22. Define a class that represents random variables with an exponential distribution.

A random variable represents a process that generates a random number. We can define a class to represent a random variable with an any distribution by providing a method called *'generate'* that uses the given distribution (random process eg: exponential/normal, etc) to generate a random number.

The following class generates random numbers from an exponential distribution.

```
import random

class RandomVariable(object):
    """Parent class for all random variables."""
    pass

class Exponential(RandomVariable):
  def __init__(self, lam):
    self.lam = lam

  def generate(self):
    return random.expovariate(self.lam)

a = Exponential(0.5)
a.generate()
```

Example Output: 2.819811073280088

44. Explain the k-NN algorithm with sample code.
The k-NN algorithm is arguably the simplest machine learning algorithm. Building the model consists only of storing the training dataset.

Algorithm:
Step 1: Determine parameter K = number of nearest neighbours
Step 2: Calculate the distance between new data point and all the training examples
Step 3: Sort the distances and determine the K nearest neighbours based on minimum distance.
Step 4: Gather the category Y of the K nearest neighbours
Step 5: Use simple majority of the category of K nearest neighbours as the prediction value for the new data point.

Sample Code:

Split data into training and test set so as to evaluate generalization performance:

```
from sklearn.datasets import load_breast_cancer
from sklearn.model_selection import train_test_split
cancer = load_breast_cancer()
X_train, X_test, y_train, y_test =
train_test_split(cancer.data, cancer.target,
stratify=cancer.target, random_state=0)
```

Import and instantiate the classifier. Set parameters i.e k = 3 (number of neighbours to use)

```
from sklearn.neighbors import KNeighborsClassifier
clf = KNeighborsClassifier(n_neighbors=3)
```

Fit the classifier using the training set. For KNeighborsClassifier this means storing the dataset, so as to compute neighbors during prediction:

```
clf.fit(X_train, y_train)
```

To make predictions on the test data, the predict method is called. For eac
in the test set, this computes its nearest neighbors in the training set and finds the

most common class among these.

```
print("Test set predictions: {}".format(clf.predict(X_test)))
```

To evaluate how well the model generalizes, the score method is called with the test data together with the test labels:

```
print("Test set accuracy: {:.2f}".format(clf.score(X_test,
y_test)))
```

If the model is say about 86% accurate, it means the model predicted the class correctly for 86% of the samples in the test dataset.

45. Walkthrough k-NN classification algorithm with an example.

The k-NN algorithm is arguably the simplest machine learning algorithm. Building the model consists only of storing the training dataset.

Algorithm:

Step 1: Determine parameter K = number of nearest neighbours

Step 2: Calculate the distance between new data point and all the training examples

Step 3: Sort the distances and determine the K nearest neighbours based on minimum distance.

Step 4: Gather the category Y of the K nearest neighbours

Step 5: Use simple majority of the category of K nearest neighbours as the prediction value for the new data point.

46. Explain the k-neighbourhood regression technique.

The k-neighbourhood regression technique is similar to the k-NN algorithm but is used for regression problems as opposed to classification.

Algorithm:

Step 1: Determine parameter K = number of nearest neighbours

Step 2: Calculate the distance between new data point and all the training examples

Step 3: Sort the distances and determine the K nearest neighbours based on minimum distance.

Step 4: Gather the output value Y of the K nearest neighbours

Step 5: Use the average, or mean, of the K nearest neighbors as the prediction value for the new data point.

47. How to calculate the effectiveness of k-NN algorithms?

```
from sklearn.datasets import load_breast_cancer
cancer = load_breast_cancer()
X_train, X_test, y_train, y_test = train_test_split(
```

```
cancer.data, cancer.target, stratify=cancer.target,
random_state=66)
training_accuracy = []
test_accuracy = []
# try n_neighbors from 1 to 10
neighbors_settings = range(1, 11)
for n_neighbors in neighbors_settings:
# build the model
clf = KNeighborsClassifier(n_neighbors=n_neighbors)
clf.fit(X_train, y_train)
# record training set accuracy
training_accuracy.append(clf.score(X_train, y_train))
# record generalization accuracy
test_accuracy.append(clf.score(X_test, y_test))
plt.plot(neighbors_settings, training_accuracy,
label="training accuracy")
plt.plot(neighbors_settings, test_accuracy, label="test
accuracy")
plt.ylabel("Accuracy")
plt.xlabel("n_neighbors")
plt.legend()
```

51. Write pseudocode to simulate various linear regression models.

Linear Regression

```
from sklearn.linear_model import LinearRegression
X, y = mglearn.datasets.load_extended_boston()
X_train, X_test, y_train, y_test = train_test_split(X, y,
random_state=42)
lr = LinearRegression().fit(X_train, y_train)
print("lr.coef_: {}".format(lr.coef_))
print("lr.intercept_: {}".format(lr.intercept_))
print("Training set score: {:.2f}".format(lr.score(X_train,
y_train)))
```

```
print("Test set score: {:.2f}".format(lr.score(X_test,
y_test)))
```

Ridge Regression

```
from sklearn.linear_model import Ridge
X, y = mglearn.datasets.load_extended_boston()
X_train, X_test, y_train, y_test = train_test_split(X, y,
random_state=42)
ridge = Ridge(alpha=1.0).fit(X_train, y_train)
print("ridge.coef_: {}".format(ridge.coef_))
print("ridge.intercept_: {}".format(ridge.intercept_))
print("Training set score:
{:.2f}".format(ridge.score(X_train, y_train)))
print("Test set score: {:.2f}".format(ridge.score(X_test,
y_test)))
```

Lasso Regression

```
from sklearn.linear_model import Lasso
X, y = mglearn.datasets.load_extended_boston()
X_train, X_test, y_train, y_test = train_test_split(X, y,
random_state=42)
lasso = Lasso().fit(X_train, y_train)
print("lasso.coef_: {}".format(lasso.coef_))
print("lasso.intercept_: {}".format(lasso.intercept_))
print("Training set score:
{:.2f}".format(lasso.score(X_train, y_train)))
print("Test set score: {:.2f}".format(lasso.score(X_test,
y_test)))
print("Number of features used: {}".format(np.sum(lasso.coef_
!= 0)))
```

53. Write the pseudocode for Naive bayes classifiers.

```
import numpy as np
X = np.array([[-1, -1], [-2, -1], [-3, -2], [1, 1], [2, 1], [3, 2]])
Y = np.array([1, 1, 1, 2, 2, 2])
from sklearn.naive_bayes import GaussianNB
clf = GaussianNB()
clf.fit(X, Y)
print(clf.predict([[-0.8, -1]]))
```

54. Why do Naive bayes classifiers perform better than linear models?

Naive Bayes classifiers tend to be faster in training compared to linear models. The reason that naive Bayes models are so efficient is that they learn parameters by looking at each feature individually and collect simple per-class statistics from each feature. The models work very well with high-dimensional sparse data and are relatively robust to the parameters. Naive Bayes models are great baseline models and are often used on very large datasets, where training even a linear model might take too long. The price paid for this efficiency is that naive Bayes models often provide generalization performance that is slightly worse than that of linear classifiers like LogisticRegression and LinearSVC.

55. Write the pseudocode for decision trees.

```
from sklearn.tree import DecisionTreeClassifier
from sklearn.datasets import load_breast_cancer
from sklearn.model_selection import train_test_split
cancer = load_breast_cancer()
X_train, X_test, y_train, y_test = train_test_split(
```

```
cancer.data, cancer.target, stratify=cancer.target,
random_state=42)
tree = DecisionTreeClassifier(max_depth=4, random_state=0)
tree.fit(X_train, y_train)
print("Accuracy on training set:
{:.3f}".format(tree.score(X_train, y_train)))
print("Accuracy on test set:
{:.3f}".format(tree.score(X_test, y_test)))
```

58. Write the pseudocode for random forests.

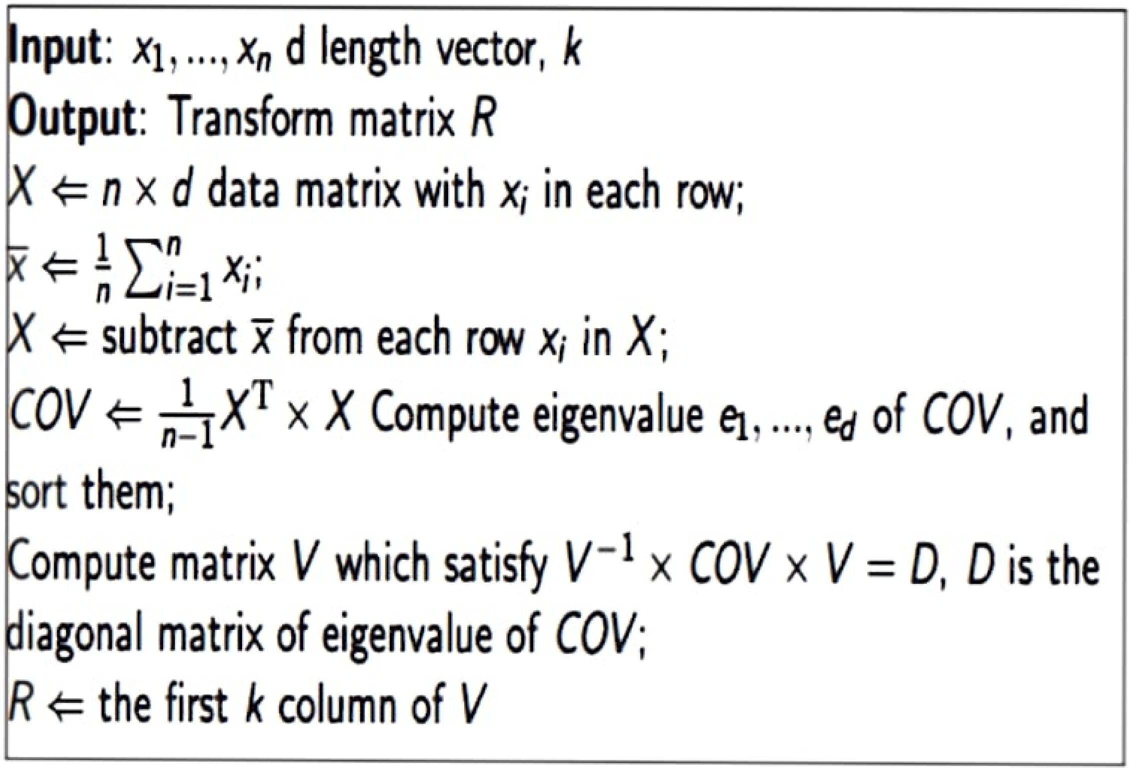
```
from sklearn.tree import RandomForestClassifier
from sklearn.datasets import load_breast_cancer
from sklearn.model_selection import train_test_split
cancer = load_breast_cancer()
X_train, X_test, y_train, y_test = train_test_split(
cancer.data, cancer.target, stratify=cancer.target,
random_state=42)
tree = RandomForestClassifier(n_estimators=100,
random_state=0)
tree.fit(X_train, y_train)
print("Accuracy on training set:
{:.3f}".format(tree.score(X_train, y_train)))
print("Accuracy on test set:
{:.3f}".format(tree.score(X_test, y_test)))
```

70.Write a pseudocode for calculating and visualising PCA.

In many problems, the measured data vectors are high-dimensional.Algorithms suffer from "Curse of Dimensionality".Instead of giving specific application of Dimensionality Reduction such as how to extract information from image, a simplified problem which takes n d dimension vector as input will be dressed here. More complicated applications can be somehow reduced to this problem.

Principal Component Analysis (PCA) was invented in 1901 by Karl Pearson PCA finds a linear projection of high dimensional data into a lower dimensional subspace such as The variance retained is maximized. The least square reconstruction error is minimized.

<u>Pseudo Code:</u>



```
Input: $x_1, ..., x_n$ d length vector, $k$
Output: Transform matrix $R$
1 $X \Leftarrow n \times d$ data matrix with $x_i$ in each row;
2 $\overline{x} \Leftarrow \frac{1}{n} \sum_{i=1}^{n} x_i$;
3 $X \Leftarrow$ subtract $\overline{x}$ from each row $x_i$ in $X$;
4 $COV \Leftarrow \frac{1}{n-1} X^{T} \times X$ Compute eigenvalue $e_1, ..., e_d$ of $COV$, and
  sort them;
5 Compute matrix $V$ which satisfy $V^{-1} \times COV \times V = D$, $D$ is the
  diagonal matrix of eigenvalue of $COV$;
6 $R \Leftarrow$ the first $k$ column of $V$
```

72. Write the pseudo code for k-means clustering algorithm.
k-means clustering is a method of vector quantization, originally from signal processing, that is popular for cluster analysis in data mining. k-means clustering aims to partition n observations into k clusters in which each observation belongs to the cluster with the nearest mean, serving as a prototype of the cluster. This results in a partitioning of the data space into Voronoi cells.

```
Input: k (the number of clusters),
       D (a set of lift ratios)
Output: a set of k clusters
Method:
Arbitrarily choose k objects from D as the initial cluster centers;
Repeat:
  1. (re)assign each object to the cluster to which the object is
     the most similar, based on the mean value of the objects in the
     cluster;
  2. Update the cluster means, i.e., calculate the mean value of
     the objects for each cluster
Until no change;
```

74.Write the pseudo code for agglomerative hierarchical clustering.

This is a major data mining algorithm.Agglomerative aThe input to this algorithm is (1) a data-set consisting of points in an *n*-dimensional space and (2) a measure of the similarity between items in the data set. We refer to this measure of similarity as a *distance metric*; the more dissimilar items are, the farther apart they are according to the distance metric. The output of the algorithm is a binary tree (called a *dendrogram*) representing a hierarchical, pair-wise clustering of the items in the data set. The parallelism in this algorithm arises from the multiple pairs of points that can be clustered independently.

Initially, all data points are placed onto a worklist (line 1). We then build a *kd-tree*, an acceleration structure that allows the quick determination of a point's nearest neighbor (line 2). The algorithm proceeds as follows. For each point *p* in the worklist, find its nearest neighbor *q* (line 5) and determine if *q*'s nearest neighbor is *p* (lines 6-8). If so, cluster *p* and *q* and insert a new point into the new nodes list representing the new clusters. Otherwise, place *p* back onto the new nodes list to be processed in the next iteration of the outer loop. The new clusters are then copied back to the worklist after rebuilding a new Kd-Tree. The algorithm terminates when there is only one point left in the worklist.

The pseudo code is shown below:

```
Workset ws = new Set(points);
KDTree kdtree = new KDTree(points);
while (true) {
  foreach (Element p in ws) {
    if (p.hasCluster()) continue;
    Point q = kdtree.findNearest(p);
    if (q == null) break; // stop if p is last element
    Point r = kdtree.findNearest(q);
    if (p == r) { // create new cluster e that contains a and b
      Element e = cluster(p, q);
      newWork.add(e);
    } else { // can't cluster yet, try again later
      newWork.add(p); // add back to worklist
    }
  }
  if (newWork.size() == 1) // we have a single cluster
    break;

  ws.addAll(newWork); //add new nodes to worklist
  kdtree.clear();
  kdtree.addAll(newWork);
  newWork.clear();
}
```

76. Write the pseuo code for hierarchical clustering algorithm.

Given a set of N items to be clustered, and an N*N distance (or similarity) matrix, the basic process of hierarchical clustering is this:

1. Start by assigning each item to a cluster, so that if you have N items, you now have N clusters, each containing just one item. Let the distances (similarities) between the clusters the same as the distances (similarities) between the items they contain.
2. Find the closest (most similar) pair of clusters and merge them into a single cluster, so that now you have one cluster less.
3. Compute distances (similarities) between the new cluster and each of the old clusters.
4. Repeat steps 2 and 3 until all items are clustered into a single cluster of size N. (*)

```
1. Begin with the disjoint clustering having level $L(0) = 0$ and sequence number $m = 0$.
2. Find the least dissimilar pair of clusters in the current clustering, say pair (r), (s), according to

      $d[(r),(s)] = \min d[(i),(j)]$

   where the minimum is over all pairs of clusters in the current clustering.
3. Increment the sequence number : $m = m + 1$. Merge clusters (r) and (s) into a single cluster to form the next clustering m. Set the level of
   this clustering to

      $L(m) = d[(r),(s)]$
4. Update the proximity matrix, D, by deleting the rows and columns corresponding to clusters (r) and (s) and adding a row and column
   corresponding to the newly formed cluster. The proximity between the new cluster, denoted (r,s) and old cluster (k) is defined in this way:

      $d[(k), (r,s)] = \min d[(k),(r)], d[(k),(s)]$
5. If all objects are in one cluster, stop. Else, go to step 2.
```

77.Explain DBSCAN Algorithm with a sample dataset.

DBSCAN is a well-known data clustering algorithm that is commonly used in data mining and machine learning.

Based on a set of points (let's think in a bidimensional space as exemplified in the figure), DBSCAN groups together points that are close to each other based on a distance measurement (usually Euclidean distance) and a minimum number of points. It also marks as outliers the points that are in low-density regions.

Parameters:
The DBSCAN algorithm basically requires 2 parameters:
eps: the minimum distance between two points. It means that if the distance between two points is lower or equal to this value (eps), these points are considered neighbors.
minPoints: the minimum number of points to form a dense region. For example, if we set the minPoints parameter as 5, then we need at least 5 points to form a dense region.
Parameter estimation:
The parameter estimation is a problem for every data mining task. To choose good parameters we need to understand how they are used and have at least a basic previous knowledge about the data set that will be used.
eps: if the eps value chosen is too small, a large part of the data will not be clustered. It will be considered outliers because don't satisfy the number of points to create a dense region. On the other hand, if the value that was chosen is too high, clusters will merge and the majority of objects will be in the same cluster. The eps should be chosen based on the distance of the dataset (we can use a k-distance graph to find it), but in general small eps values are preferable.
minPoints: As a general rule, a minimum minPoints can be derived from a number of dimensions (D) in the data set, as minPoints ≥ D + 1. Larger values are usually better for data sets with noise and will form more significant clusters. The minimum value for the minPoints must be 3, but the larger the data set, the larger the minPoints value that should be chosen.

Let Netflix be an example. Based on previous shows you have watched in the past, Netflix will recommend shows for you to watch next. Anyone who has ever watched or been on Netflix has seen the screen below with recommendations

78.Write a pseudo code to perform DBSCAN algorithm.

Density-based spatial clustering of applications with noise (DBSCAN) is a data clustering algorithm proposed by Martin Ester, Hans-Peter Kriegel, Jörg Sander and Xiaowei Xu in 1996.It is a density-based clustering algorithm: given a set of points in some space, it groups together points that are closely packed together (points with many nearby neighbors), marking as outliers points that lie alone in low-density regions (whose nearest neighbors are too far away). DBSCAN is one of the most common clustering algorithms and also most cited in scientific literature.

The pseudo code for DBSCAN algorithm can be shown below:

```
DBSCAN(D, epsilon, min_points):
    C = 0
    for each unvisited point P in dataset
        mark P as visited
        sphere_points = regionQuery(P, epsilon)
        if sizeof(sphere_points) < min_points
            ignore P
        else
            C = next cluster
            expandCluster(P, sphere_points, C, epsilon, min_points)

expandCluster(P, sphere_points, C, epsilon, min_points):
    add P to cluster C
    for each point P’ in sphere_points
        if P’ is not visited
            mark P' as visited
            sphere_points' = regionQuery(P’, epsilon)
            if sizeof(sphere_points’) >= min_points
                sphere_points = sphere_points joined with sphere_points’
            if P’ is not yet member of any cluster
                add P' to cluster C

regionQuery(P, epsilon):
    return all points within the n-dimensional sphere centered at P with radius epsilon (including
P)
```