HW- 3 Similarity Based KNN

Ques 2. Email spam filtering models often use a bag-of-words representation for emails. In a bag-of-words representation, the descriptive features that describe a document (in our case, an email) each represent how many times a particular word occurs in the document. One descriptive feature is included for each word in a predefined dictionary. The dictionary is typically defined as the complete set of words that occur in the training dataset. The table below lists the bag-of-words representation for the following five emails and a target feature, SPAM, whether they are spam emails or genuine emails:

- *"money, money, money"*
- *"free money for free gambling fun"*
- *"gambling for fun"*
- *"machine learning for fun, fun, fun"*
- *"free machine learning"*

	Bag-of-Words							
ID	MONEY	FREE	FOR	GAMBLING	FUN	MACHINE	LEARNING	SPAM
1	3	0	0	0	0	0	0	true
2	1	2	1	1	1	0	0	true
3	0	0	1	1	1	0	0	true
4	0	0	1	0	3	1	1	false
5	0	1	0	0	0	1	1	false

a. What target level would a nearest neighbour model using Euclidean distance return for the following email: "machine learning for free"?

Ans: Formula for Euclidean distance is

$$Euclidean(\mathbf{a}, \mathbf{b}) = \sqrt{\sum_{i=1}^{m} (\mathbf{a}[i] - \mathbf{b}[i])^2}$$

For the above query, the bag of words would look like:

Money	Free	For	Gambling	Fun	Machine	Learning
0	1	1	0	0	1	1

Hence, Euclidean distance between query and instance 1(ID =1) is given below:

Euclidean(ID=1, query)= math.sqrt((3-0)**2 + (0-1)**2+ (0-1)**2 + (0-0)**2 + (0-0)**2 + (0-1)**2 + (0-1)**2) = sqrt(9 + 1 + 1 +0 +0 +1+1) = sqrt(13) = √13= 3.6056

Likewise, Euclidean distance between query and instances is calculated below in the table

ID	$(q_i - d_i)^2$, where d is the word in each instance							Euclidean Distance
	Money	Free	For	Gambling	Fun	Machine	Learning	
1	9	1	1	0	0	1	1	3.6056
2	1	1	0	1	1	1	1	2.4495
3	0	1	0	1	1	1	1	2.2361
4	0	1	0	0	9	0	0	3.1623
5	0	0	1	0	0	0	0	1

Therefore, based on the Euclidean distance calculated above, we can see that the minimum distance is for instance ID= 5. That is ID=5 is the nearest neighbor to the query.
Hence for the query "machine learning for free", **the model will return target value = False i.e. it is not SPAM.**

b. What target level would a k-NN model with k = 3 and using Euclidean distance return for the same query?

Ans: As per the Euclidean Distance calculated above, we can see that for k=3, three nearest neighbour are ID= 5,2,3. Instances

- d_5 have target value = False
- d_2 have target value = True
- d_3 have target value = True

So, majority of the value = True. Hence, **3-NN model will return the prediction of SPAM = True for query.**

c. What target level would a weighted k-NN model with k = 5 and using a weighting scheme of the reciprocal of the squared Euclidean distance between the neighbor and the query, return for the query?

Ans: The formula for weighted K-NN model is given below. It is calculated as the reciprocal of square of the Euclidean Distance.

$$\mathbb{M}_k(\mathbf{q}) = \arg\max_{l \in levels(t)} \sum_{i=1}^{k} \frac{1}{dist(\mathbf{q}, \mathbf{d_i})^2} \times \delta(t_i, l)$$

For d_1, weight = 1/(Euclidean distance)^2= 1/ $(3.6056)^2$ = 0.0769.
Likewise, the weights for all instances are calculated below

ID	$(q_i - d_i)^2$, where d is the word in each instance								Euclidean Distance	Weights
	Money	**Free**	**For**	**Gambling**	**Fun**	**Machine**	**Learning**	**Spam**		
1	9	1	1	0	0	1	1	True	3.6056	0.0769
2	1	1	0	1	1	1	1	True	2.4495	0.1667
3	0	1	0	1	1	1	1	True	2.2361	0.1999
4	0	1	0	0	9	0	0	False	3.1623	0.0999
5	0	0	1	0	0	0	0	False	1	1

For Spam= True, total weight = 0.0769+0.1667+0.1999= 0.4435
For Spam= False, total weight = 0.0999+ 1 = 1.0999
Hence, **maximum value of weight is for Spam= False, so the prediction model will return Spam= False for the query**.

d. What target level would a k-NN model with k = 3 and using Manhattan distance return for the same query?

Ans: Formula to calculate Manhattan distance is given below.

$$Manhattan(\mathbf{a}, \mathbf{b}) = \sum_{i=1}^{m} abs(\mathbf{a}[i] - \mathbf{b}[i])$$

For the above query, the bag of words would look like:

Money	Free	For	Gambling	Fun	Machine	Learning
0	1	1	0	0	1	1

Hence, Manhattan distance between query and instance 1(ID =1) is given below:

Manhattan(ID=1, query)= |3-0| + |0-1|+ |0-1| + |0-0| + |0-0| + |0-1|+ |0-1| = 3 + 1 + 1 +0 +0 +1+1) = 7

Likewise Manhattan distance for all instances is calculated below

ID	$\lvert q_i - d_i \rvert$, where d is the word in each instance							Manhattan Distance
	Money	**Free**	**For**	**Gambling**	**Fun**	**Machine**	**Learning**	
1	3	1	1	0	0	1	1	7
2	1	1	0	1	1	1	1	6
3	0	1	0	1	1	1	1	5
4	0	1	0	0	3	0	0	4
5	0	0	1	0	0	0	0	1

Hence, based on Manhattan distance three nearest neighbors are d_5 ,d_4, d_3. For k = 3,

- d_5 have target value = False
- d_4 have target value = False
- d_3 have target value = True

Majority of the target value = False, **the 3-NN model based on Manhattan distance will return False for the query.**

e. There are a lot of zero entries in the spam bag-of-words dataset. This is indicative of sparse data and is typical for text analytics. Cosine similarity is often a good choice when dealing with sparse non-binary data. What target level would a 3-NN model

using cosine similarity return for the query?
Ans: Formula to calculate Cosine Similarity is given below

$$sim_{COSINE}(\mathbf{a},\mathbf{b}) = \frac{\mathbf{a}\cdot\mathbf{b}}{\sqrt{\sum_{i=1}^{m}\mathbf{a}[i]^2}\times\sqrt{\sum_{i=1}^{m}\mathbf{b}[i]^2}}$$

Steps for calculating cosine similarity is given below:

Step1: first we will calculate the vector length for each instances and query
Formula to calculate vector length is $\sqrt{(\sum X_i^2)}$ where x is for instances and query.

Vector length for d1= math.sqrt(3**2+ 0**2 + 0**2 + 0**2 + 0**2+0**2 +0**2)= $\sqrt{9} = 3$
Vector length for query= math.sqrt(0**2+ 1**2 + 1**2 + 0**2 + 0**2+1**2 +1**2)= $\sqrt{4} = 2$

Step2: we will calculate the dot product between each instance and query i.e. (q_i . d_i)

Dot product of query and instance d1 (q.d1) = (0*3 + 1*0 + 1*0 + 0*0 + 0*0 + 1*0 + 1*0)= 0

Step3: To calculate cosine similarity we shall divide dot product by query vector length and instance vector length.

Cosine similarity (q,d1) = dot product / (instance vector length * query vector length) = 0/ (2*3) = **0**

Likewise, cosine similarity between query and each instance is calculated below in the table

Step1: first we will calculate the vector length for each instances and query as shown in the table below.

ID	(Vector Length)								Vector Length
	Money	Free	For	Gambling	Fun	Machine	Learning	Sum	
1	9	0	0	0	0	0	0	9	3
2	1	4	1	1	1	0	0	8	2.8284
3	0	0	1	1	1	0	0	3	1.7320
4	0	0	1	0	9	1	1	12	3.4641
5	0	1	0	0	0	1	1	3	1.7320
query	0	1	1	0	0	1	1	4	2

Step2: we will calculate the dot product between each instance and query i.e. (q_i . d_i) which is shown in the table below.

(q_i . d_i)								Dot product
	Money	Free	For	Gambling	Fun	Machine	Learning	
(q,d1)	0	0	0	0	0	0	0	0
(q,d2)	0	2	1	0	0	0	0	3
(q,d3)	0	0	1	0	0	0	0	1
(q,d4)	0	0	1	0	0	1	1	3
(q,d5)	0	1	0	0	0	1	1	3

Step3: To calculate cosine similarity we shall divide dot product by query vector length and instance vector length. Calculation are shown in the table below

(q_i . d_i)	Dot product (q,d_i)							Sum of Dot product	Vector Length	Cosine Similarity
	Money	Free	For	Gambling	Fun	Machine	Learning			
(q,d1)	0	0	0	0	0	0	0	0	q= 2, d= 3	0/ (2*3)= **0**
(q,d2)	0	2	1	0	0	0	0	3	q= 2, d= 2.8284	3/(2*2.8284)= **0.5303**
(q,d3)	0	0	1	0	0	0	0	1	q= 2, d= 1.7320	1/(2*1.7320)= **0.2887**
(q,d4)	0	0	1	0	0	1	1	3	q= 2, d= 3.4641	3/(2*3.4641)= **0.4330**
(q,d5)	0	1	0	0	0	1	1	3	q= 2, d= 1.7320	3/(2*1.7320)= **0.8660**

In cosine similarity, the higher the value is more similar it is to the particular instance. For 3-NN model and from the table above, we can see that for instance d5,d4,d2 values are higher.

- d_5 have target value = False
- d_4 have target value = False
- d_2 have target value = True

Majority of the target value = False, **the 3-NN model based on cosine similarity will return False for the query.**

Ques. 3. The predictive task in this question is to predict the level of corruption in a country based on a range of macro-economic and social features. The table below lists some countries described by the following descriptive features:

- LIFE EXP., the mean life expectancy at birth
- TOP-10 INCOME, the percentage of the annual income of the country that goes to the top 10% of earners
- INFANT MORT., the number of infant deaths per 1,000 births
- MIL. SPEND, the percentage of GDP spent on the military
- SCHOOL YEARS, the mean number years spent in school by adult females

The target feature is the Corruption Perception Index (CPI). The CPI measures the perceived levels of corruption in the public sector of countries and ranges from 0 (highly corrupt) to 100 (very clean).[34]

COUNTRY ID	LIFE EXP.	TOP-10 INCOME	INFANT MORT.	MIL. SPEND	SCHOOL YEARS	CPI
Afghanistan	59.61	23.21	74.30	4.44	0.40	1.5171
Haiti	45.00	47.67	73.10	0.09	3.40	1.7999
Nigeria	51.30	38.23	82.60	1.07	4.10	2.4493
Egypt	70.48	26.58	19.60	1.86	5.30	2.8622
Argentina	75.77	32.30	13.30	0.76	10.10	2.9961
China	74.87	29.98	13.70	1.95	6.40	3.6356
Brazil	73.12	42.93	14.50	1.43	7.20	3.7741
Israel	81.30	28.80	3.60	6.77	12.50	5.8069
U.S.A	78.51	29.85	6.30	4.72	13.70	7.1357
Ireland	80.15	27.23	3.50	0.60	11.50	7.5360
U.K.	80.09	28.49	4.40	2.59	13.00	7.7751
Germany	80.24	22.07	3.50	1.31	12.00	8.0461
Canada	80.99	24.79	4.90	1.42	14.20	8.6725
Australia	82.09	25.40	4.20	1.86	11.50	8.8442
Sweden	81.43	22.18	2.40	1.27	12.80	9.2985
New Zealand	80.67	27.81	4.90	1.13	12.30	9.4627

We will use Russia as our query country for this question. The table below lists the descriptive features for Russia.

COUNTRY ID	LIFE EXP.	TOP-10 INCOME	INFANT MORT.	MIL. SPEND	SCHOOL YEARS	CPI
Russia	67.62	31.68	10.00	3.87	12.90	?

a. What value would a 3-nearest neighbour prediction model using Euclidean distance return for the CPI of Russia?

Ans: Formula to calculate Euclidean distance is

$$Euclidean(\mathbf{a},\mathbf{b}) = \sqrt{\sum_{i=1}^{m}(\mathbf{a}[i]-\mathbf{b}[i])^2}$$

Euclidean(query, Afghanistan) = math.sqrt((67.62- 59.61)**2 + (31.68-23.21)**2 + (10-74.30)**2 + (3.87-4.44)**2 + (12.9- 0.40)**2) = math.sqrt(64.1601+ 71.7409+ 4134.49+ 0.3249 + 156.25) = math.sqrt(4426.9659) = 66.5354

Likewise Euclidean distance for all instances is calculated below

Country ID	Life Exp	Top 10 income	Infant Mort	Mil spend	School years	CPI	Euclidean Distance
Afghanistan	59.61	23.21	74.30	4.44	0.40	1.5171	66.5354
Haiti	45.00	47.67	73.10	0.09	3.40	1.7999	69.6670
Nigeria	51.30	38.23	82.60	1.07	4.10	2.4493	75.2681
Egypt	70.48	26.58	19.60	1.86	5.30	2.8622	13.7168
Argentina	**75.77**	**32.30**	**13.30**	**0.76**	**10.10**	**2.9961**	**9.7575**
China	**74.87**	**29.98**	**13.70**	**1.95**	**6.40**	**3.6356**	**10.7275**
Brazil	73.12	42.93	14.50	1.43	7.20	3.7741	14.6801
Israel	81.30	28.80	3.60	6.77	12.50	5.8069	15.6514
U.S.A	**78.51**	**29.85**	**6.30**	**4.72**	**13.70**	**7.1357**	**11.7044**
Ireland	80.15	27.23	3.50	0.60	11.50	7.5360	15.2219
U.K.	80.09	28.49	4.40	2.59	13.00	7.7751	14.0956
Germany	80.24	22.07	3.50	1.31	12.00	8.0461	17.3560
Canada	80.99	24.79	4.90	1.42	14.20	8.6725	16.1224
Australia	82.09	25.40	4.20	1.86	11.50	8.8442	16.9841
Sweden	81.43	22.18	2.40	1.27	12.80	9.2985	18.5875
New Zealand	80.67	27.81	4.90	1.13	12.30	9.4627	14.8040

From the Euclidean distance calculated above we can see that three nearest neighbor are Argentina, China and USA. Hence, the model will return CPI value which will be the average of the CPI values of these three countries.

$CPI_{(Russia)}$ = (2.9961 +3.6356+7.1357)/ 3 = 4.5891

b. What value would a weighted k-NN prediction model return for the CPI of Russia? Use k = 16 (i.e., the full dataset) and a weighting scheme of the reciprocal of the squared Euclidean distance between the neighbor and the query.

Ans: Formula to calculate Weighted average of KNN model is given below

$$\mathbb{M}_k(\mathbf{q}) = \frac{\sum_{i=1}^{k} \frac{1}{dist(\mathbf{q}, \mathbf{d}_i)^2} \times t_i}{\sum_{i=1}^{k} \frac{1}{dist(\mathbf{q}, \mathbf{d}_i)^2}}$$

The formula for weights. It is calculated as the reciprocal of square of the Euclidean Distance.

$$\mathbb{M}_k(\mathbf{q}) = \arg\max_{l \in levels(t)} \sum_{i=1}^{k} \frac{1}{dist(\mathbf{q}, \mathbf{d}_i)^2} \times \delta(t_i, l)$$

Weights for Afghanistan = 1/ (66.5354)**2 = 1/ 4426.9594= 0.0002.

Likewise table below shows the weights for each country

Country ID	CPI	Euclidean Distance	Weights= 1/(Euc.Distance)**2	Weight X CPI
Afghanistan	1.5171	66.5354	0.0002	0.0003
Haiti	1.7999	69.6670	0.0002	0.0004
Nigeria	2.4493	75.2681	0.0002	0.0004
Egypt	2.8622	13.7168	0.0053	0.0152
Argentina	2.9961	9.7575	0.0105	0.0315
China	3.6356	10.7275	0.0087	0.0316
Brazil	3.7741	14.6801	0.0046	0.0175
Israel	5.8069	15.6514	0.0041	0.0237
U.S.A	7.1357	11.7044	0.0073	0.0521
Ireland	7.5360	15.2219	0.0043	0.0325
U.K.	7.7751	14.0956	0.0050	0.0391
Germany	8.0461	17.3560	0.0033	0.0267
Canada	8.6725	16.1224	0.0038	0.0334
Australia	8.8442	16.9841	0.0035	0.0307
Sweden	9.2985	18.5875	0.0029	0.0269
New Zealand	9.4627	14.8040	0.0046	0.0432
SUM			0.0686	0.4052

As the value of K = 16, So the value of the CPI predicted by the model will be :
Russia(CPI) = 0.4052/ 0.0686= 5.9067

c. The descriptive features in this dataset are of different types. For example, some are percentages, others are measured in years, and others are measured in counts per 1,000. We should always consider normalizing our data, but it is particularly important to do this when the descriptive features are measured in different units. What value would a 3-nearest neighbor prediction model using Euclidean distance return for the CPI of Russia.

Ans: Formula to normalise values in the range (0,1) if given below

$A_{i=}$ (A_i – min(A)/ max(A)- min(A)) * (high – low) + low

Normalised values for Afghanistan:

Life Exp = (59.61- 45) / (82.09 – 45) = 0.3939

Top 10 Income= (23.21- 22.07)/ (47.67- 22.07) = 0.0445

Infant mort= (74.3- 2.4)/ (82.09- 2.4)= 0.9022

Mil spend= (4.44- 0.09)/ (6.77- 0.09)= 0.6512

School years = (0.4-0.4)/(14.2- 0.4)= 0.0000

Likewise normalised values for all columns are shown below

Country ID	Life Exp	Top 10 income	Infant Mort	Mil spend	School years
Afghanistan	0.3939	0.0445	0.9022	0.6512	0.0000
Haiti	0.0000	1.0000	0.8872	0.0000	0.2174
Nigeria	0.1699	0.6313	1.0064	0.1467	0.2681
Egypt	0.6870	0.1762	0.2158	0.2650	0.3551
Argentina	0.8296	0.3996	0.1368	0.1003	0.7029
China	0.8053	0.3090	0.1418	0.2784	0.4348
Brazil	0.7582	0.8148	0.1518	0.2006	0.4928
Israel	0.9787	0.2629	0.0151	1.0000	0.8768
U.S.A	0.9035	0.3039	0.0489	0.6931	0.9638
Ireland	0.9477	0.2016	0.0138	0.0763	0.8043
U.K.	0.9461	0.2508	0.0251	0.3743	0.9130
Germany	0.9501	0.0000	0.0138	0.1826	0.8406
Canada	0.9703	0.1063	0.0314	0.1991	1.0000
Australia	1.0000	0.1301	0.0226	0.2650	0.8043
Sweden	0.9822	0.0043	0.0000	0.1766	0.8986
New Zealand	0.9617	0.2242	0.0314	0.1557	0.8623

We will also normalize the values for query:

Country ID	Life Exp	Top 10 income	Infant Mort	Mil spend	School years
Russia	0.6099	0.3754	0.0954	0.5659	0.9058

Now calculating the Euclidean distance based on normalized values. Calculations are shown in the table below

Country ID	Life Exp	Top 10 income	Infant Mort	Mil spend	School years	CPI	Euclidean Distance
Afghanistan	0.3939	0.0445	0.9022	0.6512	0.0000	1.5171	1.2786
Haiti	0.0000	1.0000	0.8872	0.0000	0.2174	1.7999	1.4776
Nigeria	0.1699	0.6313	1.0064	0.1467	0.2681	2.4493	1.2928
Egypt	0.6870	0.1762	0.2158	0.2650	0.3551	2.8622	0.6737
Argentina	**0.8296**	**0.3996**	**0.1368**	**0.1003**	**0.7029**	2.9961	**0.5554**
China	0.8053	0.3090	0.1418	0.2784	0.4348	3.6356	0.5910
Brazil	0.7582	0.8148	0.1518	0.2006	0.4928	3.7741	0.7227
Israel	0.9787	0.2629	0.0151	1.0000	0.8768	5.8069	0.5869
U.S.A	**0.9035**	**0.3039**	**0.0489**	**0.6931**	**0.9638**	7.1357	**0.3362**
Ireland	0.9477	0.2016	0.0138	0.0763	0.8043	7.5360	0.6332
U.K.	**0.9461**	**0.2508**	**0.0251**	**0.3743**	**0.9130**	7.7751	**0.4126**
Germany	0.9501	0.0000	0.0138	0.1826	0.8406	8.0461	0.6438
Canada	0.9703	0.1063	0.0314	0.1991	1.0000	8.6725	0.5915
Australia	1.0000	0.1301	0.0226	0.2650	0.8043	8.8442	0.5644

Sweden	0.9822	0.0043	0.0000	0.1766	0.8986	9.2985	0.6611
New Zealand	0.9617	0.2242	0.0314	0.1557	0.8623	9.4627	0.5665

From the Euclidean distance calculated above we can see that three nearest neighbors are USA,UK, and Argentina. Hence the 3-NN model will predict CPI values as the average of CPI for these three countries.

CPI(Russia) = (2.9961 + 7.1357 + 7.7751) /3 = 5.9690

d. What value would a weighted k-NN prediction model—with k = 16 (i.e., the full dataset) and using a weighting scheme of the reciprocal of the squared Euclidean distance between the neighbor and the query—return for the CPI of Russia when it is applied to the range-normalized data?

Ans: Formula to calculate Weighted average of KNN model is given below

$$\mathbb{M}_k(\mathbf{q}) = \frac{\sum_{i=1}^{k} \frac{1}{dist(\mathbf{q}, \mathbf{d}_i)^2} \times t_i}{\sum_{i=1}^{k} \frac{1}{dist(\mathbf{q}, \mathbf{d}_i)^2}}$$

The formula for weights. It is calculated as the reciprocal of square of the Euclidean Distance.

$$\mathbb{M}_k(\mathbf{q}) = \arg\max_{l \in levels(t)} \sum_{i=1}^{k} \frac{1}{dist(\mathbf{q}, \mathbf{d_i})^2} \times \delta(t_i, l)$$

Weights for Afghanistan = 1/ (1.2786)**2 = 1/ 1.6348= 0.6117

Likewise table below shows the weights for each country

Country ID	CPI	Euclidean Distance	Weights	Weight X CPI
Afghanistan	1.5171	1.2786	0.6117	0.9280
Haiti	1.7999	1.4776	0.4580	0.8244
Nigeria	2.4493	1.2928	0.5983	1.4654
Egypt	2.8622	0.6737	2.2030	6.3053
Argentina	2.9961	0.5554	3.2413	9.7112
China	3.6356	0.5910	2.8634	10.4100
Brazil	3.7741	0.7227	1.9148	7.2265
Israel	5.8069	0.5869	2.9036	16.8608
U.S.A	7.1357	0.3362	8.8480	63.1370
Ireland	7.5360	0.6332	2.4939	18.7938
U.K.	7.7751	0.4126	5.8734	45.6660

Germany	8.0461	0.6438	2.4126	19.4121
Canada	8.6725	0.5915	2.8587	24.7917
Australia	8.8442	0.5644	3.1398	27.7689
Sweden	9.2985	0.6611	2.2882	21.2766
New Zealand	9.4627	0.5665	3.1165	29.4901
SUM			45.8249	304.0679

As the value of K = 16, So the value of the CPI predicted by the model will be :

Russia(CPI) = 304.0679/ 45.8249= 6.6354

e. The actual 2011 CPI for Russia was 2.4488. Which of the predictions made was the most accurate? Why do you think this was?

Ans: With reference to calculations done above, closest CPI value of Russia is **4.5891** which the was predicted by the unnormalized 3-NN model based on Euclidean distance. Smaller datasets produced most accurate results. The calculations of the data can be made better by weighted KNN and normalized values. Normalized data is uniform and consistently reliable in giving predictions.

However, in the above case when we checked results using normalized values and weighted KNN then the value of CPI is not close to 2.4488 as compared to unnormalized model. Hence, in this case unnormalized 3-NN model based on Euclidean Distance gave the closest value(4.5891) to 2.4488

Ques 4. You have been given the job of building a recommender system for a large online shop that has a stock of over 100,000 items. In this domain the behavior of customers is captured in terms of what items they have bought or not bought. For example, the following table lists the behavior of two customers in this domain for a subset of the items that at least one of the customers has bought.

ID	ITEM 107	ITEM 498	ITEM 7256	ITEM 28063	ITEM 75328
1	true	true	true	false	false
2	true	false	false	true	true

a. The company has decided to use a similarity-based model to implement the recommender system. Which of the following three similarity indexes do you think the system should be based on?

$$\text{Russell-Rao}(X,Y) = \frac{CP(X,Y)}{P}$$

$$\text{Sokal-Michener}(X,Y) = \frac{CP(X,Y) + CA(X,Y)}{P}$$

$$\text{Jaccard}(X,Y) = \frac{CP(X,Y)}{CP(X,Y) + PA(X,Y) + AP(X,Y)}$$

Ans:

Russel Rao formula is the ratio of number of co-presences and the total number of binary features. However, in some cases co-absence is important such as in medical cases. Hence Sokal Michener takes co absence (CA) into account.

Although, in retail domain in which there are so many items/products that most people haven't bought, seen, or visited the vast majority of them, and as a result, the majority of features will be co-absences. As a result, this will create a sparse matrix where most of the features have zero values. Therefore, **Jaccard similarity index** is often used to overcome these issues. The index ignores co-absences and is defined as the ratio between the number of co-presences and the total number of features, excluding those that record a co-absence between a pair of instances.

As we are building a recommender system for online shop (retail business), so we should implement recommender system based on Jaccard similarity index to recommend items to the customers.

b. What items will the system recommend to the following customer? Assume that the recommender system uses the similarity index you chose in the first part of this question and is trained on the sample dataset listed above. Also assume that the system generates recommendations for query customers by finding the customer most similar to them in the dataset and then recommending the items that this similar customer has bought but that the query customer has not bought.

	ITEM	ITEM	ITEM	ITEM	ITEM
ID	107	498	7256	28063	75328
Query	true	false	true	false	false

Ans: Step 1: First we will calculate the number of co absence, co presence, presence-absence, absence-presence between query and training dataset

Table below shows the number of co absence, co presence, presence- absence, absence-presence between query and training dataset.

		Query	
D1		PRESENT	ABSENT
	PRESENT	CP= 2	PA= 0
	ABSENT	AP= 1	CA= 2

		Query	
D2		PRESENT	ABSENT
	PRESENT	CP= 1	PA= 1
	ABSENT	AP= 2	CA= 1

1. **Using Jaccard Coefficient** to calculate similarity

$$\text{Jaccard(X,Y)} = \frac{CP(X,Y)}{CP(X,Y) + PA(X,Y) + AP(X,Y)}$$

Sim(d1,query) = 2/ (2+0 +1) = 2/ 3= **0.6667**

Sim(d2,query) = 1/ (1+1+2) = ¼= 0.25

As per the similarity metric, the higher the values, the more similar the two items are. Hence, from the calculation above we can see that query is more similar to d1.
There is only one item that the d1 has bought and query has not bought and that is item No 498. **Therefore, the system will recommend item 498 to the query customer.**

We shall also cross check similarity values using other two methods.

2. Using Russel Rao formula to calculate similarity. Russel Rao formula is the ratio of number of co-presences and the total number of binary features
   Here number of binary features= 5

$$\text{Russell-Rao(X,Y)} = \frac{CP(X,Y)}{P}$$

Sim(d1,query) = 2/ 5 = **0.4**

Sim(d2,query) = 1/ 5 = 0.2

3. Using Sokal - Michener (X,Y) to calculate similarity. Sokal Michener takes co absence (CA) into account. Here, P is the number of binary features.

$$\text{Sokal-Michener(X,Y)} = \frac{CP(X,Y) + CA(X,Y)}{P}$$

Sim(d1,query) = 2+ 2/ 5 = **0.8**

Sim(d2,query) = 1+1/ 5 = 0.4

Hence, by using the other two methods also we can observe that query is more similar to instance d1.
So, the **system will recommend item 498** irrespective of which similarity metric used.

Ques 5. You are working as an assistant biologist to Charles Darwin on the Beagle voyage. You are at the Galápagos Islands, and you have just discovered a new animal that has not yet been classified. Mr. Darwin has asked you to classify the animal using a nearest neighbor approach, and he has supplied you the following dataset of already classified animals.

ID	BIRTHS LIVE YOUNG	LAYS EGGS	FEEDS OFFSPRING OWN MILK	WARM-BLOODED	COLD-BLOODED	LAND AND WATER BASED	HAS HAIR	HAS FEATHERS	CLASS
1	true	false	true	true	false	false	true	false	mammal
2	false	true	false	false	true	true	false	false	amphibian
3	true	false	true	true	false	false	true	false	mammal
4	false	true	false	true	false	true	false	true	bird

The descriptive features of the mysterious newly discovered animal are as follows:

ID	BIRTHS LIVE YOUNG	LAYS EGGS	FEEDS OFFSPRING OWN MILK	WARM-BLOODED	COLD-BLOODED	LAND AND WATER BASED	HAS HAIR	HAS FEATHERS	CLASS
Query	false	true	false	false	false	true	false	false	?

a. A good measure of distance between two instances with categorical features is the overlap metric (also known as the hamming distance), which simply counts the number of descriptive features that have different values. Using this measure of distance, compute the distances between the mystery animal and each of the animals in the animal dataset.

Ans: Hamming distance is used for categorical features. If the the value of query and instance is same then distance will be zero else it will be 1.

The table below calculate the overlap metric by counting the number of features that have different values.

ID	BIRTHS LIVE YOUNG	LAYS EGGS	FEEDS OFFSPRING OWN MILK	WARM-BLOODED	COLD-BLOODED	LAND AND WATER BASED	HAS HAIR	HAS FEATHERS	CLASS
1	true	false	true	true	false	false	true	false	mammal
2	false	true	false	false	true	true	false	false	amphibian
3	true	false	true	true	false	false	true	false	mammal
4	false	true	false	true	false	true	false	true	bird

The descriptive features of the mysterious newly discovered animal are as follows:

ID	BIRTHS LIVE YOUNG	LAYS EGGS	FEEDS OFFSPRING OWN MILK	WARM-BLOODED	COLD-BLOODED	LAND AND WATER BASED	HAS HAIR	HAS FEATHERS	CLASS
Query	false	true	false	false	false	true	false	false	?

(Query,ID)	Birth live young	Lay eggs	Feed offspring	Warm blooded	Cold Blooded	Land and Water based	Has Hair	Has features	Class	Number of Overlap metric (distance)
Query,1	1	1	1	1	0	1	1	0	Mammal	6
Query,2	0	0	0	0	1	0	0	0	Amphibian	1
Query,3	1	1	1	1	0	1	1	0	Mammal	6
Query,4	0	0	0	1	0	0	0	1	Bird	2

b. If you used a 1-NN model, what class would be assigned to the mystery animal?
Ans: when k =1 , then instance d2 is the nearest neighbour to the query. Hence, the model will predict the **class= Amphibian**

c. If you used a 4-NN model, what class would be assigned to the mystery animal? Would this be a good value for k for this dataset?
Ans: when k = 4 then the query is nearest all the instances in the dataset as the value of k is equal to the number of records in the dataset.
These instances have the following class:

- D_1 have target value = Mammal
- D_2 have target value = Amphibian
- D_3 have target value = Mammal
- D_4 have target value = Bird

Majority of the value belongs to class= Mammal, hence if the k= 4 then the 4-NN model will return the **class= Mammal.**
If the value of k is equal to number of instances in the dataset, then it would include all the instances irrespective of Euclidean distance. As a result, the model will simply assign any query to the majority of the class in the dataset, like in the above case it is mammal. This type of case lead of underfitting where the values are highly biased.

Ques 6. You have been asked by a San Francisco property investment company to create a predictive model that will generate house price estimates for properties they are considering purchasing as rental properties. The table below lists a sample of properties that have recently been sold for rental in the city. The descriptive features in this dataset are SIZE (the property size in square feet) and RENT (the estimated monthly rental value of the property in dollars). The target feature, PRICE, lists the prices that these properties were sold for in dollars.

ID	SIZE	RENT	PRICE
1	2,700	9,235	2,000,000
2	1,315	1,800	820,000
3	1,050	1,250	800,000
4	2,200	7,000	1,750,000
5	1,800	3,800	1,450,500
6	1,900	4,000	1,500,500
7	960	800	720,000

a. Create a k-d tree for this dataset. Assume the following order over the features: RENT then SIZE.

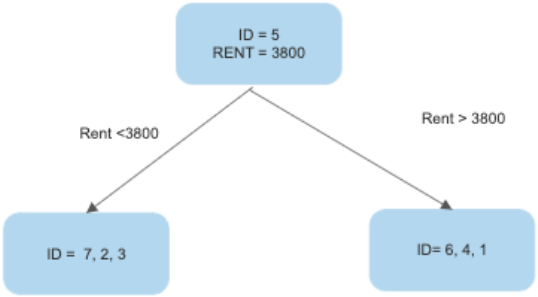
Ans:

Step 1: Sorting the Rent values and finding the median

Rent
800
1250
1800
3800
4000
7000
9235

Median is : 3800. Hence the median value 3800 is for instance d5.

Step 2: Creating the split based on Rent median value. Values less than median will be on the left side and values greater than median will be on the right side

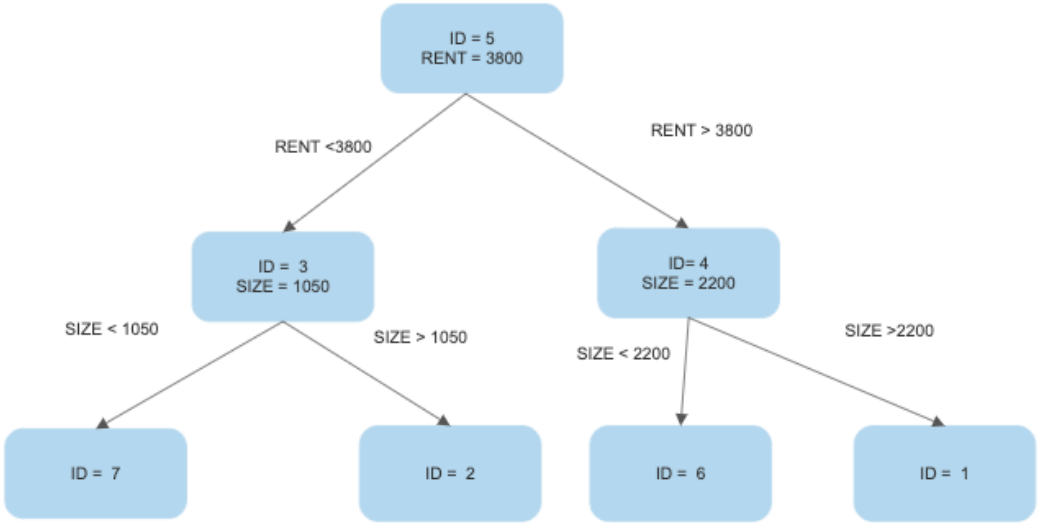


Step 3: Now sorting the Size feature values for instance ID = 7,2,3 and finding the median.
Median value is 1050 and this value is at instance d3

Step 4: Now sorting the Size feature for instance ID = 6,4,1 and finding the median.
Median value is 2200 and this value is at instance d4

Step 5: Creating the split based on size feature. Values less than median will be on the left side and values greater than median will be on the right side.

Below is the final K-D tree for this dataset.



b. Using the k-d tree that you created in the first part of this question, find the nearest neighbor to the following query: SIZE = 1,000, RENT = 2,200.

Ans: Initially , the k-d tree will find the nearest neigbour by descending the tree. For this query, as the value of rent = 2200 which is less than the median value of rent (3800). So the tree will search in the left sub tree. Now with reference to the size, its value is less than the median value of size which is 1050. So it will descend towards the left branch i.e. till d7 and the current best variable is set to the instance d7. The current best distance is calculated between query and instance d7.

Table below shows the Euclidean Distance between d7 and query

Instance	Size	Rent	Euclidean Distance
D7	960	800	1400.57131

Now, the algorithm will ascend the tree. The node that algorithm will first face is the node that stores instance d3. The function will calculate the Euclidean distance between query and d3.
Table below shows the Euclidean Distance between d3 and query

Instance	Size	Rent	Euclidean Distance
D3	1050	1250	951.31488

As the Euclidean distance is less as compared to instance d7. So the current best variable is updated to d3 and current best distance is updated to 951.31488.

As, the difference between the splitting feature value i.e. size= 1050 and the query Size= 1000 is less, so the algorithm will descend the other branch of the tree from the node. The function will then calculate the Euclidean distance between query and d2.

Instance	Size	Rent	Euclidean Distance
D2	1315	1800	509.141434

As, the Euclidean distance is less ,so the current best variable is updated to instance d2 and Euclidean distance is updated to 509.141434.

The algorithm will then ascend the tree and as it has visited all nodes on the sub tree it will not check for nodes closer than the best variable until it gets back to the root.
In this current situation, the distance between query and root node d3 is greater than the current best variable, so current distance and current variable are not updated when the algorithm reach the root node.
Also, the difference between the splitting feature RENT =3800 and the query feature value RENT = 2200 is greater than the current best distance, the algorithm will not descend to the branches (i.e. it will prune the branches). As a result, the algorithm will return instance d2 as the nearest neighbour to the query.
So, **the model will predict the value of price = 820,000** for the query.

Que 6.Could you implement your own KNN function from scratch (without calling any existing classifier packages)? Your own KNN function must have following parameters to tune: n_neighbors, weights. Can you compare your own classifier with sklearn neighbors.KNeighborsClassifier in terms performance, complexity, etc. Data usage: datasets.load_iris() from sklearn.

```
from sklearn.datasets import load_iris #importing the iris dataset from SKlearn
import seaborn as sns
import pandas as pd
import numpy as np
from scipy.stats import mode
from sklearn.preprocessing import StandardScaler

```

EDA

```
data = load_iris()
df = pd.DataFrame(data=data.data, columns=data.feature_names)
df1 = pd.DataFrame(load_iris().target, columns=['Species'])
df = pd.concat([df,df1], axis=1)
df.head()
```

	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)	Species
0	5.1	3.5	1.4	0.2	0
1	4.9	3.0	1.4	0.2	0
2	4.7	3.2	1.3	0.2	0
3	4.6	3.1	1.5	0.2	0
4	5.0	3.6	1.4	0.2	0

```
df.columns= ["sepal_length", "sepal_width","petal_length","petal_width","species"] #changing the column names
```

```
df.info()
# checking the data types
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 150 entries, 0 to 149
Data columns (total 5 columns):
 #   Column        Non-Null Count  Dtype  
---  ------        --------------  -----  
 0   sepal_length  150 non-null    float64
 1   sepal_width   150 non-null    float64
 2   petal_length  150 non-null    float64
 3   petal_width   150 non-null    float64
 4   species       150 non-null    int64  
dtypes: float64(4), int64(1)
memory usage: 6.0 KB
```

```
df.isnull().sum() # checking for null values
```

```
sepal_length    0
sepal_width     0
petal_length    0
petal_width     0
species         0
dtype: int64
```

```
df.describe()
```

	sepal_length	sepal_width	petal_length	petal_width	species
count	150.000000	150.000000	150.000000	150.000000	150.000000
mean	5.843333	3.057333	3.758000	1.199333	1.000000
std	0.828066	0.435866	1.765298	0.762238	0.819232
min	4.300000	2.000000	1.000000	0.100000	0.000000
25%	5.100000	2.800000	1.600000	0.300000	0.000000
50%	5.800000	3.000000	4.350000	1.300000	1.000000
75%	6.400000	3.300000	5.100000	1.800000	2.000000
max	7.900000	4.400000	6.900000	2.500000	2.000000

```
1 list(data.target_names)
```

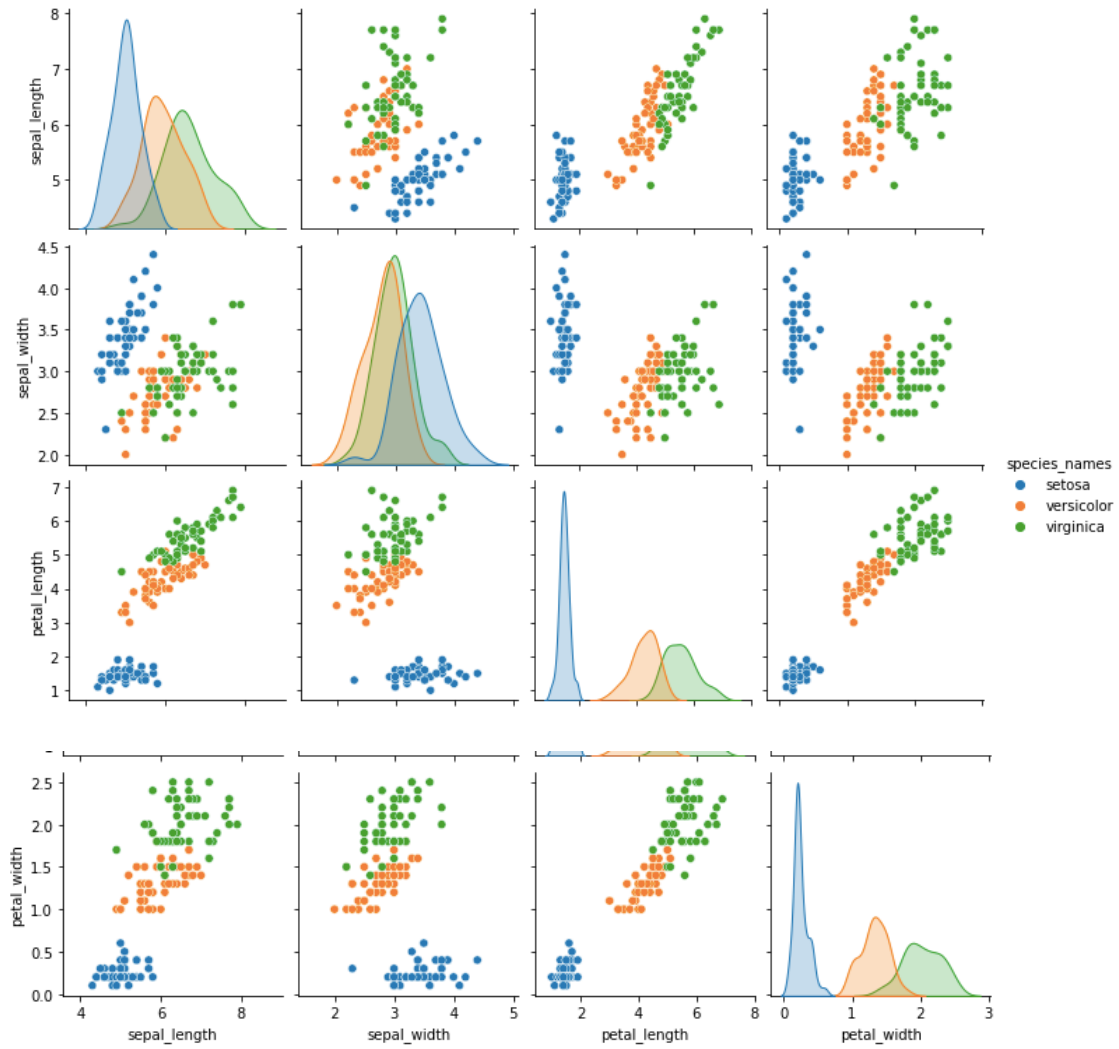
```
['setosa', 'versicolor', 'virginica']
```

```
1 species_map={0:'setosa',1:'versicolor',2:'virginica'}
2 df["species_names"]= df["species"].apply(lambda x:species_map.get(x))
3 df= df.drop('species',axis=1)
4 df.species_names.value_counts()
```

```
setosa        50
versicolor    50
virginica     50
Name: species_names, dtype: int64
```

```
1 sns.pairplot(df,hue="species_names", kind="scatter")
```

```
<seaborn.axisgrid.PairGrid at 0x7fde250747f0>
```



Creating the KNN Function for K=4

```
X = data.data
y = data.target
```

```
from sklearn.model_selection import train_test_split
```

```
class KNN_Classifier:

    def __init__( self, K ) :  # initialising Value of K neighbours

        self.K = K

    # Function to store training set

    def fit( self, X_train, Y_train ) :

        self.X_train = X_train

        self.Y_train = Y_train

        # no_of_training_examples, no_of_features

        self.m, self.n = X_train.shape

    # Function for prediction

    def predict( self, X_test ) :

        self.X_test = X_test

        # no_of_test_examples, no_of_features

        self.m_test, self.n = X_test.shape

        # initialize Y_predict

```

```
        Y_predict = np.zeros( self.m_test )

        for i in range( self.m_test ) :

            x = self.X_test[i]



            neighbors = np.zeros( self.K )

            neighbors = self.find_neighbors( x )

            # majority class in K neighbors

            Y_predict[i] = mode( neighbors )[0][0]

        return Y_predict

    # Function to find the K nearest neighbors to current test example

    def find_neighbors( self, x ) :

        # calculate all the euclidean distances between current
        # test example x and training set X_train

        euclidean_distances = np.zeros( self.m )

        for i in range( self.m ) :

            d = self.euclidean( x, self.X_train[i] )

            euclidean_distances[i] = d

        # sort Y_train according to euclidean_distance_array and
        # store into Y_train_sorted
```

```
        inds = euclidean_distances.argsort()

        Y_train_sorted = self.Y_train[inds]


        return Y_train_sorted[:self.K]

    # Function to calculate euclidean distance

    def euclidean( self, x, x_train ) :

        return np.linalg.norm( x - x_train )



X_train1, X_test1, Y_train, Y_test = train_test_split( X, y, test_size = 0.3, random_state = 0 )

scaler = StandardScaler()
X_train = scaler.fit_transform(X_train1)
X_test = scaler.transform(X_test1)

# Model training

model = KNN_Classifier( K = 4)

model.fit( X_train, Y_train )



# Prediction on test set

Y_pred = model.predict( X_test )
```

```

# measure performance

correctly_classified = 0


# counter

count = 0

for count in range( np.size( Y_pred ) ) :

    if Y_test[count] == Y_pred[count] :

        correctly_classified = correctly_classified + 1


print( "The Accuracy of the model is:", round(( correctly_classified / count ) * 100.0,2) )

```

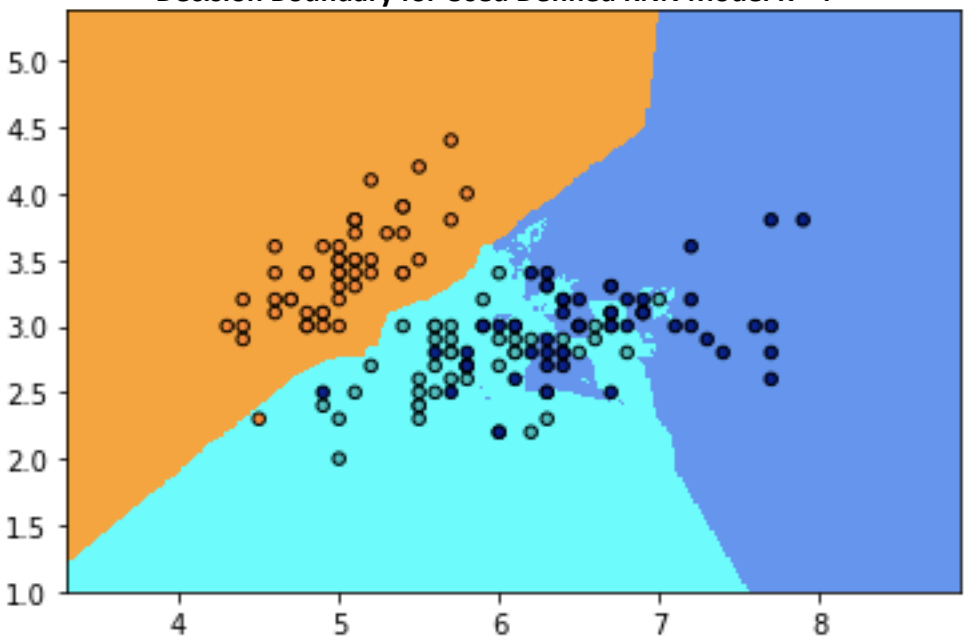
```
The Accuracy of the model is: 100.0
```

Confusion Matrix for Weighted KNN model

```
def confusion_mat(actual, predicted):

    # extract the different classes
    classes = np.unique(actual)

    # initialize the confusion matrix
    confmat = np.zeros((len(classes), len(classes)))

    # loop across the different combinations of actual / predicted classes
    for i in range(len(classes)):
        for j in range(len(classes)):

            # count the number of instances in each combination of actual / predicted classes
            confmat[i, j] = np.sum((actual == classes[i]) & (predicted == classes[j]))

    return confmat

# sample data
actual = Y_test
predicted = Y_pred

# confusion matrix
print(confusion_mat(actual, predicted))
```

```
[[16.  0.  0.]
 [ 0. 17.  1.]
 [ 0.  0. 11.]]
```

Decision Boundary for Used Defined KNN Model K= 4



Implementing the KNN Model using Sklearn

```
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score
from sklearn.metrics import confusion_matrix
from sklearn.metrics import classification_report
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
```

```
from sklearn.datasets import load_iris
data = load_iris()
X = data.data
y = data.target
```

```
knn= KNeighborsClassifier(n_neighbors = 4)

X_train_sklearn, X_test_sklearn, Y_train_sklearn, Y_test_sklearn = train_test_split( X, y, test_size = 0.3, r

scaler = StandardScaler()
X_train_1 = scaler.fit_transform(X_train_sklearn)
X_test_1 = scaler.transform(X_test_sklearn)


```

```
knn.fit(X_train_1, Y_train_sklearn)
y_pred= knn.predict(X_test_1)

Accuracy= accuracy_score(Y_test_sklearn,y_pred)

print("The accuracy of the model is :", round(Accuracy*100.0,2))
```

```
The accuracy of the model is : 97.78
```

Confusion Matrix

```
confusion_matrix(Y_test_sklearn,y_pred)
```

```
array([[16,  0,  0],
       [ 0, 17,  1],
       [ 0,  0, 11]])
```

Classification Report

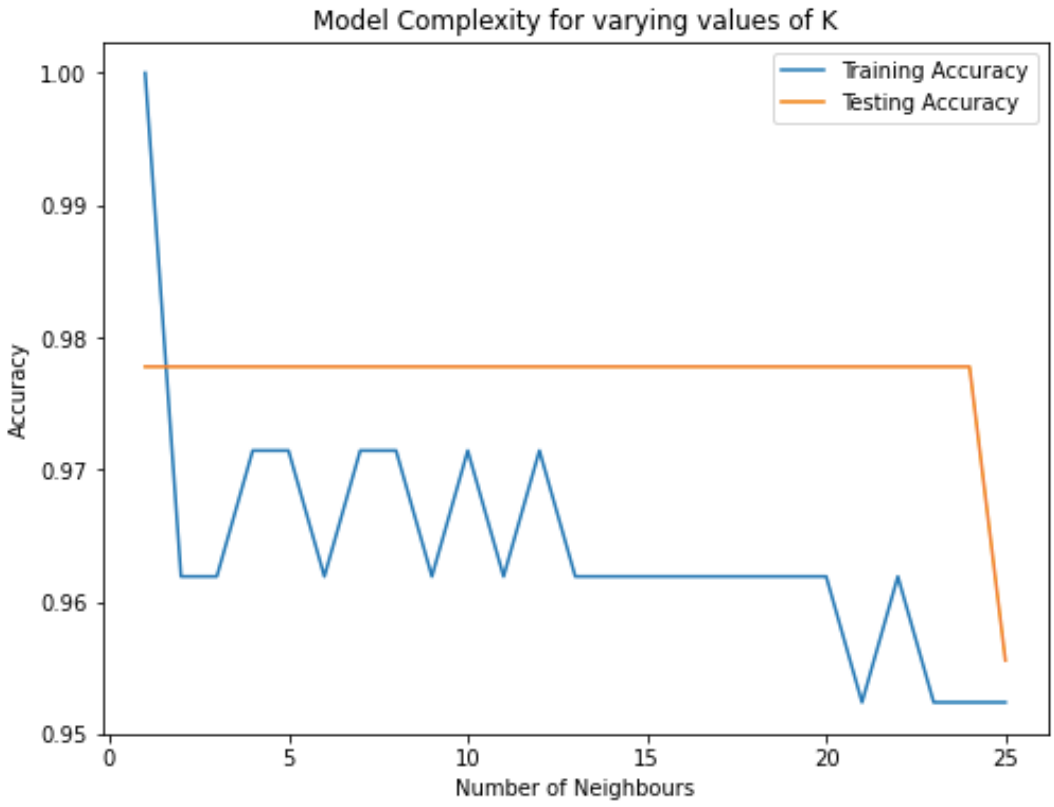
```
print(classification_report(Y_test_sklearn, y_pred))
```

```
              precision    recall  f1-score   support

           0       1.00      1.00      1.00        16
           1       1.00      0.94      0.97        18
           2       0.92      1.00      0.96        11

    accuracy                           0.98        45
   macro avg       0.97      0.98      0.98        45
weighted avg       0.98      0.98      0.98        45
```

Model Complexity

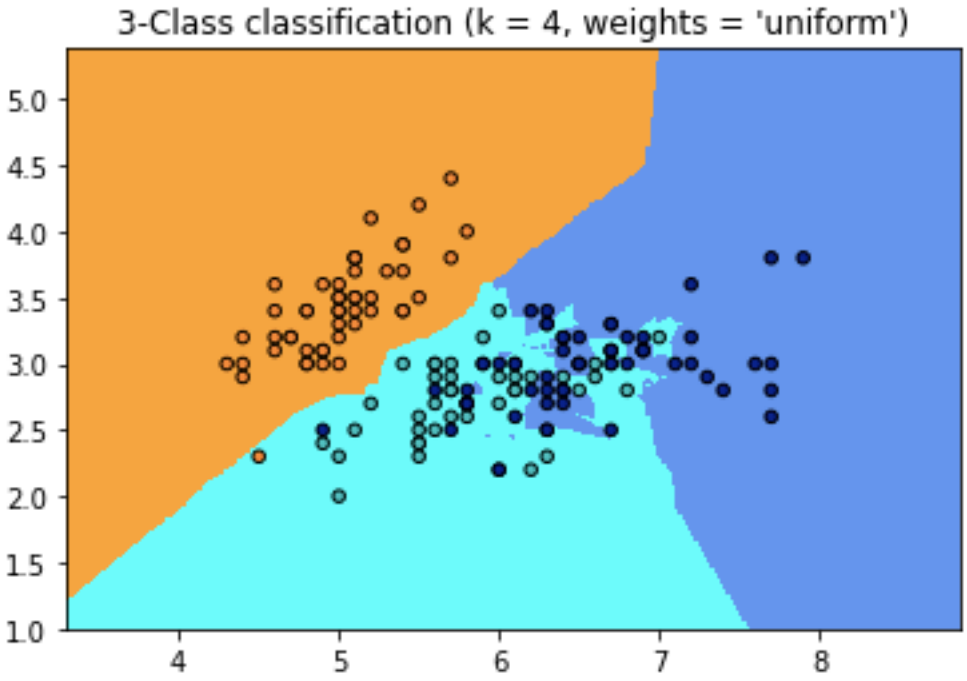
```
train_accuracies = {}
test_accuracies = {}
import numpy as np
neighbors = np.arange(1, 26)
for neighbor in neighbors:
    knn = KNeighborsClassifier(n_neighbors=neighbor)
    knn.fit(X_train_sklearn, Y_train_sklearn)
    train_accuracies[neighbor] = knn.score(X_train_sklearn, Y_train_sklearn)
    test_accuracies[neighbor] = knn.score(X_test_sklearn, Y_test_sklearn)

```

```
import matplotlib.pyplot as plt
plt.figure(figsize=(8, 6))
plt.title("Model Complexity for varying values of K")
plt.plot(neighbors, train_accuracies.values(), label="Training Accuracy")
plt.plot(neighbors, test_accuracies.values(), label="Testing Accuracy")
plt.legend()
plt.xlabel("Variance")
plt.ylabel("Bias")
plt.show()
```



Decision Boundaries

```
# Create color maps
cmap_light = ListedColormap(['orange', 'cyan', 'cornflowerblue'])
cmap_bold = ListedColormap(['darkorange', 'c', 'darkblue'])

for weights in ['uniform', 'distance']:
    # we create an instance of Neighbours Classifier and fit the data.
    clf = neighbors.KNeighborsClassifier(n_neighbors, weights=weights)
    clf.fit(X, y)

    # Plot the decision boundary. For that, we will assign a color to each
    # point in the mesh [x_min, x_max]x[y_min, y_max].
    x_min, x_max = X[:, 0].min() - 1, X[:, 0].max() + 1
    y_min, y_max = X[:, 1].min() - 1, X[:, 1].max() + 1
    xx, yy = np.meshgrid(np.arange(x_min, x_max, h),
                         np.arange(y_min, y_max, h))
    Z = clf.predict(np.c_[xx.ravel(), yy.ravel()])

    # Put the result into a color plot
    Z = Z.reshape(xx.shape)
    plt.figure()
    plt.pcolormesh(xx, yy, Z, cmap=cmap_light)

    # Plot also the training points
    plt.scatter(X[:, 0], X[:, 1], c=y, cmap=cmap_bold,
                edgecolor='k', s=20)
    plt.xlim(xx.min(), xx.max())
    plt.ylim(yy.min(), yy.max())
    plt.title("3-Class classification (k = %i, weights = '%s')"
              % (n_neighbors, weights))

plt.show()
```



Observation

From the above function we can see that when K=4 and random_state= 0 then the accuracy is quite close in both the methods (i.e. using user defined function and sklearn). Accuracy is 100% in KNN function and Accuracy is 98% in KNN Skearn. We shall further cross check the accuracy using K-Fold Cross validation.

Creating the 5 Fold Cross_Val Function

```
class KNN:
    def __init__(self, K):
        self.K = K
        self.X_train = None
        self.y_train = None

    def fit(self, X_train, y_train):
        self.X_train = X_train
        self.y_train = y_train

    def predict_instance(self, test_instance):
        inputs = self.X_train.copy()
        # calculate distance between all training points and given test_point
        inputs['distance'] = np.linalg.norm(inputs.values-test_instance.values, axis=1)

        # concatenate inputs and labels before sorting the distances
        inputs = pd.concat([inputs, self.y_train], axis=1)

        # sort based on distance
        inputs = inputs.sort_values('distance', ascending=True)

        # pick k neighbors
        neighbors = inputs.head(self.K)

        # getting target from dataframe column
        classes = neighbors['Species'].tolist()

        from collections import Counter

        # create counter of labels
        majority_count = Counter(classes)
```

```
        return majority_count.most_common(1).pop()[0]


    def predict(self, X_test):
        predictions = np.zeros(X_test.shape[0])
        # we want out index to be start from 0
        X_test.reset_index(drop=True, inplace=True)
        for index, row in X_test.iterrows():
            predictions[index] = self.predict_instance(row)
        return predictions

def cross_validation(n, k, data, n_neighbors):
    """
    n : # iterations
    k : k-fold size
    data: training data
    n_neighbors: k in knn
    """
    accuracies = []

    for _ in range(0, n):
        # data shuffle
        data.sample(frac=1)

        fold=int(data.shape[0]/k)

        for j in range(k):
            test = data[j*fold:j*fold+fold]
            train = data[~data.index.isin(test.index)]
            X_train, y_train = train.drop('Species', axis=1), train['Species']
            X_test, y_test = test.drop('Species', axis=1), test['Species']

```

```
            knn = KNN(n_neighbors)
            knn.fit(X_train, y_train)

            predictions = knn.predict(X_test)
            true_values = y_test.to_numpy()
            accuracy = np.mean(predictions == true_values)

            accuracies.append(accuracy)
    return np.array(accuracies).mean()
```

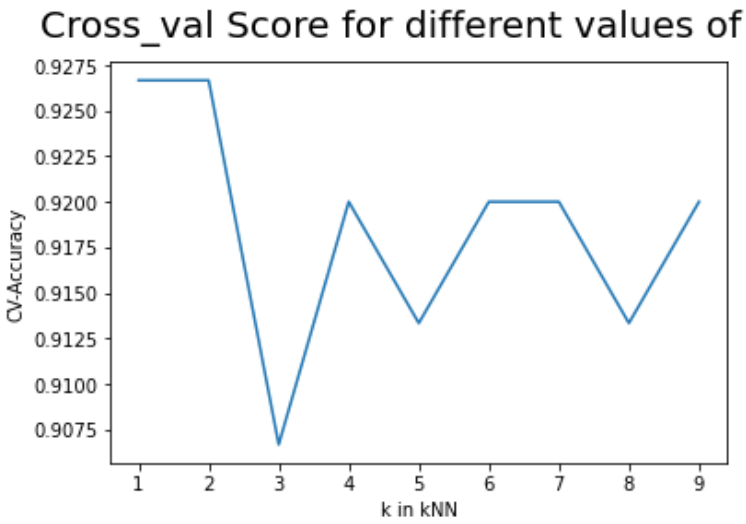
```
k_values = np.arange(1, 10)
cross_validation_fold = 5
accuracies = []
```

```
for k in k_values:
    # run cross-validation with given neighbor size k
    accuracy = cross_validation(1, cross_validation_fold, df, k)
    accuracies.append(accuracy)
print(accuracies)
```

```
[0.9266666666666665, 0.9266666666666665, 0.9066666666666666, 0.9199999999999999, 0.9133333333333333, 0.9199999999999999, 0.9200000000000002, 0.9133333333333333, 0.9200000000000002]
```

```
import matplotlib.pyplot as plt
fig = plt.figure()
plt.plot(k_values, accuracies)
plt.xlabel('k in kNN')
plt.ylabel('CV-Accuracy')
fig.suptitle('Cross_val Score for different values of K', fontsize=20)
```

```
Text(0.5, 0.98, 'Cross_val Score for different values of K')
```



Implementing the 5 Fold Cross_Val through Sklearn

```
accuracy_list=[]
def cross_val(K):
    from sklearn.model_selection import cross_val_score, KFold
    kf = KFold(n_splits=5, shuffle=True, random_state=0)
    from sklearn.datasets import load_iris
    data = load_iris()
    X = data.data
    y = data.target
    KNN = KNeighborsClassifier(K)

    cv_results = cross_val_score(KNN, X, y, cv=kf)


    print("For K= {0}, the 5- fold cross val score is given below:".format(K))

    print("scores are",cv_results)
    print("-----------------------------------------------------------------")

    print("Mean is {0}, standard deviation is {1}".format(np.mean(cv_results), np.std(cv_results)))
    print("-----------------------------------------------------------------")
    print("Confidence Interval is",np.quantile(cv_results, [0.025, 0.975]))
    print("-----------------------------------------------------------------")

    print("\n")
    print("*******")
    return np.mean(cv_results)


for i in range(1,10):
    x= cross_val(i)
    accuracy_list.append(x)
```

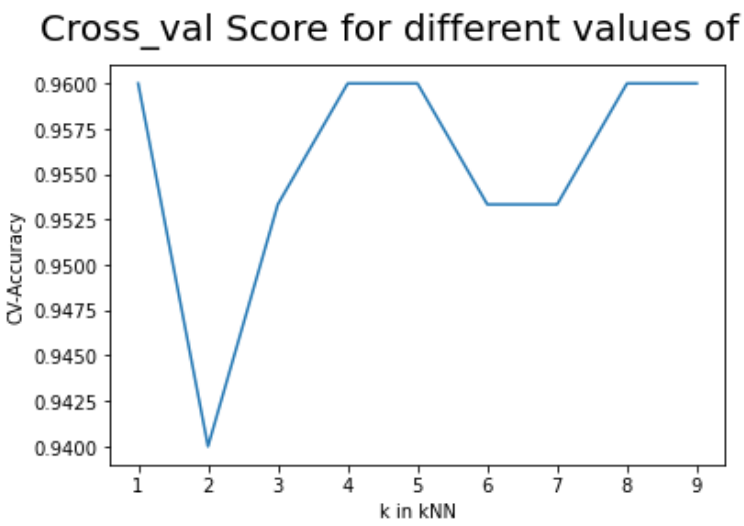
```
For K= 1, the 5- fold cross val score is given below:
scores are [1.         0.86666667 1.         1.         0.93333333]
-----------------------------------------------------------------
Mean is 0.96, standard deviation is 0.053333333333333316
-----------------------------------------------------------------
Confidence Interval is [0.87333333 1.        ]
-----------------------------------------------------------------


*******
For K= 2, the 5- fold cross val score is given below:
scores are [0.96666667 0.83333333 1.         0.96666667 0.93333333]
-----------------------------------------------------------------
Mean is 0.9400000000000001, standard deviation is 0.0573488351136175
-----------------------------------------------------------------
Confidence Interval is [0.84333333 0.99666667]
-----------------------------------------------------------------


*******
For K= 3, the 5- fold cross val score is given below:
scores are [0.96666667 0.86666667 1.         1.         0.93333333]
-----------------------------------------------------------------
Mean is 0.9533333333333334, standard deviation is 0.04988876515698587
-----------------------------------------------------------------
Confidence Interval is [0.87333333 1.        ]
-----------------------------------------------------------------
```

```
*******
For K= 4, the 5- fold cross val score is given below:
scores are [1.         0.86666667 1.         1.         0.93333333]
-----------------------------------------------------------------
Mean is 0.96, standard deviation is 0.053333333333333316
-----------------------------------------------------------------
Confidence Interval is [0.87333333 1.        ]
-----------------------------------------------------------------


*******
For K= 5, the 5- fold cross val score is given below:
scores are [0.96666667 0.9        1.         1.         0.93333333]
-----------------------------------------------------------------
Mean is 0.96, standard deviation is 0.038873012632301994
-----------------------------------------------------------------
Confidence Interval is [0.90333333 1.        ]
-----------------------------------------------------------------


*******
For K= 6, the 5- fold cross val score is given below:
scores are [1.         0.83333333 1.         1.         0.93333333]
-----------------------------------------------------------------
Mean is 0.9533333333333334, standard deviation is 0.06531972647421808
-----------------------------------------------------------------
Confidence Interval is [0.84333333 1.        ]
-----------------------------------------------------------------
```

```
*******
For K= 7, the 5- fold cross val score is given below:
scores are [1.         0.86666667 1.         1.         0.9       ]
--------------------------------------------------------------------
Mean is 0.9533333333333334, standard deviation is 0.0581186525805423
--------------------------------------------------------------------
Confidence Interval is [0.87 1.  ]
--------------------------------------------------------------------


*******
For K= 8, the 5- fold cross val score is given below:
scores are [1.         0.86666667 1.         0.96666667 0.96666667]
--------------------------------------------------------------------
Mean is 0.96, standard deviation is 0.04898979485566354
--------------------------------------------------------------------
Confidence Interval is [0.87666667 1.        ]
--------------------------------------------------------------------


*******
For K= 9, the 5- fold cross val score is given below:
scores are [1.         0.9        1.         0.96666667 0.93333333]
--------------------------------------------------------------------
Mean is 0.96, standard deviation is 0.038873012632301994
--------------------------------------------------------------------
Confidence Interval is [0.90333333 1.        ]
--------------------------------------------------------------------


*******
```

```
k_values= [1,2,3,4,5,6,7,8,9]
fig2 = plt.figure()
plt.plot(k_values, accuracy_list)
plt.xlabel('k in kNN')
plt.ylabel('CV-Accuracy')
fig2.suptitle('Cross_val Score for different values of K', fontsize=20)
```

```
Text(0.5, 0.98, 'Cross_val Score for different values of K')
```



Observation

From the 5 fold- cross validation score calculated above we can see in average cross validation function accuracy is 92% although in cross val using sklearn accuracy was 96% . The sklearn cross val score showed a fluctuating trend whereas the cross val function showed a decreasing trend.

Creating the Weighted KNN Model for k = 4

```
class WeightedKNN:

    def __init__( self, K ) :  # initialising Value of K neighbours

        self.K = K

    # Function to store training set

    def fit( self, X_train, Y_train ) :

        self.X_train = X_train

        self.Y_train = Y_train

        # no_of_training_examples, no_of_features

        self.m, self.n = X_train.shape

    # Function for prediction

    def predict( self, X_test ) :

        self.X_test = X_test

        # no_of_test_examples, no_of_features

        self.m_test, self.n = X_test.shape

        # initialize Y_predict
```

```
        Y_predict = np.zeros( self.m_test )

        for i in range( self.m_test ) :

            x = self.X_test[i]

            # find the K nearest neighbors from current test example

            neighbors = np.zeros( self.K )

            neighbors = self.find_neighbors( x )

            # determining the class by max weights
            weight_1 = 0
            weight_2 = 0
            weight_0 = 0
            for t in neighbors:  # finding the max weights in different classes
                if t[0] == 1:
                    weight_1 += t[1]
                elif t[0] ==2:
                    weight_2 += t[1]
                else:
                    weight_0 += t[1]

            weight_dict= {weight_1 : 1, weight_2 : 2, weight_0 : 0} #creating dictionary

            max_weight= max(weight_dict.keys()) #finding the key with max weight


            Y_predict[i] = weight_dict[max_weight]  # class with max weight

```

```
        return Y_predict

    # Function to find the K nearest neighbors to current test example

    def find_neighbors( self, x ) :

        # calculate all the euclidean distances between current
        # test example x and training set X_train

        weights = np.zeros( self.m )

        for i in range( self.m ) :

            d = self.weighted( x, self.X_train[i] )

            weights[i] = d

        # sort Y_train according to weights and
        # store into Y_train_sorted in reverse order i.e. max weight(closest data point)

        inds = weights.argsort()
        inds= inds[::-1]
        weights= weights[::-1]

        Y_train_reversed = self.Y_train[inds]

        weight_target = np.column_stack((Y_train_reversed,weights))
```

```
        return weight_target[:self.K]



    def weighted( self, x, x_train ) :

        dist= np.linalg.norm( x - x_train ) # calculating the euclidean distance

        square_distance= dist**2

        return 1/square_distance

 # Splitting dataset into train and test set

from sklearn.datasets import load_iris
data = load_iris()
X = data.data
y = data.target

X_train1, X_test1, Y_train, Y_test = train_test_split( X, y, test_size = 0.3, random_state = 0 )

scaler = StandardScaler()
X_train = scaler.fit_transform(X_train1)
X_test = scaler.transform(X_test1)


# Model training

model = WeightedKNN( K = 4)
```

```
model.fit( X_train, Y_train )



# Prediction on test set

Y_pred = model.predict( X_test )


# measure performance

correctly_classified = 0


# counter

count = 0

for count in range( np.size( Y_pred ) ) :

    if Y_test[count] == Y_pred[count] :

        correctly_classified = correctly_classified + 1


print( "The accuracy of the model using weights is:  ", round(correctly_classified / count,2 ) * 100.0 )


```

```
The accuracy of the model using weights is:   93.0
```

Confusion Matrix for Weighted KNN model

```
def weight_confmat(actual, predicted):

    # extract the different classes
    classes = np.unique(actual)

    # initialize the confusion matrix
    confmat = np.zeros((len(classes), len(classes)))

    # loop across the different combinations of actual / predicted classes
    for i in range(len(classes)):
        for j in range(len(classes)):

            # count the number of instances in each combination of actual / predicted classes
            confmat[i, j] = np.sum((actual == classes[i]) & (predicted == classes[j]))

    return confmat

# sample data
actual = Y_test
predicted = Y_pred

# confusion matrix
print(weight_confmat(actual, predicted))
```

```
[[16.  0.  0.]
 [ 0. 15.  3.]
 [ 0.  1. 10.]]
```

Implementing weighted KNN using Sklearn

```
from sklearn.datasets import load_iris
data = load_iris()
X = data.data
y = data.target

knn= KNeighborsClassifier(n_neighbors = 4, weights= 'distance')

X_train1, X_test1, y_train, y_test = train_test_split(X, y, test_size=0.3,stratify=y,random_state= 30)
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train1)
X_test = scaler.transform(X_test1)
```

```
knn.fit(X_train, y_train)
y_pred= knn.predict(X_test)

Accuracy= accuracy_score(y_test,y_pred)

print("The accuracy of the model is :", round(Accuracy*100.0,2))
```

```
The accuracy of the model is : 91.11
```

Confusion Matrix

```
confusion_matrix(y_test,y_pred)
```

```
array([[15,  0,  0],
       [ 0, 14,  1],
       [ 0,  3, 12]])
```

Observation

From the weighted KNN created above and by using Sklearn , we can see that for k=4 the accuracy of weighted KNN is quite close. Using KNN Sklearn accuracy is 91% whereas user defined Weighted KNN accuracy is 93%.This nominal difference might be possible because of rounding differences.

Hyper-Parameter Tuning

First Tuning with different values of K

```
class GridSearchCV:

    def __init__( self, K ) :  # initialising Value of K neighbours

        self.K = K

    # Function to store training set

    def fit( self, X_train, Y_train ) :

        self.X_train = X_train

        self.Y_train = Y_train

        # no_of_training_examples, no_of_features

        self.m, self.n = X_train.shape

    # Function for prediction

    def predict( self, X_test ) :

        self.X_test = X_test

        # no_of_test_examples, no_of_features

        self.m_test, self.n = X_test.shape

        # initialize Y_predict
```

```
        Y_predict = np.zeros( self.m_test )

        for i in range( self.m_test ) :

            x = self.X_test[i]

            # find the K nearest neighbors from current test example

            neighbors = np.zeros( self.K )

            neighbors = self.find_neighbors( x )

            # most frequent class in K neighbors

            Y_predict[i] = mode( neighbors )[0][0]

        return Y_predict

    # Function to find the K nearest neighbors to current test example

    def find_neighbors( self, x ) :

        # calculate all the euclidean distances between current
        # test example x and training set X_train

        euclidean_distances = np.zeros( self.m )

        for i in range( self.m ) :

            d = self.euclidean( x, self.X_train[i] )

            euclidean_distances[i] = d

        # sort Y_train according to euclidean_distance_array and
        # store into Y_train_sorted
```

```
        inds = euclidean_distances.argsort()

        Y_train_sorted = self.Y_train[inds]


        return Y_train_sorted[:self.K]

    # Function to calculate euclidean distance

    def euclidean( self, x, x_train ) :

        return np.linalg.norm( x - x_train )


from sklearn.datasets import load_iris

data = load_iris()
X = data.data
y = data.target

X_train1, X_test1, Y_train, Y_test = train_test_split( X, y, test_size = 0.3, random_state = 0 )

scaler = StandardScaler()
X_train = scaler.fit_transform(X_train1)
X_test = scaler.transform(X_test1)



# Model training

for i in range(1,10):
    model = GridSearchCV( K = i )

    model.fit( X_train, Y_train )
```

```
    model.fit( X_train, Y_train )



    # Prediction on test set

    Y_pred = model.predict( X_test )


    # measure performance

    correctly_classified = 0

    # counter

    count = 0

    for count in range( np.size( Y_pred ) ) :

        if Y_test[count] == Y_pred[count] :

            correctly_classified = correctly_classified + 1


    print( "Accuracy on test set by our model for k = {0} is {1}:  ".format(i,( correctly_classified / count )
    print("----------------------------------------------------")
```

```
Accuracy on test set by our model for k = 1 is 95.45454545454545:
-----------------------------------------------------
Accuracy on test set by our model for k = 2 is 97.72727272727273:
-----------------------------------------------------
Accuracy on test set by our model for k = 3 is 100.0:
-----------------------------------------------------
Accuracy on test set by our model for k = 4 is 100.0:
-----------------------------------------------------
Accuracy on test set by our model for k = 5 is 100.0:
-----------------------------------------------------
Accuracy on test set by our model for k = 6 is 100.0:
-----------------------------------------------------
Accuracy on test set by our model for k = 7 is 100.0:
-----------------------------------------------------
Accuracy on test set by our model for k = 8 is 100.0:
-----------------------------------------------------
Accuracy on test set by our model for k = 9 is 100.0:
-----------------------------------------------------
```

Creating Function for Hyperparameter Tuning for different values of neighbors and weights

```
:
```

```python
class GridSearchCV1:

    def __init__( self, K, W) :  # initialising Value of K neighbours

        self.K = K
        self.W= W

    # Function to store training set

    def fit( self, X_train, Y_train ) :

        self.X_train = X_train

        self.Y_train = Y_train

        # no_of_training_examples, no_of_features

        self.m, self.n = X_train.shape

    # Function for prediction

    def predict( self, X_test ) :

```

```
        self.X_test = X_test

        # no_of_test_examples, no_of_features

        self.m_test, self.n = X_test.shape

        # initialize Y_predict

        Y_predict = np.zeros( self.m_test )

        if self.W== "uniform":

            for i in range( self.m_test ) :

                x = self.X_test[i]

                # find the K nearest neighbors from current test example

                neighbors = np.zeros( self.K )

                neighbors = self.find_neighbors( x )

                # most frequent class in K neighbors

                Y_predict[i] = mode( neighbors )[0][0]

            return Y_predict

        elif self.W== "distance":
            for i in range( self.m_test ) :

                x = self.X_test[i]
```

```
                # find the K nearest neighbors from current test example

                neighbors = np.zeros( self.K )

                neighbors = self.find_neighbors( x )

                # determining the class by max weights
                weight_1 = 0
                weight_2 = 0
                weight_0 = 0
                for t in neighbors:  # finding the max weights in different classes
                    if t[0] == 1:
                        weight_1 += t[1]
                    elif t[0] ==2:
                        weight_2 += t[1]
                    else:
                        weight_0 += t[1]

                weight_dict= {weight_1 : 1, weight_2 : 2, weight_0 : 0} #creating dictionary

                max_weight= max(weight_dict.keys()) #finding the key with max weight


                Y_predict[i] = weight_dict[max_weight]  # class with max weight


            return Y_predict
```

```
87      # Function to find the K nearest neighbors to current test example
88
89      def find_neighbors( self, x ) :
90
91          # calculate all the euclidean distances between current
92          # test example x and training set X_train
93          if self.W =="uniform":
94
95              euclidean_distances = np.zeros( self.m )
96
97              for i in range( self.m ) :
98
99                  d = self.euclidean( x, self.X_train[i] )
100
101                 euclidean_distances[i] = d
102
103             # sort Y_train according to euclidean_distance_array and
104             # store into Y_train_sorted
105
106             inds = euclidean_distances.argsort()
107
108             Y_train_sorted = self.Y_train[inds]
109
110
111             return Y_train_sorted[:self.K]
112
113         elif self.W=="distance":
114
115             weights = np.zeros( self.m )
116
117             for i in range( self.m ) :
118
119                 d = self.euclidean( x, self.X_train[i] )
120
121                 weights[i] = d
```

```
            # sort Y_train according to weights and
            # store into Y_train_sorted in reverse order i.e. max weight(closest data point)

            inds = weights.argsort()
            inds= inds[::-1]
            weights= weights[::-1]

            Y_train_reversed = self.Y_train[inds]

            weight_target = np.column_stack((Y_train_reversed,weights))


            return weight_target[:self.K]



    # Function to calculate euclidean distance

    def euclidean( self, x, x_train ) :

        if self.W== "uniform":

            return np.linalg.norm( x - x_train )

        elif self.W== "distance":

            dist= np.linalg.norm( x - x_train ) # calculating the euclidean distanc

            square_distance= dist**2

            return 1/square_distance
```

```
from sklearn.datasets import load_iris
data = load_iris()
X = data.data
y = data.target

 # Splitting dataset into train and test set
from sklearn.preprocessing import StandardScaler


X_train1, X_test1, Y_train, Y_test = train_test_split(
  X, y, test_size = 0.3, random_state = 30 )
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train1)
X_test = scaler.transform(X_test1)
```

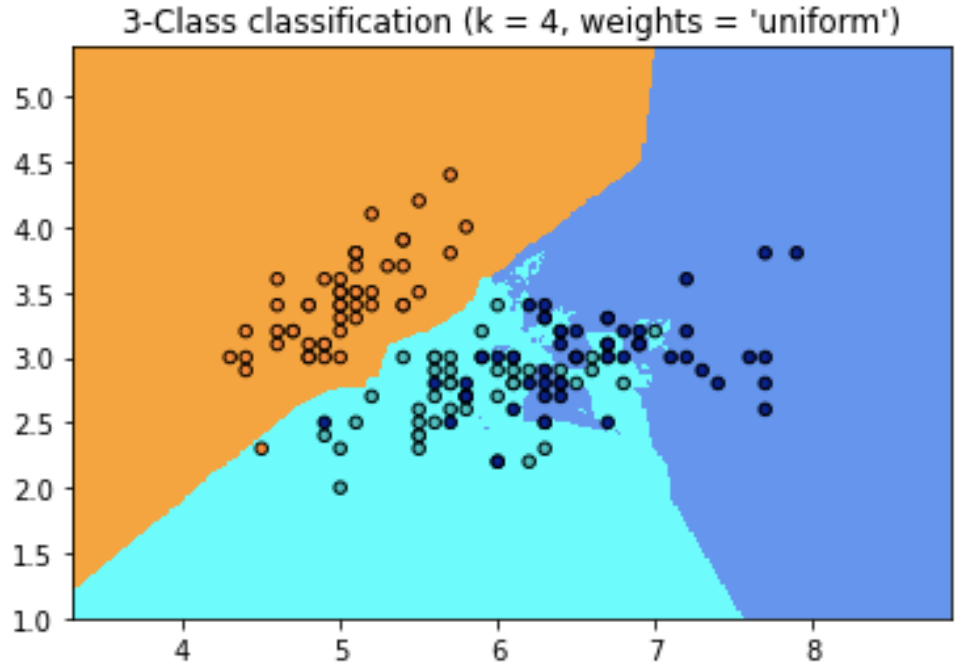
```
K_val= [1,2,3,4,5,6,7,8,9,10]

Weight_val= ["uniform", "distance"]

max_accuracy_val=0

for n_val in K_val:
    for w in Weight_val:

        model1 = GridSearchCV1(n_val, w)

        model1.fit( X_train, Y_train )



        # Prediction on test set

        Y_pred = model1.predict( X_test )


        # measure performance

        correctly_classified = 0


        # counter

        count = 0

        for count in range( np.size( Y_pred ) ) :

            if Y_test[count] == Y_pred[count] :
```

```
                correctly_classified = correctly_classified + 1


    acc= ( correctly_classified / count )* 100.0

    if max_accuracy_val<acc:
        max_accuracy_val= acc
        Best_K= n_val
        Best_W= w


    print( "Accuracy on test set by our model for k = {0} is {1}:  ".format(n_val,( correctly_classified /
    print("the value of K is {0} and Weight is {1}".format(n_val,w))
    print("-----------------------------------------------------")
    print("\n")


print("************************************")
print("The best accuracy is", max_accuracy_val)
print("The best value of K  is", Best_K)
print("The best value of Weight  is", Best_W)
```

```
Accuracy on test set by our model for k = 1 is 90.9090909090909:
the value of K is 1 and Weight is uniform
--------------------------------------------------


Accuracy on test set by our model for k = 1 is 90.9090909090909:
the value of K is 1 and Weight is distance
--------------------------------------------------


Accuracy on test set by our model for k = 2 is 93.18181818181817:
the value of K is 2 and Weight is uniform
--------------------------------------------------


Accuracy on test set by our model for k = 2 is 93.18181818181817:
the value of K is 2 and Weight is distance
--------------------------------------------------


Accuracy on test set by our model for k = 3 is 93.18181818181817:
the value of K is 3 and Weight is uniform
--------------------------------------------------


Accuracy on test set by our model for k = 3 is 95.45454545454545:
the value of K is 3 and Weight is distance
--------------------------------------------------


Accuracy on test set by our model for k = 4 is 97.72727272727273:
the value of K is 4 and Weight is uniform
--------------------------------------------------


Accuracy on test set by our model for k = 4 is 93.18181818181817:
the value of K is 4 and Weight is distance
--------------------------------------------------
```

```
Accuracy on test set by our model for k = 5 is 93.18181818181817:
the value of K is 5 and Weight is uniform
--------------------------------------------------


Accuracy on test set by our model for k = 5 is 90.9090909090909:
the value of K is 5 and Weight is distance
--------------------------------------------------


Accuracy on test set by our model for k = 6 is 93.18181818181817:
the value of K is 6 and Weight is uniform
--------------------------------------------------


Accuracy on test set by our model for k = 6 is 90.9090909090909:
the value of K is 6 and Weight is distance
--------------------------------------------------


Accuracy on test set by our model for k = 7 is 95.45454545454545:
the value of K is 7 and Weight is uniform
--------------------------------------------------


Accuracy on test set by our model for k = 7 is 90.9090909090909:
the value of K is 7 and Weight is distance
--------------------------------------------------


Accuracy on test set by our model for k = 8 is 93.18181818181817:
the value of K is 8 and Weight is uniform
--------------------------------------------------


Accuracy on test set by our model for k = 8 is 93.18181818181817:
the value of K is 8 and Weight is distance
```

```
Accuracy on test set by our model for k = 9 is 95.45454545454545:
the value of K is 9 and Weight is uniform
-----------------------------------------------------


Accuracy on test set by our model for k = 9 is 90.9090909090909:
the value of K is 9 and Weight is distance
-----------------------------------------------------


Accuracy on test set by our model for k = 10 is 97.72727272727273:
the value of K is 10 and Weight is uniform
-----------------------------------------------------


Accuracy on test set by our model for k = 10 is 93.18181818181817:
the value of K is 10 and Weight is distance
-----------------------------------------------------



****************************************
The best accuracy is 97.72727272727273
The best value of K  is 4
The best value of Weight  is uniform
```

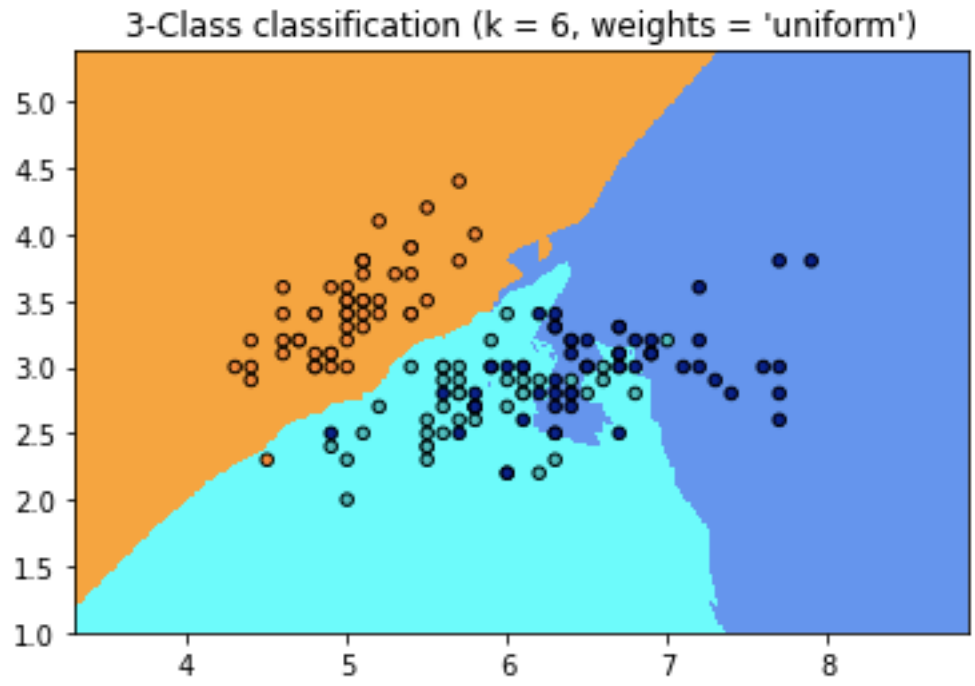


Hyperparameter Tuning for different values of neighbors and weights using Sklearn

Using GridSearchCV

```
from sklearn.model_selection import GridSearchCV

from sklearn.datasets import load_iris
data = load_iris()
X = data.data
y = data.target

X_train1, X_test1, y_train, y_test = train_test_split(X, y, test_size=0.3,stratify=y,random_state= 30)

scaler = StandardScaler()
X_train = scaler.fit_transform(X_train1)
X_test = scaler.transform(X_test1)

metrics = ['euclidean']
neighbors = np.arange(1, 10)
weight=['uniform','distance']
param_grid  = dict(metric=metrics, n_neighbors=neighbors,weights=weight )

knn = KNeighborsClassifier()
knn_cv = GridSearchCV(knn, param_grid)
knn_cv.fit(X_train, y_train)
print("The best parameter is ",knn_cv.best_params_)
print("The best accuracy score is ",knn_cv.best_score_)
```

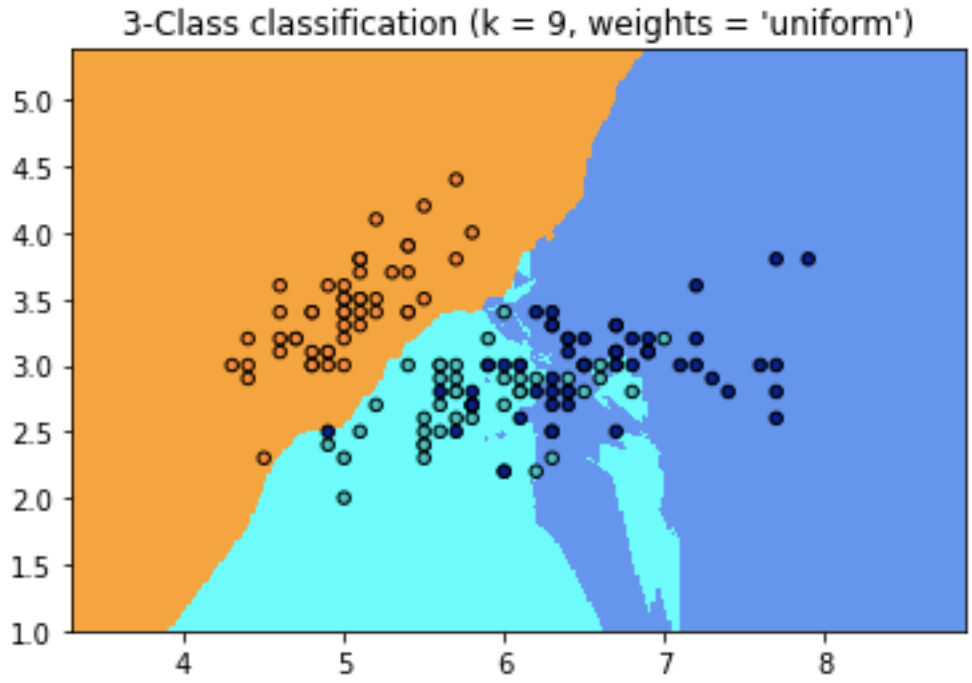
```
The best parameter is  {'metric': 'euclidean', 'n_neighbors': 6, 'weights': 'uniform'}
The best accuracy score is  0.980952380952381
```



Using RandomSearchCV

```
from sklearn.model_selection import RandomizedSearchCV

from sklearn.datasets import load_iris
data = load_iris()
X = data.data
y = data.target

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3,stratify=y,random_state= 30)
metrics = ['euclidean']
neighbors = np.arange(1, 10)
weight=['uniform','distance']
param_grid  = dict(metric=metrics, n_neighbors=neighbors,weights=weight )

knn = KNeighborsClassifier()
knn_cv = RandomizedSearchCV(knn, param_grid)
knn_cv.fit(X_train, y_train)
print("The best parameter is ",knn_cv.best_params_)
print("The best accuracy score is ",knn_cv.best_score_)
```

```
The best parameter is  {'weights': 'uniform', 'n_neighbors': 9, 'metric': 'euclidean'}
The best accuracy score is  0.980952380952381
```



Observation

From the Hyperparameter tuning function created above, we can see that the best value for weights is "Uniform" and this is also similar to randomsearchCV and GridsearchCV which was implemented using Sklearn.

By creating hyperparameter tuning function, we see the best parameters as weight=uniform, K= 4 ,accuracy = 97%

By doing RandomSearchCV, we see the best paramters are weight= uniform, K= 9 and accuracy = 98.9%

By doing GridSearchCV, we see the best paramters are weight= uniform, K= 6 and accuracy = 98.9%.

RESULTS:

MODELS	**User Defined Function (Accuracy)**	**Sklearn (Accuracy)**
KNN Model for K= 4	100%	98%
5- Fold Cross Validation	93%	97%
Weighted KNN	93%	91%
Hyper-Parameter Tuning	K=4, Weight= uniform, accuracy= 97%	RandomsearchCV: weight= uniform, K= 9 and accuracy = 98.9% GridSearchCV: weight= uniform, K= 6 and accuracy = 98.9%.

From the table above, we can see that in hyper-parameter tuning and 5-fold cross validation KNN model using Sklearn is giving higher accuracy although the value of K is large in Sklearn model. We have further cross checked the result using cross validation and cross val score accuracy is close to the hypertuning accuracy. Hence, the model is not overfitted.
However, the user defined function is performing better in 4-NN model and weighted KNN as compared to Sklearn. Also, there is very slight difference in accuracy in both the models which might be due to rounding difference.