
processors has been deprecated. Intel oneAPI Math Kernel Library 2025.0 will require Intel(R) Advanc
ed Vector Extensions (Intel(R) AVX) instructions.
```

5.a. preprocessing

for **function** "load_data":

- load data
- vectorize them
- add target (exist or dne)
  - for colname use something that won't be a word in any text e.g. 'Y'
- combine two dfs
- train test split (70% train, 15% val, 15% test)

In [ ]:
```
# df_exists = pd.read_csv('h1_data/exists_climate.csv')
# df_dne = pd.read_csv('h1_data/DNE_climate.csv')
# print(df_exists.shape,df_dne.shape)
# df_exists.head(),df_dne.head()
```

In [ ]:
```
def read_and_vec(path):
    # read csv
    #only one col i.e. text; header is 'tweet'
    df = pd.read_csv(path, header=0)

    # count vec
    vec = CountVectorizer()
    X_counts = vec.fit_transform(df['tweet'])
    # bag of words
    bow_df = pd.DataFrame(
        X_counts.toarray(),
        columns = vec.get_feature_names_out(),
        index = df['tweet']
    )
    return bow_df
```

In [ ]:
```
def load_data():
    df_exists = read_and_vec('h1_data/exists_climate.csv')
    df_dne = read_and_vec('h1_data/DNE_climate.csv')

    # add target under column 'Y'
    df_exists['Y']=1
    df_dne['Y']=0

    # combine two dfs
    df_with_target = pd.concat([df_exists, df_dne], axis=0).fillna(0)

    # train test split
    train_df, temp_df = train_test_split(df_with_target, test_size=0.30, random_state=42)
    val_df, test_df = train_test_split(temp_df, test_size=0.50, random_state=42)


    return train_df, val_df, test_df
```

In [ ]:
```
train_df, val_df, test_df = load_data()

# just checking
train_df.sample(5, random_state=69)[['Y']]
```

Out[ ]:

tweet	Y
Rebecca Solnit shares The Good News About the Very Bad News About Climate Change: http://bitly/9Gg5Ke TomDispatch	1
BarackObama - dont tell me- global warming wi be a topic-ha ha	0
Need for precise information on climate change: Shyam Saran - dnaindiacom http://retwtme/1LVCt via tnewsindia	1
algore you are such a cocksucker global warming is fake you should be put in prison stop trying to push your lies on America	0
33 US Military Generals Admirals: "Climate Change is Threatening http://bitly/bbFBhD	1

5.b. decision tree classifier

for **function** "select_tree_model":

- train dt classifier
  - 5 different *sensible* values of max_depth
  - 2 different split criteria (information gain & gini coefficient)
- eval each one on val set
  - **write val code yourself**
- print the resulting accuracies of each model

In [ ]:
```
def get_accuracy(pred, target):
    '''
    compare pred (np.1darray) to target (np.1darray) and calculate the accuracy
    '''
    return np.sum(pred==target)/len(pred)
```

In [ ]:
```
X_train = train_df.drop(columns=['Y'])
y_train = train_df['Y']
X_val = val_df.drop(columns=['Y'])
y_val = val_df['Y']
X_test = test_df.drop(columns=['Y'])
y_test = test_df['Y']

max_depth = [10,50,100,250,500,1000]
criteria = ['information_gain','gini_coefficient']

def select_tree_model(X_train, y_train, X_test, y_test, max_depth, criterion):
    if criterion == 'information_gain':
        c = 'entropy'
    else:
        c = 'gini'
    clf = DecisionTreeClassifier(max_depth = max_depth, criterion = c, random_state = 42)
    clf.fit(X_train, y_train)
    pred = clf.predict(X_test)
    accuracy = get_accuracy(pred, y_test)
    return clf,accuracy
```

In [ ]:
```
highest_accuracy = 0
best_max_depth = 0
best_criterion = ''

for i in max_depth:
    for j in criteria:
        clf,accuracy = select_tree_model(X_train, y_train, X_val, y_val, i, j)
        print(f'max_depth = {i}, criterion = {j}, accuracy = {round(accuracy,3)}')
        if accuracy>highest_accuracy:
            best_max_depth, best_criterion, highest_accuracy = i,j,accuracy
```

```
max_depth = 10, criterion = information_gain, accuracy = 0.761
max_depth = 10, criterion = gini_coefficient, accuracy = 0.761
max_depth = 50, criterion = information_gain, accuracy = 0.787
max_depth = 50, criterion = gini_coefficient, accuracy = 0.787
max_depth = 100, criterion = information_gain, accuracy = 0.78
max_depth = 100, criterion = gini_coefficient, accuracy = 0.782
max_depth = 250, criterion = information_gain, accuracy = 0.78
max_depth = 250, criterion = gini_coefficient, accuracy = 0.788
max_depth = 500, criterion = information_gain, accuracy = 0.78
max_depth = 500, criterion = gini_coefficient, accuracy = 0.788
max_depth = 1000, criterion = information_gain, accuracy = 0.78
max_depth = 1000, criterion = gini_coefficient, accuracy = 0.788
```

5.c. test & tree plot

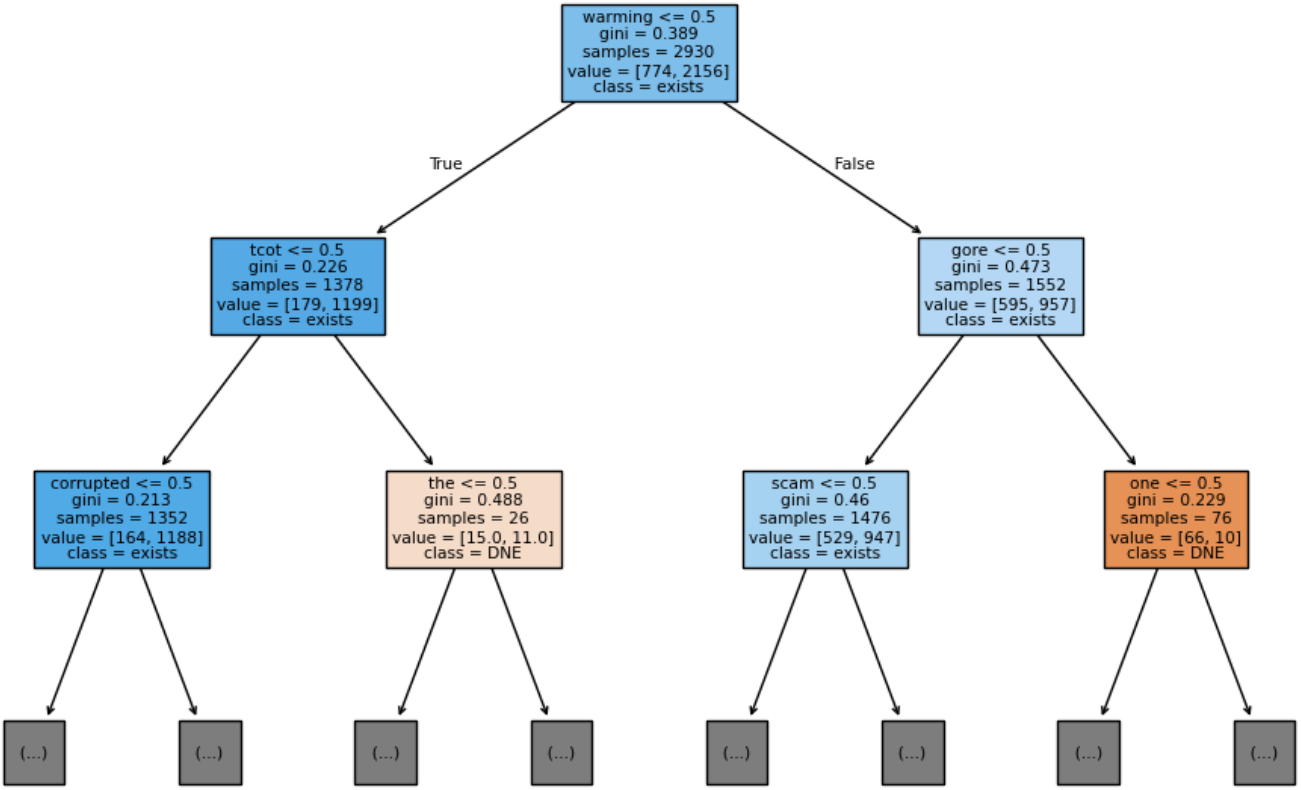
- choose the hyperparameters with the highest validation accuracy
- report the accuracy on test set
- visualize the first two layers of the tree

In [ ]:
```
clf_best, test_accuracy = select_tree_model(X_train,y_train,X_test,y_test,best_max_depth,best_criter
print(f'the hyperparameters associated with the best validation accuray is: \n'
      f'max_depth = {best_max_depth}; criterion = {best_criterion}.\n'
      f'with these hyperparameters, the test accuracy is {round(test_accuracy,3)}.')
```

```
the hyperparameters associated with the best validation accuray is:
max_depth = 250; criterion = gini_coefficient.
with these hyperparameters, the test accuracy is 0.824.
```

In [ ]:
```
# confirm the order for class_names
# if it's [0,1] then class_names = ['DNE','exists']
if np.array_equal(y_train.unique(), [0,1]):
    class_names = ['DNE','exists']
else:
    class_names = ['exists','DNE']

# np.array_equal(y_train.unique(), [0,1])
```

In [ ]:
```
plt.figure(figsize=(12,8))
plot_tree(clf, max_depth=2, filled=True, feature_names=X_train.columns.tolist(), class_names=class_r
plt.show()
```



5.d. compute information gain

- compute I(Y,x_i) (for a split on the **training data**)
  - information gain I(Y,x_i) = H(Y) - H(Y|x_i)
  - entropy H(X) = $-\sum_{x \in X} p(x) log_2 p(x)$
- report the outputs of this function for the **topmost split from 5.c.**
- as well as other keywords
- only consider whether the keyword exists

In [ ]:
```python
def entropy(x):
    # x is a 1darray of targets
    # calc probs i.e. the proportion of each class
    # np.bincount always start from the smallest class i.e. 0 for class=[0,1]
    probs = np.bincount(x)/len(x)
    # np.bincount may include 0s if the bin has no count
    # should get rid of 0s by using p > 0
    return -np.sum([p * np.log2(p) for p in probs if p > 0])
```

In [ ]:
```python
def compute_information_gain(X_train,y_train,xi):
    '''
    X_trian is a df that we are about to split at the root node
    y_train is Y, which is an array of targets of that df (X_train)
    xi is the keyword that we are using for splitting
    '''
    # root entropy = H(Y)
    root_entropy = entropy(y_train)

    # leafs entropy = H(Y|x_i)
    # = H(Y|x_i = True) * P(x_i = True) + H(Y|x_i = False) * P(x_i = False)
    mask_xi_true = X_train[xi] > 0
    mask_xi_false = X_train[xi]==0
```

```
    y_true = y_train[mask_xi_true]
    y_false = y_train[mask_xi_false]
    y_true_prob = len(y_true)/len(y_train)
    y_false_prob = 1-y_true_prob

    leafs_entropy = entropy(y_true) * y_true_prob + entropy(y_false) * y_false_prob

    information_gain = root_entropy - leafs_entropy
    return root_entropy, leafs_entropy, information_gain
```

In [ ]:
```
# for the topmost split from the previous part
root_entropy, leafs_entropy, information_gain = compute_information_gain(X_train,y_train,'warming')
print(f"from the plot in 5.c. the top split has information gain = {round(information_gain, 3)}.")
```

```
from the plot in 5.c. the top split has information gain = 0.062.
```

In [ ]:
```
# to double check
clf_check,accuracy_check = select_tree_model(X_train, y_train, X_val, y_val, 1, 'information_gain')
tree_check = clf_check.tree_
root_entropy_check = tree_check.impurity[0]
left_leaf_entropy_check = tree_check.impurity[1]
right_leaf_entropy_check = tree_check.impurity[2]
left_leaf_proportion_check = tree_check.n_node_samples[1] / tree_check.n_node_samples[0]
right_leaf_proportion_check = tree_check.n_node_samples[2] / tree_check.n_node_samples[0]
leafs_entropy_check = left_leaf_entropy_check * left_leaf_proportion_check + right_leaf_entropy_chec
information_gain_check = root_entropy_check - leafs_entropy_check

function_above_vals = {
      'root_entropy': root_entropy,
      'leafs_entropy': leafs_entropy,
      'information_gain': information_gain
}

check_from_tree_vals = {
      'root_entropy': root_entropy_check,
      'leafs_entropy': leafs_entropy_check,
      'information_gain': information_gain_check
}

df_check = pd.DataFrame(
      [function_above_vals, check_from_tree_vals],
      index=['function_above', 'check_from_tree']
)

print(df_check)
# plt.figure(figsize=(6,4))
# plot_tree(clf_check, max_depth=2, filled=True, feature_names=X_train.columns.tolist(), class_names
# plt.show()
```

```
                root_entropy  leafs_entropy  information_gain
function_above      0.832965       0.770749          0.062216
check_from_tree     0.832965       0.770749          0.062216
```

In [ ]:
```
# for other several keywords

keywords = ['president','scientists','conspiracy','school','hot']
for k in keywords:
    root_entropy, leafs_entropy, information_gain = compute_information_gain(X_train,y_train,k)
    print(f'keyword: {k}, information gain: {information_gain}')
```

```
keyword: president, information gain: 0.0009648045919823378
keyword: scientists, information gain: 2.872997182856718e-05
keyword: conspiracy, information gain: 0.006578124860050605
keyword: school, information gain: 0.00010732191986118078
keyword: hot, information gain: 0.0014530708874183063
```

5.e.

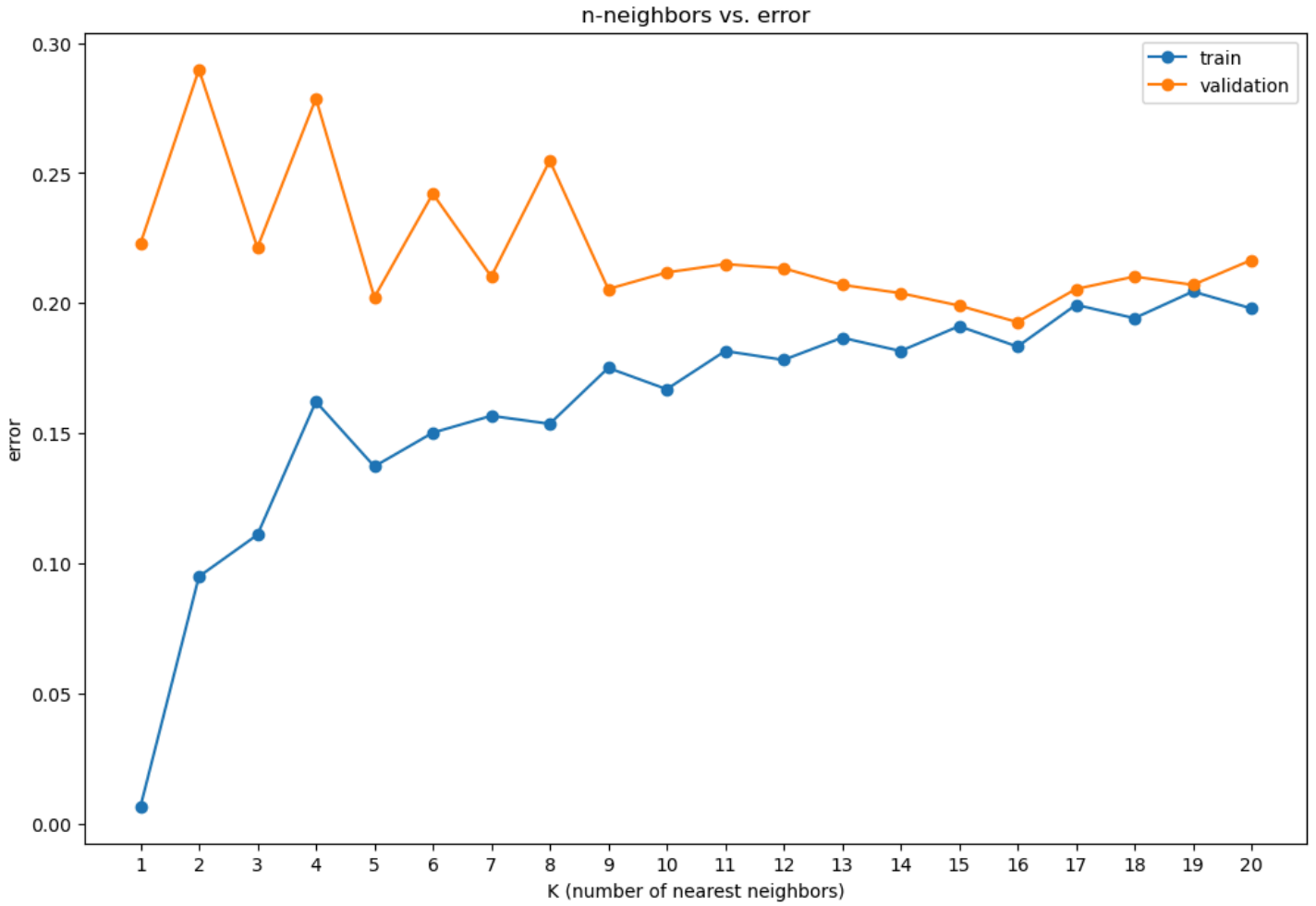
write a function **select_knn_model**:

- use a range of k between 1~20
- compute training and val errors
- generate a graph for k vs. training error
- report the generated graph in your report
- choose the model with the best validation accuracy and report its accuracy on the test data

In [ ]:
```
def select_knn_model(X_train,y_train,X_test,y_test,k):
    # for some reasons it raises the following error when using original dataframes and series
    # AttributeError: 'Flags' object has no attribute 'c_contiguous'

    # change to np.array instead
    X_train = np.array(X_train)
    y_train = np.array(y_train)
    X_test = np.array(X_test)
    y_test = np.array(y_test)

    knn = KNeighborsClassifier(n_neighbors=k)
    knn.fit(X_train, y_train)

    pred_training = knn.predict(X_train)
    training_accuracy = accuracy_score(y_train, pred_training)
    training_error = 1 - training_accuracy
    pred_testing = knn.predict(X_test)
    test_accuracy = accuracy_score(y_test, pred_testing)
    test_error = 1 - test_accuracy

    return training_error, test_error
```

In [ ]:
```
# training and validation errors in k range of 1~20
ks = np.arange(1,21,1)
training_errors = []
val_errors = []
for k in ks:
    training_error, val_error = select_knn_model(X_train,y_train,X_val,y_val,k)
    training_errors.append(training_error)
    val_errors.append(val_error)
```

In [ ]:
```
plt.figure(figsize=(12,8))
plt.plot(ks, training_errors, marker='o', label='train')
plt.plot(ks, val_errors, marker='o', label='validation')
plt.xticks(ks)
plt.xlabel('K (number of nearest neighbors)')
plt.ylabel('error')
plt.title('n-neighbors vs. error')
plt.legend()
plt.show()
```



In [ ]:
```
# find the k that has the best validation accuracy (lowest error)
# need to +1 to the index
best_k = np.argmin(val_errors)+1
print(f'best k is {best_k}')
training_error, test_error = select_knn_model(X_train,y_train,X_test,y_test,best_k)
# accuracy = 1 - error
print(f'accuracy on the test set with the best k={best_k} is {round(1-test_error, 3)}')
```

```
best k is 16
accuracy on the test set with the best k=16 is 0.776
```

In [ ]: