raghav_lab-5

February 24, 2025

Instructions Follow the instructions given in comments prefixed with ## and write your code below that.

Also fill the partial code in given blanks.

Don't make any changes to the rest part of the codes

0.0.1 Answer the questions given at the end of this notebook within your report.

0.0.2 You would need to submit your GitHub repository link. Refer to the Section 6: Final Submission on the PDF document for the details.

[4]:
```
import cv2
import numpy as np
import matplotlib.pyplot as plt
from sklearn.cluster import KMeans
from scipy.spatial import distance
```

[ ]:
```
import wandb
from config import API_KEY
wandb.login(key=API_KEY)
wandb.init(project="face_detection_project", name="lab5_experiment")
```

```
Failed to detect the name of this notebook, you can set it manually with the
WANDB_NOTEBOOK_NAME environment variable to enable code saving.
wandb: Using wandb-core as the SDK backend.  Please refer to
https://wandb.me/wandb-core for more information.
wandb: WARNING If you're specifying your api key in code,
ensure this code is not shared publicly.
wandb: WARNING Consider setting the WANDB_API_KEY
environment variable, or running `wandb login` from the command line.
wandb: No netrc file found, creating one.
wandb: Appending key for api.wandb.ai to your netrc file:
C:\Users\Dell\_netrc
wandb: Currently logged in as: raghavsarna
(raghavsarna1) to https://api.wandb.ai. Use `wandb login
--relogin` to force relogin
<IPython.core.display.HTML object>
<IPython.core.display.HTML object>
```

```
<IPython.core.display.HTML object>
```

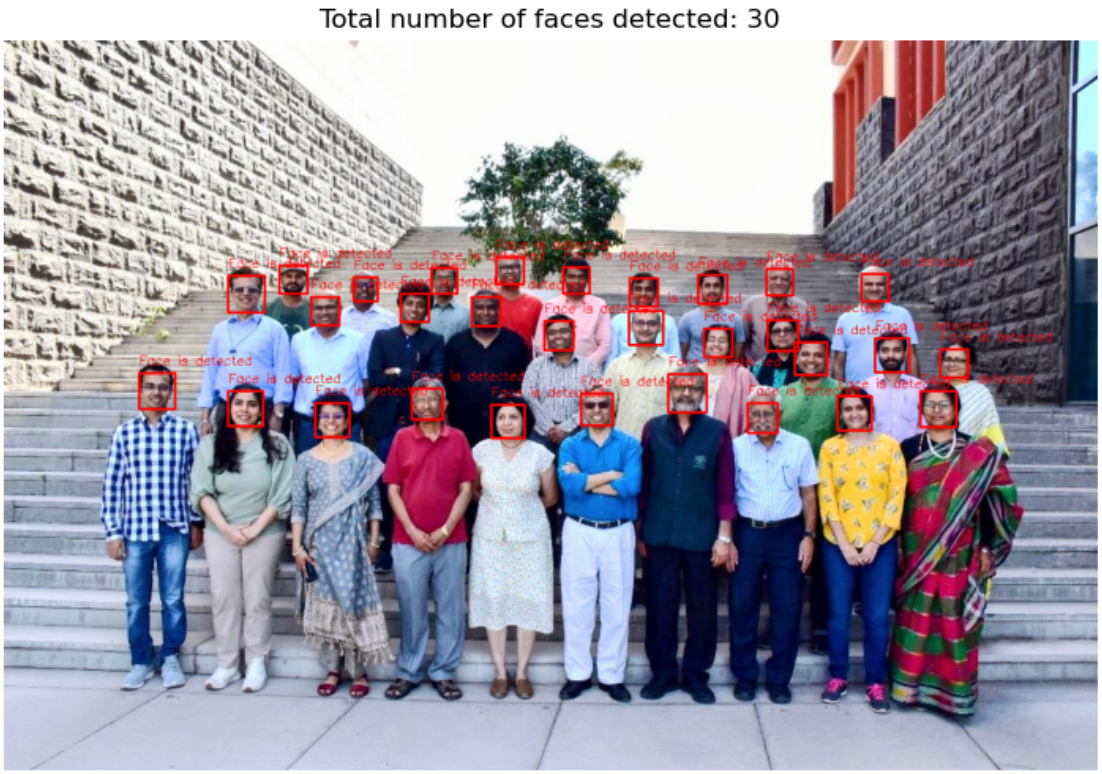
```
<IPython.core.display.HTML object>
```

```
<IPython.core.display.HTML object>
```

[ ]:
```
<wandb.sdk.wandb_run.Run at 0x18133120510>
```

[13]:
```
## Reading the image plaksha_Faculty.jpg
img = cv2.imread("Plaksha_Faculty.jpg")

## Convert the image to grayscale
gray_img = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)

# Loading the required haar-cascade xml classifier file
face_cascade = cv2.CascadeClassifier(cv2.data.haarcascades +␣
 ↪"haarcascade_frontalface_default.xml")

# Applying the face detection method on the grayscale image.
## Change the parameters for better detection of faces in your case.
faces_rect = face_cascade.detectMultiScale(gray_img, 1.05, 4, minSize=(25,25),␣
 ↪maxSize=(50,50))

# Define the text and font parameters
text = "Face is detected" ## The text you want to write
font = cv2.FONT_HERSHEY_SIMPLEX  ## Font type
font_scale = 0.5  ## Font scale factor
font_color = (0,0,255)  ## Text color in BGR format (here, it's red)
font_thickness = 1  ## Thickness of the text


# Iterating through rectangles of detected faces
for (x, y, w, h) in faces_rect:
    cv2.rectangle(img, (x, y), (x+w, y+h), (0, 0, 255), 2)
    # Use cv2.putText to add the text to the image, Use text, font, font_scale,␣
 ↪font_color, font_thickness here
    cv2.putText(img, text, (x, y-10), font, font_scale, font_color,␣
 ↪font_thickness)

## Display the image and window title should be "Total number of face detected␣
 ↪are #"
# Convert BGR to RGB for proper Matplotlib display
img_rgb = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)

# Display the image inline in the notebook
plt.figure(figsize=(10, 6))
plt.imshow(img_rgb)
plt.axis("off")  # Hide axis
```

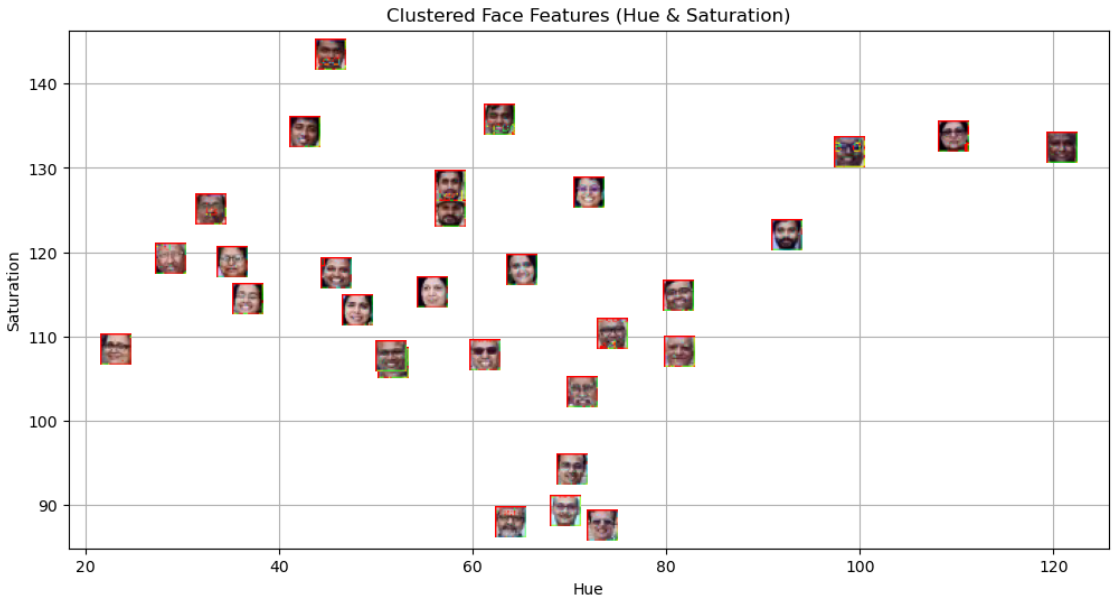
```
plt.title(f"Total number of faces detected: {len(faces_rect)}")
wandb.log({"Face Detection": wandb.Image(plt.gcf())})
plt.show()
```



[7]:
```
from matplotlib.offsetbox import OffsetImage, AnnotationBbox
# Extract face region features (Hue and Saturation)
img_hsv = cv2.cvtColor(img, cv2.COLOR_BGR2HSV)
## call the img and convert it from BGR to HSV and store in img_hsv
hue_saturation = []
face_images = []  # To store detected face images

for (x, y, w, h) in faces_rect:
    face = img_hsv[y:y + h, x:x + w]
    hue = np.mean(face[:, :, 0])
    saturation = np.mean(face[:, :, 1])
    hue_saturation.append((hue, saturation))
    face_images.append(face)

hue_saturation = np.array(hue_saturation)

## Perform k-Means clustering on hue_saturation and store in kmeans
kmeans = KMeans(n_clusters=2, random_state=0).fit(hue_saturation)
```

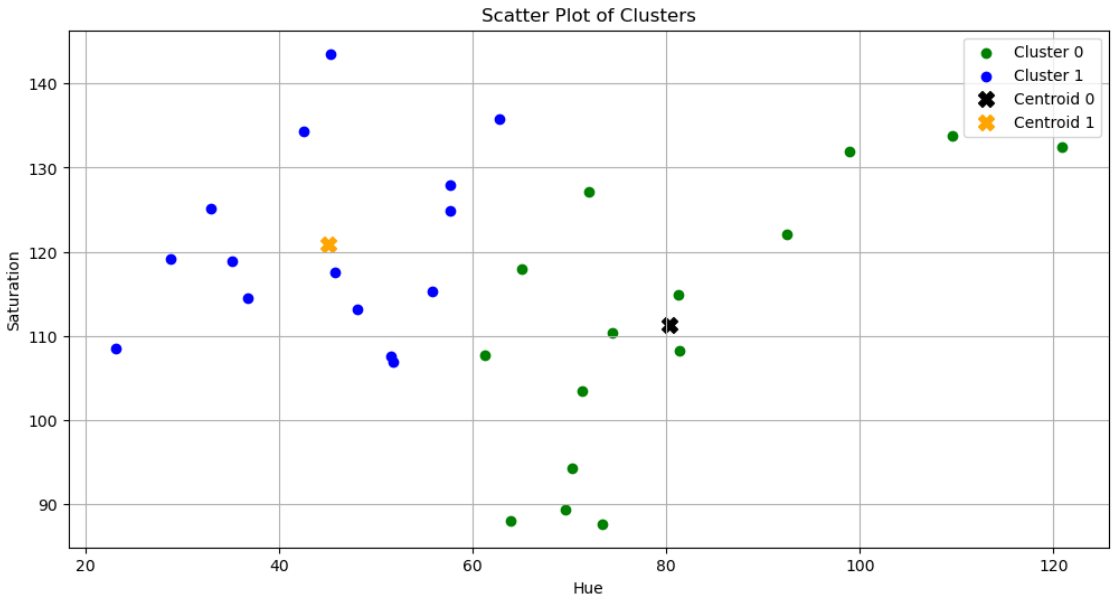
```
centroids = kmeans.cluster_centers_
labels = kmeans.labels_

# Create a figure and axis
fig, ax = plt.subplots(figsize=(12, 6))

# Plot the clustered faces with custom markers
for i, (x,y,w,h ) in enumerate(faces_rect):
    im = OffsetImage(cv2.cvtColor(cv2.resize(face_images[i], (20, 20)), cv2.
 ↪COLOR_HSV2RGB))
    ab = AnnotationBbox(im, (hue_saturation[i, 0], hue_saturation[i, 1]),␣
 ↪frameon=False, pad=0)
    ax.add_artist(ab)
    plt.plot(hue_saturation[i, 0], hue_saturation[i, 1])


## Put x label
plt.xlabel("Hue")
## Put y label
plt.ylabel("Saturation")
## Put title
plt.title("Clustered Face Features (Hue & Saturation)")
## Put grid
plt.grid(True)
## show the plot
wandb.log({"Clustered Face Features": wandb.Image(plt.gcf())})
plt.show()
```

```
c:\Users\Dell\anaconda3\Lib\site-packages\sklearn\cluster\_kmeans.py:870:
FutureWarning: The default value of `n_init` will change from 10 to 'auto' in
1.4. Set the value of `n_init` explicitly to suppress the warning
  warnings.warn(
```



[8]:
```
# Create an empty list to store legend labels
legend_labels = []

# Create lists to store points for each cluster
cluster_0_points = []
cluster_1_points = []

# Your code for scatter plot goes here
fig, ax = plt.subplots(figsize=(12, 6))
for i, (x, y, w, h) in enumerate(faces_rect):
    if kmeans.labels_[i] == 0:
        cluster_0_points.append((hue_saturation[i, 0], hue_saturation[i, 1]))
    else:
        cluster_1_points.append((hue_saturation[i, 0], hue_saturation[i, 1]))


cluster_0_points = np.array(cluster_0_points)
# Plot points for cluster 0 in green
plt.scatter(cluster_0_points[:, 0], cluster_0_points[:, 1], color='green',␣
 ↪label='Cluster 0')


cluster_1_points = np.array(cluster_1_points)
# Plot points for cluster 1 in blue
plt.scatter(cluster_1_points[:, 0], cluster_1_points[:, 1], color='blue',␣
 ↪label='Cluster 1')
```

```
# Calculate and plot centroids
centroid_0 = kmeans.cluster_centers_[0]
centroid_1 = kmeans.cluster_centers_[1]

# Plot both the centroid for cluster 0 and cluster 1
plt.scatter(centroid_0[0], centroid_0[1], color='black', marker='X', s=100,␣
 ↪label='Centroid 0')
plt.scatter(centroid_1[0], centroid_1[1], color='orange', marker='X', s=100,␣
 ↪label='Centroid 1')

## Put x label
plt.xlabel("Hue")
## Put y label
plt.ylabel("Saturation")
## Put title
plt.title("Scatter Plot of Clusters")
## Add a legend
plt.legend()
## Add grid
plt.grid(True)
## Show the plot
wandb.log({"Scatter Plot of Clusters": wandb.Image(plt.gcf())})
plt.show()
```

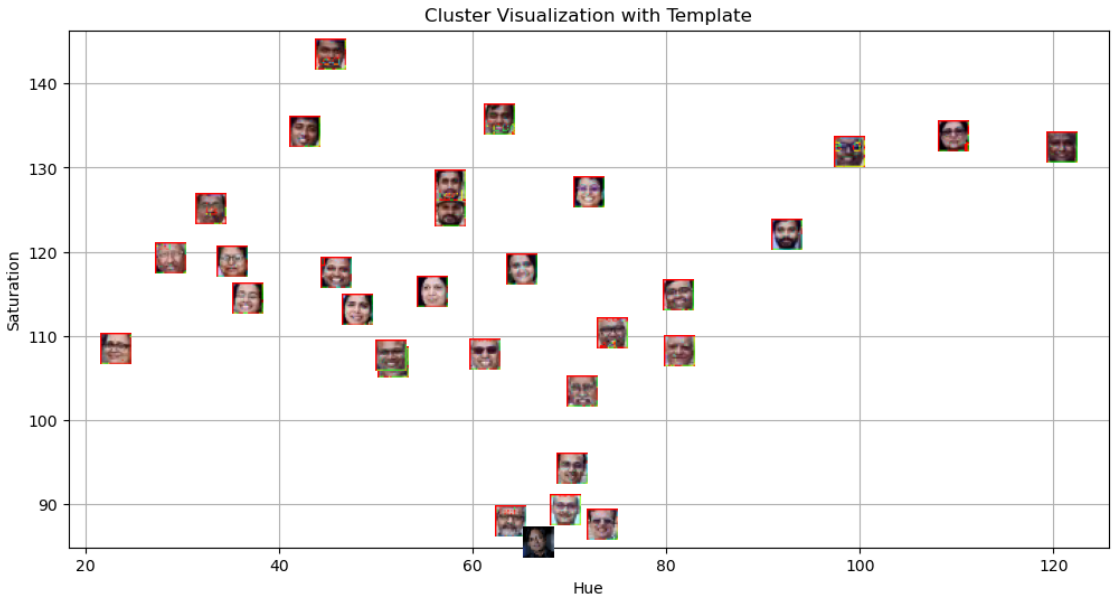


[9]:
```
## Read the class of the template image 'Dr_Shashi_Tharoor.jpg' using cv2 and␣
 ↪store it in template_img
template_img = cv2.imread("Dr_Shashi_Tharoor.jpg")
```

```
# Detect face  in the template image after converting it to gray and store it␣
 ↪in template_faces
template_gray = cv2.cvtColor(template_img, cv2.COLOR_BGR2GRAY)
template_faces = face_cascade.detectMultiScale(template_gray, 1.05, 4,␣
 ↪minSize=(25,25), maxSize=(50,50))

# Draw rectangles around the detected faces
for (x, y, w, h) in template_faces:
    cv2.rectangle(template_img, (x, y), (x + w, y + h), (0, 255, 0), 3)

template_rgb = cv2.cvtColor(template_img, cv2.COLOR_BGR2RGB)
plt.figure(figsize=(6, 6))
plt.imshow(template_rgb)
plt.axis("off")  # Hide axis
plt.title("Detected Faces in Template Image")
wandb.log({"Template Face Detection": wandb.Image(plt.gcf())})
plt.show()
```

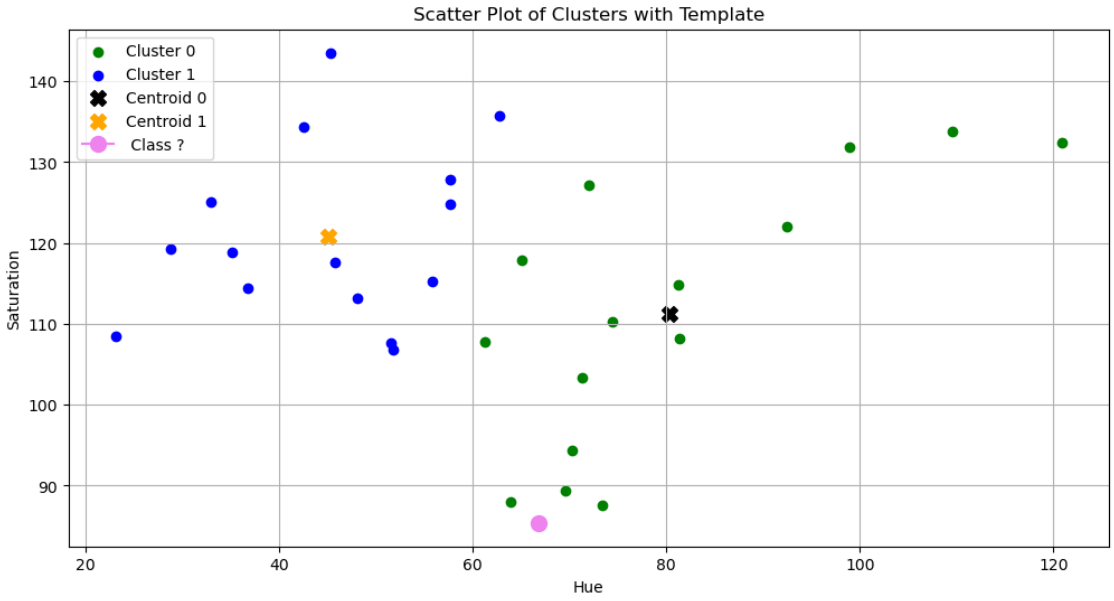
[10]:
```
# Convert the template image to HSV color space and store it in template_hsv
template_hsv = cv2.cvtColor(template_img, cv2.COLOR_BGR2HSV)

# Extract hue and saturation features from the template image as we did it for␣
 ↪detected faces.
template_hue = np.mean(template_hsv[:, :, 0])
template_saturation = np.mean(template_hsv[:, :, 1])

# Predict the cluster label for the template image and store it in␣
 ↪template_label
template_label = kmeans.predict([[template_hue, template_saturation]])[0]

# Create a figure and axis for visualization
fig, ax = plt.subplots(figsize=(12, 6))

# Plot the clustered faces with custom markers (similar to previous code)
for i, (x, y, w, h) in enumerate(faces_rect):
    color = 'red' if kmeans.labels_[i] == 0 else 'blue'
    im = OffsetImage(cv2.cvtColor(cv2.resize(face_images[i], (20, 20)), cv2.
 ↪COLOR_HSV2RGB))
    ab = AnnotationBbox(im, (hue_saturation[i, 0], hue_saturation[i, 1]),␣
 ↪frameon=False, pad=0)
    ax.add_artist(ab)
    plt.plot(hue_saturation[i, 0], hue_saturation[i, 1], 'o', markersize=5,␣
 ↪color=color)

# Plot the template image in the respective cluster
if template_label == 0:
    color = 'red'
else:
    color = 'blue'
im = OffsetImage(cv2.cvtColor(cv2.resize(template_img, (20, 20)), cv2.
 ↪COLOR_BGR2RGB))
ab = AnnotationBbox(im, (template_hue, template_saturation), frameon=False,␣
 ↪pad=0)
ax.add_artist(ab)

## Put x label
plt.xlabel("Hue")
## Put y label
plt.ylabel("Saturation")
## Put title
plt.title("Cluster Visualization with Template")
## Add grid
plt.grid(True)
## show plot
```

```
wandb.log({"Cluster Visualization with Template": wandb.Image(plt.gcf())})
plt.show()
```



[11]:
```
# Create an empty list to store legend labels
legend_labels = []

# Create lists to store points for each cluster
cluster_0_points = []
cluster_1_points = []

# Your code for scatter plot goes here
fig, ax = plt.subplots(figsize=(12, 6))
for i, (x, y, w, h) in enumerate(faces_rect):
    if kmeans.labels_[i] == 0:
        cluster_0_points.append((hue_saturation[i, 0], hue_saturation[i, 1]))
    else:
        cluster_1_points.append((hue_saturation[i, 0], hue_saturation[i, 1]))

# Plot points for cluster 0 in green
cluster_0_points = np.array(cluster_0_points)
plt.scatter(cluster_0_points[:, 0], cluster_0_points[:, 1], color='green',␣
 ↪label='Cluster 0')

# Plot points for cluster 1 in blue
cluster_1_points = np.array(cluster_1_points)
plt.scatter(cluster_1_points[:, 0], cluster_1_points[:, 1], color='blue',␣
 ↪label='Cluster 1')
```

```
# Calculate and plot centroids for both the clusters
centroid_0 = kmeans.cluster_centers_[0]
centroid_1 = kmeans.cluster_centers_[1]
plt.scatter(centroid_0[0], centroid_0[1], color='black', marker='X', s=100,␣
 ↪label='Centroid 0')  ## plot for centroid 0
plt.scatter(centroid_1[0], centroid_1[1], color='orange', marker='X', s=100,␣
 ↪label='Centroid 1')  ## plot for centroid 1
## plot for centroid 1
plt.plot(template_hue, template_saturation, marker='o', c= 'violet',markersize=␣
 ↪10, label=' Class ?' )

## Put x label
plt.xlabel("Hue")
## Put y label
plt.ylabel("Saturation")
## Put title
plt.title("Scatter Plot of Clusters with Template")
## Add a legend
plt.legend()
## Add grid
plt.grid(True)
## show the plot
wandb.log({"Final Scatter Plot with Template": wandb.Image(plt.gcf())})
plt.show()
                                                    ## End of the lab 5 ##
```



0.1 Report:

0.2 Answer the following questions within your report:

1. What are the common distance metrics used in distance-based classification algorithms? Some common metrics are:

- Eucledian Distance, which is the straight line distance between two points in an eucledian space.
- Mahattan distance (city block distance), which is distance between two points measured along axis at right angles.
- Chebyshev distance, where the distance between two vectors is the greatest of their differences along any coordinate dimension.
- Cosine distance, which measures angle between two vectors.

2. What are some real-world applications of distance-based classification algorithms? Distance based algorithms like KNN are used in many applications like: recommender system for suggesting things with similar characteristics, text classification to detect similar messages like spam, medical diagnosis to classify patient data.

3. Explain various distance metrics.

- Eucledian distance: It is the most common general purpose distance in low dimensions spaces. It is very sensitive to scale of data. It's formula:

$$d(x, y) = \sqrt{\sum_{i=1}^{n} (x_i - y_i)^2}$$

- Mahalanobis distance: It accounts for correlation between features and then calculates the distance. It is good for dataset with correlated features. It's formula:

$$d(x, y) = \sqrt{(x - y)^T S^{-1} (x - y)}$$

- Cosine distance: It measures orientation rather than magnitude. Used in high dimensional space where direction of vector matters more than the magnitude. It's formula:

$$d(x, y) = 1 - \frac{x \cdot y}{\|x\| \|y\|}$$

4. What is the role of cross validation in model performance? Cross validation is used to analyze the generalizability of a model. In some models, it is very helpful in tuning the hyperparameters like 'k' in KNN. It also prevents overfitting. When we partition data into several folds and train the model on the remaining subset and every data point gets to be in the test set once, so it gives reliable estimate of how good the model is performing.

5. Explain variance and bias in terms of KNN? Bias is the inability of model to perform because of some differnce/error occuring between model's predicted value and actual value. For low bias, the model will match the training dataset and for high bias, model will not match training dataset. For KNN, if we have large k, there is a smooth decision boundary and it leads to higher bias. Meaning the model is over simplyfying.

Variance is the amount by which performance of predictive model changes when trained on different subsets of data. It measures how sensitive it is to another susbet of training dataset. For KNN, small k means that model is influenced a lot by the noise and individual data points, which leads to high variance where model is good on training data but bad on unseen data.