**Part 2**

**(b)**

**Problem statement –**

There are a range of different techniques you can investigate that could potentially improve the performance of either the distance-weighted k-NN and the basic k-NN from part 1. For example, you could look at the performance of these algorithms for different hyper-parameter settings (such as the value of k).

You should describe and investigate a comprehensive range of techniques that could potentially improve the accuracy of your basic k-NN and distance-weighted k-NN. Your report should document fully the different techniques, provide a justification for selecting and investigating these techniques and present the resulting accuracy. Please note that when you incorporate additional techniques you should implement these techniques in your own code using core Python or NumPy (rather than relying on imported functionality or high level functions).

There are few techniques that could potentially improve the performance of kNN algorithm:

1. **Normalization –**

Since the range of values of raw data varies widely. For example, when we calculate the distance between two points by the [Euclidean distance](https://en.wikipedia.org/wiki/Euclidean_distance) in kNN, If one of the features has a broad range of values, the distance will be governed by this particular feature. Therefore, the range of all features should be normalized so that each feature contributes approximately proportionately to the final distance.

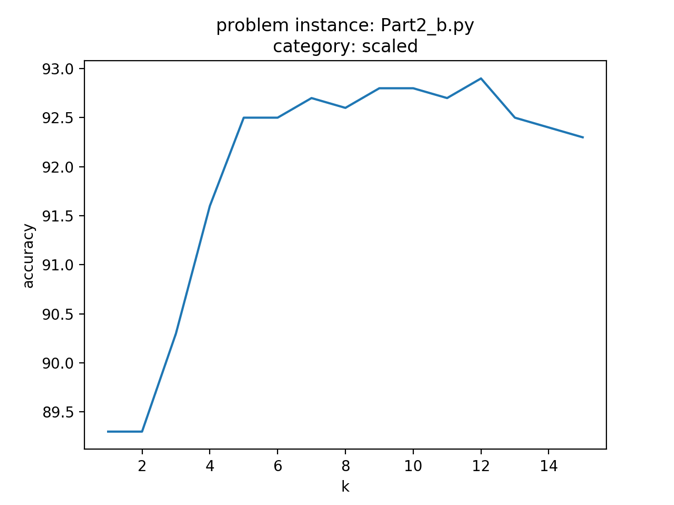
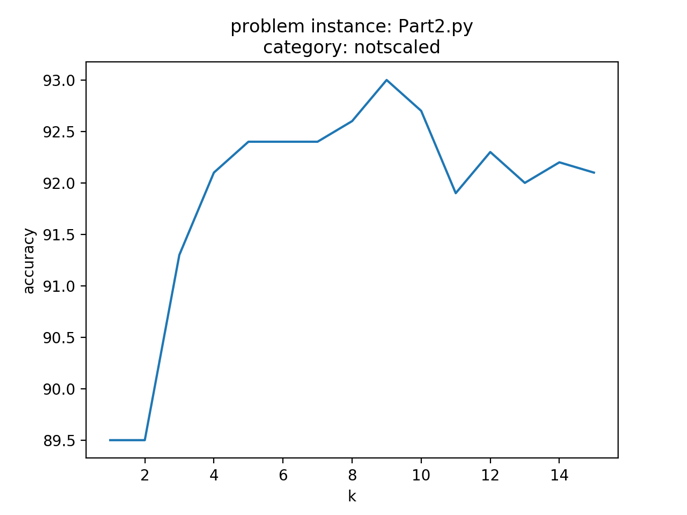
Min-max scaling is one of the many methods to achieve the normalization.

New value = (original value – minValue) / (maxValue - minValue)

It is very important that all the dimensions are normalized independently. Meaning, max and min value should be chosen from the ‘training dataset’ and should also be used with ‘test dataset’ when calculating ‘new value’

Using Range Normalization minimum and maximum value of a feature can be identified

I have shared my results in the graphs below in fig(a). Graph on the left shows the features not scaled and the one on the right had scaled features.



Fig(a)

1. **Irrelevant Features –**

By default all the features are included equally in the calculations. As a result number of features may skew the result even though they are of least impact to the classification.

In order to do so –

* Ignore all the irrelevant feature vectors from the dataset during compute. For example in this assignment the last columns for classification and regression had values for class category and regression. They had no meaning in the compute.

I removed those feature vector by slicing the numpy array in my code.

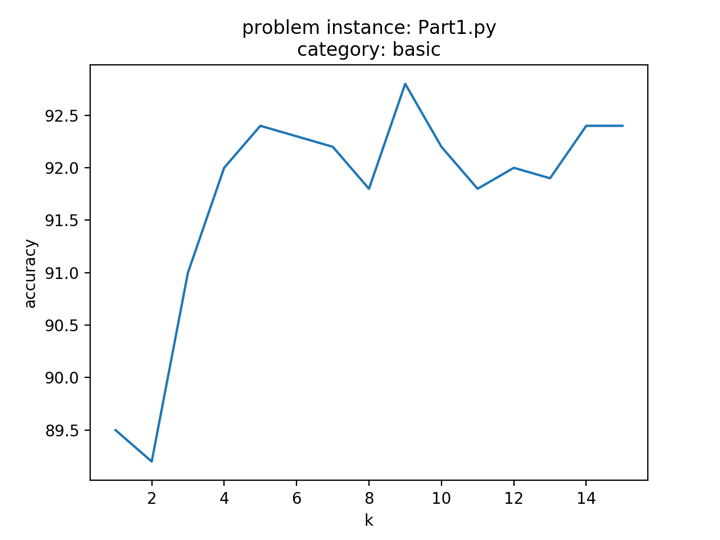
* More generic way to solve such problem is to assign weig to each dimention.
* The irrelevant features can be assigned value 0.

1. **Value of k**

Selection of an appropriate value of k is very important –

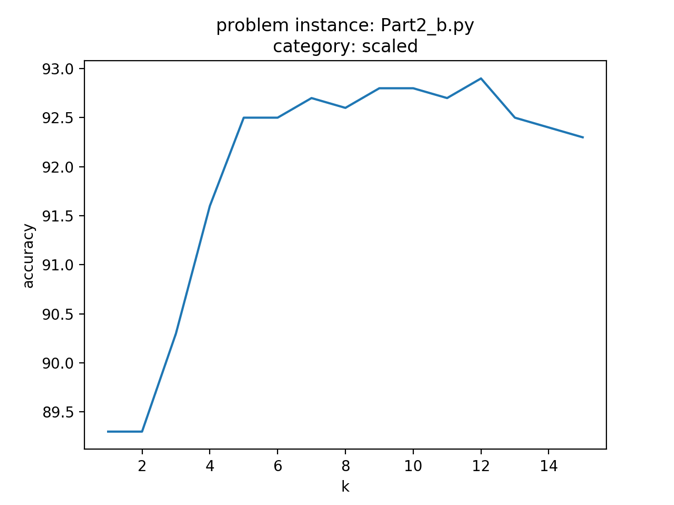
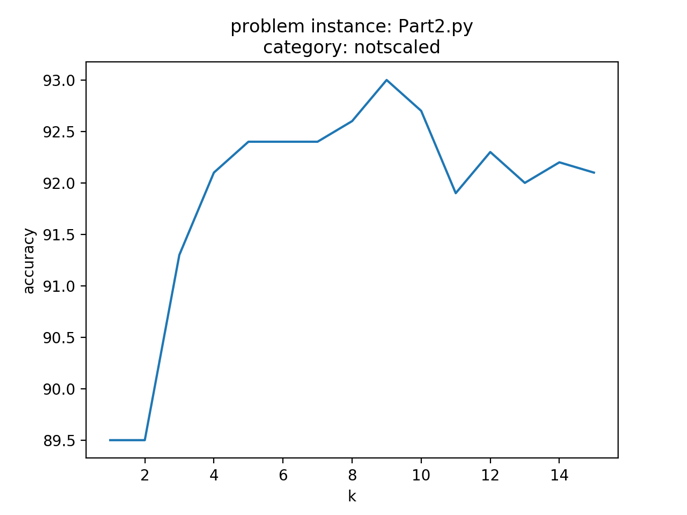
* Too small value of k can make the algorithm susceptible to noise and can overfit
* Larger value can reduce the noise but make classes less distinct

The graphs below illustrate the results and observations.



Fig(B) This is the result from Basic KNN execution (Part 1). Scales: x-axis (k) , y-axis(accuracy)

Accuracy = 89.5 for k = 1



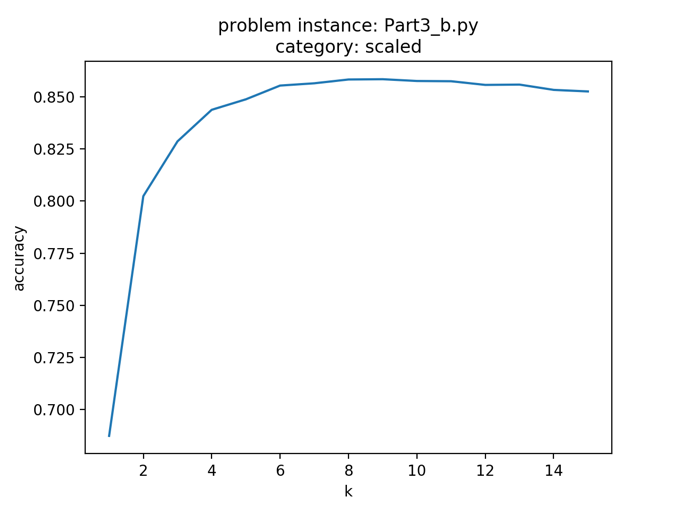
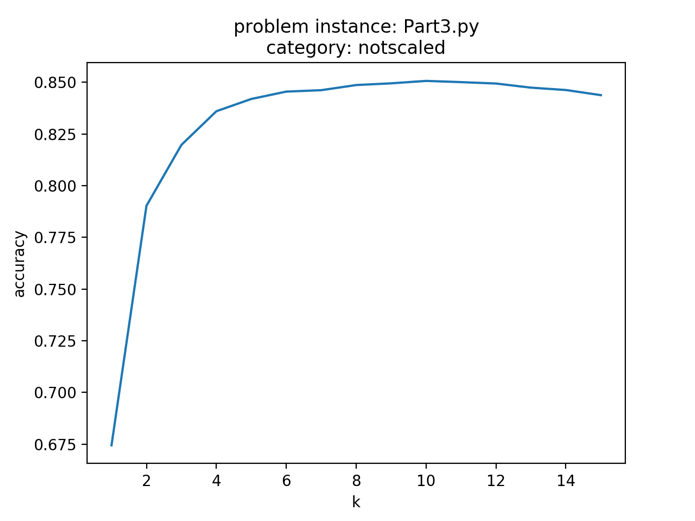
Fig(C) This is the result from Distance Weighted KNN Classification execution (Part2/Part2 (b)).

Scales: x-axis (k) , y-axis(accuracy).

Observations:

* Left Graph: When k = 9, accuracy is max for not-scaled feature values
* Right Graph: When k = 12, accuracy is max for scaled feature values

When the normalization is applied the accuracy becomes more consistent after k =5



Fig(D) This is the result from Distance Weighted KNN Regression execution (Part2/Part2 (b)).

Scales: x-axis (k) , y-axis(accuracy/rsquare).

Observations:

* for k = 10, accuracy = 85%
* graph on the right with scaled feature vectors is having better accuracy

**Part 3**

**(ii)**

* By default all the features are included equally in the calculations for kNN. As a result number of features may skew the result even though they are of least impact to the classification and hence will impact negatively the performance of kNN.

Few methods to tackle this issue:

Metric Learning -The K-nearest neighbour classification performance can often be significantly improved through (supervised) metric learning. Popular algorithms are  
•    Neighbourhood components analysis  
•    Large margin nearest neighbour  
Supervised metric learning algorithms use the label information to learn a new metric or pseudo metric.Feature Extraction –  
When the input data to an algorithm is too large to be processed and it is suspected to be redundant (e.g. the same measurement in both feet and meters) then the input data will be transformed into a reduced representation set of features (also named features vector). Transforming the input data into the set of features is called feature extraction. If the features extracted are carefully chosen it is expected that the features set will extract the relevant information from the input data in order to perform the desired task using this reduced representation instead of the full size input. Feature extraction is performed on raw data prior to applying k-NN algorithm on the transformed data in feature space.  
An example of a typical computer vision computation pipeline for face recognition using k-NN including feature extraction and dimension reduction pre-processing steps (usually implemented with OpenCV):  
•    Haar face detection  
•    Mean-shift tracking analysis  
•    PCA or Fisher LDA projection into feature space, followed by k-NN classificationDimension Reduction –  
For high-dimensional data (e.g., with number of dimensions more than 10) dimension reduction is usually performed prior to applying the k-NN algorithm in order to avoid the effects of the curse of dimensionality.

References - Wikipedia