# # K-Nearest Neighbor (KNN) Project

*Authors: Carleton Smith; W.P.G.Peterson*

## ## Project Guide



## ## Project Overview

### EXPECTED TIME: 3 HRS

This project has 3 parts:

- Part 1: Familiarize yourself with the problem and data
- Part 2: Code a KNN Classifier from scratch, evaluate performance, and compare to Scikit-Learn's implementation
- Part 3: Interpret results and explain findings.

This will include:

- Answering simple questions regarding the data
- Manipulating multiple DataFrames
- Coding functions to:
  - Calculate Euclidean distance
  - Calculate distance between many pairs of points
  - Implement a majority voting system
  - Combine the above to create a custom KNN algorithm
- Use `KNeighborsClassifier` in `sklearn`

**Motivation**: KNN is a reasonably simple algorithm that is easy to grasp and can be very effective.

**Objectives**: By the end of this assignment, you will:

- Have a firm understanding of the KNN algorithm
- Have practiced running through the data science workflow to solve a problem
- Will demonstate how to translate a mathematical algorithm into effective code
- Understand common pitfalls when working with distances

**Problem**: Classify the type of activity a person is performing based on measurements collected from a smartphone. The activities include:

- Walking
- Walking_Upstairs
- Walking_Downstairs
- Sitting
- Standing
- Laying

**Dataset**: *Human Activity Recognition Using Smartphones Data Set* from the UC Irvine Machine Learning Repositiory.

Dataset description as provided in the original authors:

```
The experiments have been carried out with a group of 30 volunteers within an age bracket of 19-48 years. Each person performed six activities (WALKING, WALKING_UPSTAIRS, WALKING_DOWNSTA

The sensor signals (accelerometer and gyroscope) were pre-processed by applying noise filters and then sampled in fixed-width sliding windows of 2.56 sec and 50% overlap (128 readings/wi

For each record it is provided:
======================================

- Triaxial acceleration from the accelerometer (total acceleration) and the estimated body acceleration.
- Triaxial Angular velocity from the gyroscope.
- A 561-feature vector with time and frequency domain variables.
- Its activity label.
- An identifier of the subject who carried out the experiment.
```

Please see the Data Folder to explore the data files further.

## ## Part 1: Acquire, Explore, and Preprocess Data

### Import Libraries and Data

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
%matplotlib inline
```

### Read in the training data

```
FEATURE_NAMES = '../resource/asnlib/publicdata/features.txt'
TRAIN_DATA = '../resource/asnlib/publicdata/X_train.txt'
TRAIN_LABELS = '../resource/asnlib/publicdata/y_train.txt'
```

```
# read feature names
feats = pd.read_table(FEATURE_NAMES, sep='\n', header=None)

# read in training data
har_train = pd.read_table(TRAIN_DATA, sep='\s+', header=None)

# read in training labels
har_train_labels = pd.read_table(TRAIN_LABELS, sep='\n', header=None, names=["label"], squeeze = True)
```

## Explore the Data

**Print out the first five rows of the training data ( `har_train` ) -- does everything look okay?**

```
har_train.head()
```

### Question 1

How many rows and columns are in `har_train` ?

```
### GRADED

### Find out how many rows and columns are in har_train
### Assign the tuple of (<rows>, <cols>) to ans1
### For your reference you may also want to print out the nubmer of rows and columns

### YOUR ANSWER BELOW

ans1 = (7352, 561)
```

**Print the first 5 rows of `feats` - the DataFrame of feature names.**

```
feats.head()
```

**Change the names of columns via the `.columns` attribute**

```
har_train.columns = feats.iloc[:,0]
har_train.head()
```

### Question 2

Are there missing values?

```
### GRADED
### How many "null" values are in the `har_train` dataframe
### Assign int answer to ans1

### YOUR ANSWER BELOW

ans1 = 0
```

**Plot the correlation plot of the first 20 features ( `har_train.iloc[:, :20])` with `seaborn` .**

- Seaborn: https://seaborn.pydata.org/examples/many_pairwise_correlations.html

```
# seaborn
first_twenty = har_train.iloc[:, :20] # pull out first 20 feats
corr = first_twenty.corr()  # compute correlation matrix
mask = np.zeros_like(corr, dtype=np.bool)  # make mask
mask[np.triu_indices_from(mask)] = True  # mask the upper triangle

fig, ax = plt.subplots(figsize=(11, 9))  # create a figure and a subplot
cmap = sns.diverging_palette(220, 10, as_cmap=True)  # custom color map
sns.heatmap(
    corr,
    mask=mask,
    cmap=cmap,
    center=0,
    linewidth=0.5,
    cbar_kws={'shrink': 0.5}
);
```

### Question 3

```
### GRADED
### In looking at the graphic above:
### True or False:
### In these first 20 features, some are  highly correlated. e.g. With correlation >0.5 or < -0.5?
### Assign boolean answer to ans1

### YOUR ANSWER BELOW

ans1 = True
```

EDA, as was performed (briefly) above is used to develop an idea of potential problems with data. Particularly with modeling, looking for Null / impossible values, and correlated features are important steps to:

1. See if any features will not be useful in models becuase of null values.
2. See if any model assumptions are violated by correlated features (such as in linear / logistic regression)

**Switching to the target variable ( `har_train_labels` ).**

### Question 4

Investigate class sizes - are they unbalanced?

```
### GRADED
### How many times does the majority class appear in our data?
### How many times does the minority class appear in our data?
### Assign int to ans_maj and ans_min
### YOUR ANSWER BELOW

ans_maj = 1407
ans_min = 986
```

While the activities are not perfectly represented equally, its fairly close.
A large imbalance in the distribution of the target variable categories can cause machine learning algorithms to train themselves well with the majority class, and perform poorly on the minority class.

**Use the `.describe()` method along with `.groupby()` method to compare the statistics within each activity.**

```
# concatenate the target variable
# give target and observations conventional names
y = har_train_labels
X = har_train

data = pd.concat([X, y], axis=1)
data.shape
```

```
# group by the 'label' and show descriptive stats
data.groupby('label').agg(['count', 'mean','std','min','max','median']).T.head(20)
```

**Summary**:
EDA should be tailored to your specific problem - to help develop and understanding of the data for a particular purpose. This is time consuming process when the data are large with many features. It's subject matter experts can guide initial explorations.

The above are examples of just a few  of the things EDA should include when starting a project.

With a feel for the data, now  we will aside a "test" data-set that will allow  us to evaluate out models.

`train_test_split` from `sklearn.model_selection` module provides an easy way to do this.

Setting `test_size=.3` and `random_state=24` .

```
from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=.3, random_state=24)
```

### Question 5

```
### GRADED
### `test_size = .3` means:

### 'a') the final 30% of the data is held out for the test data
### 'b') any observations with ".3" are held out
### 'c') a random 30% of the data are held out
### 'd') a random 70% of the data are held out
### assign character associated with your choice to ans1 as a string

### YOUR ANSWER BELOW

ans1 = 'c'
```

## Part 2: Code KNN

Note: The following was adapted from example 2.1.2 in Chapter 2 of *Machine Learning in Action* by Peter Harrington.

Before fitting a KNN model using built in functionality in the `sklearn` package, we will code our own version of KNN.

Given a value (with our movement-data, this "value" is better thought of as a vector of values) to be classified, KNN calculates the distance between that value and all other values in the training data-set. Then, the "`k`" nearest neighbors are polled as to their "label", and the given value is predicted to be of that majority value.

Thus we need to:

**Create a function that accepts the following parameters:**

- A single data point to be classified (`input_vector`)
- Training data (`X_train`)
- Labels for training data
- Value for k (some positive integer)
- Optional: Similarity Metric (Euclidean or Cosine)- This exercise will use Euclidean for simplicity.

**Function Signature:**

`def my_knn(input_vector, X_train, y_train, k, [metric])`

**Pseudo Code:**

```
for every point in our dataset:
    calculate the distance between the current point and input_vector
    sort the distances in increasing order
    take k items with lowest disances to input_vector
    find the majority class among these items
    return the majority class label from the k closest neighbors
```

**Return:**

- The prediction for `input_vector`

## Question 6

```
### GRADED
### This exercise will use Euclidean distances.
### Please find the Euclidean distance between the points (1,2,3,-4,6) and (10,2,32,-2,0)
### Assign distance as number to ans1
### YOUR ANSWER BELOW

p1 = (1,2,3,-4,6)
p2 = (10,2,32,-2,0)

dist = 0

for a, b in zip(p1,p2):
    dist += (a-b)**2

ans1 = dist ** .5
```

## Question 7

```
### GRADED
### Code a function called "euclid_dist"

### ACCEPT two inputs, points, represented as tuples in the format, (a1, b1,...n1), (a2, b2, ...n2).
### RETURN the euclidean distance

### Remember: "**" is the python operator for exponents.

### YOUR ANSWER BELOW

def euclid_dist(p1, p2):
    """
    Calculate the Euclidian Distance between two points

    Positional Arguments:
        p1 -- A tuple of n numbers
        p2 -- A tuple of n numbers

    Example:
        p1 = (5,5)
        p2 = (0,0)
        p3 = (5,6,7,8,9,10)
        p4 = (1,2,3,4,5,6)
        print(euclid_dist(p1,p2)) #--> 7.0710678118654755
        print(euclid_dist(p3,p4)) #--> 9.797958971132712
    """

    # Start with 0 distance
    dist = 0

    # For all pairs of values in two points,
    # Find difference and square
    for a, b in zip(p1,p2):
        dist += (a-b)**2

    # Take Square Root
    return dist**.5
```

## Distances with `numpy`

The above exercise is a simple check for understanding. However, in our eventual KNN function we will use `numpy` to more efficiently calculate distances with the following code : `np.linalg.norm(np.array(p1)-np.array(p2))`.

Thankfully because `Pandas` uses `Numpy` "under the hood" we will not have to cast to numpy arrays with `np.arry()` instead it will look like: `np.linalg.norm(row1 - row1)`

Now that we can easily calculate the distances between any two points, we can start to build our function.

### Question 8

```
### GRADED
### Code a function called "all_distances"
### ACCEPT two inputs:
### An observation from a data set.  e.g: har_train.iloc[50,:]
### The full data set. e.g. har_train.

### Create a <list> or numpy array of distances between:
### ### that single point, and all points in the full dataset

### RETURN the list of distances SORTED from smallest to largest.

### Notes:
### Use `np.linalg.norm()`, as described in above cell.
### The smallest distance should be 0.

### YOUR ANSWER BELOW


def all_distances(test_point, data_set):
    """
    Find and return a list of distances between the "test_point"
    and all the points in "data_set", sorted from smallest to largest.

    Positional Arguments:
        test_point -- a Pandas Series corresponding to a row in "data_set"
        data_set -- a Pandas DataFrame

    Example:
        test_point = har_train.iloc[50,:]
        data_set = har_train

        print(all_distances(test_point, data_set)[:5])
        #--> [0.0, 2.7970187358249854, 2.922792670143521, 2.966555149052483, 3.033982453218797]

    """

    # Take difference
    diff = test_point - data_set

    # Find distance
    dists = np.apply_along_axis(np.linalg.norm, 1, diff )

    # Sort
    dists = np.sort(dists)

    return dists
```

### Question 9

Returning the value of a point and the label associated with that point:

```
### GRADED
### Code a function called "labels_of_smallest"
### ACCEPT three inputs:
### 1&2: numpy arrays, corresponding to 1: a numeric column and 2: a label column.
### ### The i-th member of the numeric column corresponds to the i-th member of the label column
### 3: an integer (>0); n.

### RETURN a list (or numpy array) of the n labels corresponding to
### ### the n smallest values in the numeric column.
### NOTE: Make sure the order of labels corresponds to the order of values.

### Hint: The labels are found in har_train_labels or y
### Hint: `pd.concat()` might be useful for this or subsequent exercisces
### YOUR ANSWER BELOW

def labels_of_smallest(numeric, labels, n):

    """
    Return the n labels corresponding to the n smallest values in the "numeric"
    numpy array.

    Positional Arguments:
        numeric -- a numpy array of numbers
        labels -- a numpy array of labels (string or numeric)
            corresponding to the values in "numeric"
        n -- a positive integer

    Example:
        numeric = np.array([7,6,5,4,3,2,1])
        labels = np.array(["a","a","b","b","b","a","a"])
        n = 6

        print(labels_of_smallest(numeric, labels, n))
        #--> np.array(['a', 'a', 'b', 'b', 'b', 'a'])
    """

    # Create a df of the two arrays (to simplify sorting)
    con = np.concatenate((numeric.reshape(-1,1), labels.reshape(-1,1)), axis = 1)
    df = pd.DataFrame(con, columns = ["num","lab"])

    # Sort
    df = df.sort_values(by = 'num')

    # Return the top "n" values
    return df['lab'].head(n).values
```

### Question 10:

Voting.
Hint: look at Counter and `.most_common()`

```
### GRADED
from collections import Counter
### Build a function called "label_voting"
### ACCEPT a non-empty numpy array of labels as input
### RETURN the value that appears most frequently in that array
### In the case of of a tie, RETURN the value in the tie that appears first in the array


### YOUR ANSWER BELOW

def label_voting(labels):
    """
    Given a numpy array of labels. Return the label that appears most frequently
    If there is a tie for most frequent, return the label that appears first.

    Positional Argument:
        labels -- a numpy array of labels

    Example:
        lab1 = np.array([1,2,2,3,3])
        lab2 = np.array(["a","a","b","b","b"])

        print(label_voting(lab1)) #--> 2
        print(label_voting(lab2)) #--> "b"

    """


    # List methods used in this function, recast labels as list
    labels = list(labels)
    # instantiate counter, find most common, returns tuples
    c = Counter(labels).most_common()

    # If only one value present, return that value
    if len(c) == 1:
        return c[0][0]
    # IF first has majority, return first
    if c[0][1] > c[1][1]:
        return c[0][0]

    # Otherwise, check to see which comes first in list
    else:
        top_votes = c[0][1]
        #print(top_votes)
        poss = []
        for t in c:
            if t[1] == top_votes:
                poss.append(t[0])
        idx = dict()
       # print(poss)
        for p in poss:
            idx[labels.index(p)] = p
        #print(idx)
        return labels[sorted(idx.keys())[0]]

##### ALTERNATE FUNCTION

def alt(labels):

    # Empty Dictionary
    count = {}

    # Save length of labels
    total = len(labels)

    # Go through labels, add entry to dictionary for each with:
    # count and first occurrence
    for i, l in enumerate(labels):

        if l in count:
            count[l]['count'] +=1
        else:
            count[l] = {'count':1, 'first':total -i}

    # Create DataFrame from dict, sort, and return top result
    df = pd.DataFrame.from_dict(count).T
    df = df.sort_values(by = ['count','first'], ascending = False)

    # Ret
    return df.index[0]
```

### Question 11

Time to put everything together.

Question 6/7/8 involved calculating distances.
Question 9 involved sorting and returning n labels.
Question 10 counted "votes."

The next question asks for a KNN modeling function:

Given four inputs:

1. a single value from X_test (created above in our `test_train_split`)
2. X_train
3. y_train (labels)
4. n - the number of nearest neighbors to poll in making predictions.

Create a function that:

1. Calculates the Euclidean distance betw een that X_test-point and every point in X_train
2. Finds the labels from the "n" nearest neighbors (ordered from closest to furthest)
3. Returns a prediction according to the voting rules outlined above (simple majority - tie goes to closest/first)

Assign to "`custom_KNN`"

```
### GRADED
### Follow directions above
### YOUR ANSWER BELOW

def custom_KNN( point, X_train, y_train, n):
    """
    Predict the label for a single point, given training data and a specified
    "n" number of neighbors.

    Positional Arguments:
        point -- a pandas Series corresponding to an observation of a point with
             unknown label.
        x_train -- a pandas DataFrame corresponding to the measurements
            of points in a dataset. Assume all values are numeric, and
            observations are in the rows; features in the columns
        y_train -- a pandas Series corresponding to the labels for the observations
            in x_train

    Example:
        point = pd.Series([1,2])
        X_train = pd.DataFrame([[1,2],[3,4],[5,6]])
        y_train = pd.Series(["a","a","b"])
        n = 2
        print(custom_KNN(point, X_train, y_train, n)) #--> 'a'
    """

    # Helper Function for vote counting
    def countVotes(l):
        c = Counter(l).most_common()
        if len(c) == 1:
            return c[0][0]
        if c[0][1] > c[1][1]:
            return c[0][0]
        else:
            top_votes = c[0][1]
            #print(top_votes)
            poss = []
            for t in c:
                if t[1] == top_votes:
                    poss.append(t[0])
            idx = dict()
           # print(poss)
            for p in poss:
                idx[l.index(p)] = p
            #print(idx)
            return l[sorted(idx.keys())[0]]

    # Take difference
    diff = point - X_train

    # Find distance
    dists = np.apply_along_axis(np.linalg.norm, 1, diff )

    # Create df of distances; re-index to original data
    df = pd.DataFrame(dists)
    df.index = X_train.index

    # Add labels, column names.
    df = pd.concat([df, y_train], axis = 1)

    df.columns = ["dist","label"]

    # Take top votes, and count
    votes = list(df.sort_values("dist").head(n)['label'])
    return countVotes(votes)
```

You should now have a functioning KNN classifier assigned to the function `customKNN`.

Let's now see how good our classifier is using `n` = 5.

Be w arned, the below cell may or may not complete runing on Vocareum due to processing constraints. (The cell took 12.9s on my machine, using my fairly efficient function, a less efficient function took ~5.5mins).

**FOR FASTER GRADING, TRY COMMENTING OUT THE CELL BELOW**

```
%%time

# Create New tts
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=.3, random_state=24)

print("Total 'test' observations:", len(X_test))
print("Classifying every point in X_test would take too long - classify the first 200")
custom_preds = []
for i, idx in enumerate(X_test.index[:200]):
    if i % 100 == 0: print(i)
    pred = custom_KNN(X_test.loc[idx,:], X_train, y_train, 5)
    custom_preds.append(pred)
```

## KNN in Sklearn

While useful to see exactly how predictions are made using K-Nearest Neighbors, the `sklearn` has an implementation called KNeighborsClassifier that w ill run much faster than our home-built version.

```
%%time
# Import
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import classification_report

# Instantiate classifier
# NB: Default distance is Euclidean
knn = KNeighborsClassifier(n_neighbors = 5)

# Fit model with training data
knn.fit(X_train, y_train)

# Create predictions for first 200 test observations
# # (As was done above with customKNN)
skpreds = knn.predict(X_test[:200])

print("sklearn prediction performance")
print(classification_report(y_test[:200], skpreds))


### The below lines of code will compare the performance of your home-built classification with
### The sklearn predictions -- if all the cells above were run sucessfully, you should see identical scores

print("\nHome-Built prediction performance")
print(classification_report(y_test[:200], custom_preds))


### The below lines of code will explicitly compare predictions:
### "differences" should == 0!

### NB: Commenting/uncommenting multiple lines in Jupyter can be accomplished with:
### <ctrl-/> on windows and <cmd-/> on mac
differences = 0
for cust, sk in zip(custom_preds, skpreds):
    if cust != sk:
        differences +=1
print("Total Differences:", differences)
```

### Practice with `sklearn`:

The below questions will ask you to create a new test/train split, and fit a new KNN model with `sklearn`.

All of these steps have already been performed above. Feel free to reference.

```
# Ensure Data is consistent

# read feature names
feats = pd.read_table(FEATURE_NAMES, sep='\n', header=None)

# read in training data
har_train = pd.read_table(TRAIN_DATA, sep='\s+', header=None)

# read in training labels, and clean them.
har_train_labels = pd.read_table(TRAIN_LABELS, sep='\n', header=None)
clean_features = [feat[0].split(' ')[1] for feat in feats.values]
har_train.columns = clean_features

har_train_labels = pd.read_table(TRAIN_LABELS, sep='\n', header=None)
har_train_labels.columns = ['label']
y = har_train_labels.loc[:, 'label']
```

### Question 12

Train Test Split

```
### GRADED
### Making a new test-train-split on our data:
### ### labels found in "y" and observations in `har_train`
### ### For split, specify - test_size of .4 and a random_state of 1738.
### assign output from the split to X_train2, X_test2, y_train2, y_test2 -- take care, the "X"s are capitlaized
### and the "y"s are lower-case.
### Which of the following would accomplish that task?

### 'a') X_train2, X_test2, y_train2, y_test2 = train_test_split(har_train, y, test_size = .4, random_state = 1738)
### 'b') X_train2, X_test2, y_train2, y_test2 = train_test_split(har_train, y, train_size = .4, random_state = 1738)
### 'c') X_train2, X_test2, y_train2, y_test2 = train_test_split(har_train, y, .4, 1738)
### 'd') X_train2, X_test2, y_train2, y_test2 = train_test_split(har_train, y, t_size = .4, rs = 1738)

### Assign character associated with you choice as string to ans1
### YOUR ANSWER BELOW

ans1 = 'a'
```

### Question 13

```
### This cell creates X_train3, X_test3, y_train3, and y_test3; used below.

X_train3, X_test3, y_train3, y_test3 = train_test_split(har_train, y, test_size = .4, random_state = 2001)
```

```
### GRADED
### Build a KNN classifier using sklearn.
### ### specify n_neighbors as 10. Otherwise accept the `KNeighborsClassifier` defaults.

### Fit the model using the provided "X_train3" and "y_train3" variables, from above cell.
### assign the predictions from your model for the provided "X_test3" data to a variable called ans1.
### YOUR ANSWER BELOW
```

```
"""
Example:

# Code for Instantiating a KNN Model

# Fit KNN with X_train3 and y_train3

# Create Predictions on X_test3

ans1 = preds

print(ans1[:5])
#-->np.array([6 5 1 6 4])

### NOTE: Your predictions may look different due to how "random_state" depends on current time.
"""

clas = KNeighborsClassifier(n_neighbors=10)
clas.fit(X_train3, y_train3)
preds = clas.predict(X_test3)
ans1 = preds
```

Building a model using sklearn is just as easy as those last two steps! As long as your data is in the right format, once you make your train/test split, the syntax for fitting pretty much any of the models in `sklearn` is about the same.

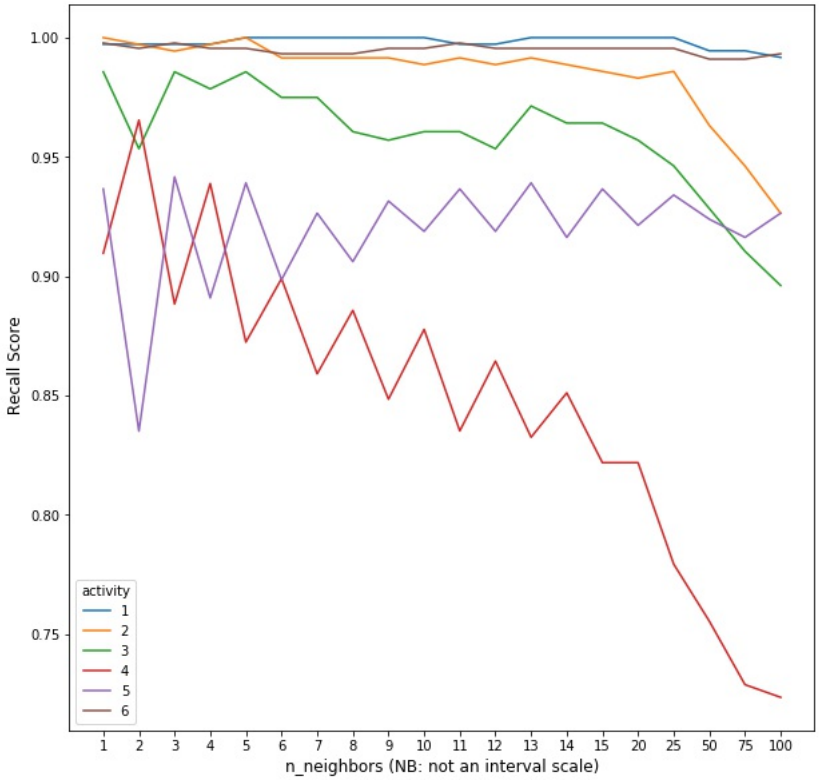
## Part 3: Interpret Results

For interpreting results we will be looking at the tradeoff between bias and variance as we change our `n_neighbors`. In many cases, false negatives are more costly and false positives. As such we will be looking primarily at the change in recall as we build a number of different models.

Note: The below takes some time to run. ~ 3.5min. Thus output is provided as screenshot below

```
# %%time
# from sklearn.metrics import recall_score

### Calculating Recal scores for multiple "n-neighbors"
# recall_scores = {}
# for n in [1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,20,25,50,75,100]:
#     knn = KNeighborsClassifier(n_neighbors=n)
#     knn.fit(X_train, y_train)
#     recall_scores[n] = recall_score(y_test, knn.predict(X_test), average = None)

### Put recall scores into DataFrame
# scores_df = pd.DataFrame(recall_scores).T
# scores_df.columns = [str(i) for i in range(1,7)]
# scores_df.index = scores_df.index.astype(str)

### Create plot of recall scores
# plt.figure(figsize = (10,10))
# for col in scores_df:
#     if col != 'n_neighbors':
#         plt.plot(scores_df[col], label = col)

# plt.ylabel(" Recall Score", fontsize = 12)
# plt.xlabel("n_neighbors (NB: not an interval scale)", fontsize = 12)
# plt.legend(title = "activity");
```

**Output**




### Question 14

```
### GRADED
### Looking at the recall scores above;
### as n_neighbors trends towards 100 do we see in increase in:

### 'a') bias
### 'b') variance

### Assign the string associated with your choice to ans1
### YOUR ANSWER BELOW

ans1 = 'a'
```

### Question 15

```
### GRADED
### In looking at the recall scores above, does it look like the bestter KNN models have:

### 'a') n_neighbors >= 15
### 'b') n_neigbors < 15

### Assign the string associated with your shoice to ans1
### YOUR ANSWER BELOW

ans1 = 'b'
```

### Question 16

```
### GRADED
### What might explain the ups-and-downs of recall in activities 4 and 6?

### 'a') calculation of Euclidean Distance
### 'b') Use of Entropy (information gain) for splitting
### 'c') tie-breaking/voting proceedures
### 'd') Simply a feature of KNN models: unavoidable
### Assign the string associated with your choice to ans1

### YOUR ANSWER BELOW

ans1 = 'c'
```

For this investigation of our model, recall w as used. In other instances other metrics might be more useful, or more translatable to your use-case.

Hopefully from this brief look into tuning our model and investigating the effects it has become clear that either "success" or "failure" might be due to circumstance.

Even in the search for the trade-off betw een bias and variance, a single observation or even a pair of observations might not tell the full story.