Model Evalution

[1]:
```
#Importing Liberies:
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import os
import cv2
import time as t
```

Statement

The dataset comprising of dogs and cats images is uploaded to google drive. You will have to write a code that can use **KNNClassifier, SVC, and logistic regression** for classification. You can train the data using the train.zip folder and report the confusion matrix on test1.zip folder.

Answer:

1.1 Feature Extraction:

1. Read each image from the train image folder and convert into grayscale image.
2. Preprocess the images by resizing to (50 x 50) pixel value.
3. Calculating Gradients (direction x and y).
    1. Change in X direction: $\Delta X_{i,j} = a_{i,j-1} - a_{i,j+1}$
    2. Change in Y direction: $\Delta Y_{i,j} = a_{i-1,j} - a_{i+1,j}$
    3. Calculate the gradient magnitude: $Total\ Gradient\ Magnitude = \sqrt{(\Delta X^2 - \Delta Y^2)}$
    4. Calculate the orientation (or direction): $\Phi = \arctan(\frac{\Delta Y}{\Delta X})$
4. Generate histogram having binsize of 16 and use the gradient magnitude to fill the values in the matrix(Histogram of Oriented Gradients (HOG)).
5. Assign the label for the images, where '0' represents 'Cat' and '1' represents 'Dog'.
6. Stack all the features and labels and store into the data variables.

[2]:
```
'''
Feature Extraction:
```

```
'''
def hog(img):
    gx = cv2.Sobel(img, cv2.CV_32F, 1, 0) #gradient in x
    gy = cv2.Sobel(img, cv2.CV_32F, 0, 1)  #gradient in x
    mag, ang = cv2.cartToPolar(gx, gy)      #magnitude and angle
    bin_n = 16                             # Number of bins
    bin = np.int32(bin_n*ang/(2*np.pi))
    bin_cells = []
    mag_cells = []

    # no of hog feature cells
    cellx = celly = 8

    for i in range(0,img.shape[0]//celly):
        for j in range(0,img.shape[1]//cellx):
            bin_cells.append(bin[i*celly : i*celly+celly, j*cellx :␣
↪j*cellx+cellx])
            mag_cells.append(mag[i*celly : i*celly+celly, j*cellx :␣
↪j*cellx+cellx])

    # appending histogram
    hists = [np.bincount(b.ravel(), m.ravel(), bin_n) for b, m in␣
↪zip(bin_cells, mag_cells)]
    hist = np.hstack(hists)

    return hist

def feature_extractor(path):
    feature = []
    label = []
    for file in os.listdir(path):
        # print(file)
        #importing the image and resize it to (50*50)
        img = cv2.resize(cv2.imread(os.path.join(path, file)), (50, 50))
        feature.append(hog(img))
        if (file[0:3]=="cat")==True:            # cat indicated by 0
            label.append(0)
        elif (file[0:3]=="dog")==True:             # dog indicated by 1
            label.append(1)

    labels = np.array([label], dtype = int)        #label array
    features = np.array(feature, dtype = float) #feature array
    data = np.hstack((features,labels.T))

    return data
```

[3]:
```
%%time
# import the Resnet data
image_folder = r"train/"
data = feature_extractor(image_folder)

X,y = data[:,:-1], data[:,-1:].flatten().astype('int64')
print('Dimention of Features into the data: ',X.shape)
print('Dimention of label into the data: ',y.shape)
```

```
Dimention of Features into the data:  (25000, 576)
Dimention of label into the data:  (25000,)
CPU times: user 34.4 s, sys: 921 ms, total: 35.3 s
Wall time: 41 s
```

1.2 Scaling the dataset and split into test and train sets

[4]:
```
'''
Test Train split:
'''
# test train split for multi dimentional datasets
def train_test_split(X,y,k):
    x_test = []
    x_train = []
    y_test = []
    y_train = []
    A = np.column_stack((X,y)) #stacking the feature and lables into single␣
 ↪array A
    np.random.shuffle(A) #suffling data into random manner
    n = np.round(len(X)*k)

    for i in range (len(X)):
        #test data
        if i<n:
            x_test.append(A[:,:-1][i])
            y_test.append(A[:,-1:][i])

        #train data
        else:
            x_train.append(A[:,:-1][i])
            y_train.append(A[:,-1:][i])

    return np.array(x_train), np.array(x_test), np.ravel(y_train).
 ↪astype('int64'), np.ravel(y_test).astype('int64')

'''
Standard Scaler:
'''
```

```
def Standard_Scaler(X):
    A = []
    B = X.T
    for row in B:
        A.append((row-np.mean(row))/np.std(row))
    x = np.array(A).T
    return x
```

[5]:
```
# Scale the featureset
X = Standard_Scaler(X)
# 30% test train split
Xtrain, Xtest, ytrain, ytest = train_test_split(X, y, 0.3)
print('Dimention of train data: ',Xtrain.shape)
print('Dimention of test data: ',Xtest.shape)
```

```
Dimention of train data:  (17500, 576)
Dimention of test data:  (7500, 576)
```

[6]:
```
'''
Confusion Matrix:
'''
def confusion_matrix(Y,y,a):
    pred = y    #predicted values
    true = Y     #actual values
    classes = len(np.unique(ytest))
    #computing the confusion matrix
    conf_matrix = np.bincount(true * classes + pred).reshape((classes, classes))

    # Print the confusion matrix using Matplotlib
    fig, ax = plt.subplots(figsize=(8.5, 8.5))
    ax.matshow(conf_matrix, cmap=plt.cm.Blues, alpha=0.5)
    ticks = ['Cat','Dog']
    for i in range(conf_matrix.shape[0]):
        for j in range(conf_matrix.shape[1]):
            ax.text(x=j, y=i,s=conf_matrix[i, j], va='center', ha='center',␣
 ↪size='xx-large')
    ax.set_xticks(np.arange(len(ticks)), labels=ticks, fontsize=12)
    ax.set_yticks(np.arange(len(ticks)), labels=ticks, fontsize=12)
    plt.xlabel('Predictions', fontsize=16)
    plt.ylabel('Actuals', fontsize=16)
    plt.title('Confusion Matrix '+ a, fontsize=22)
    plt.show()
    return np.ravel(conf_matrix)

# computing the accuracy
def accuracy(y_true, y_pred):
    accuracy = np.sum(y_true == y_pred)/len(y_true)*100
```

```python
    return np.around(accuracy, 3)
```

1.3 Support Vector Machine:

1.3.1 Algorithm:

1. Equation for the linear hyperplane:

$$y = w^T X - b$$

>where, $w^T.x_i - b \geq 1 \qquad if,\ y_i = 1$ »$w^T.x_i - b \leq 1 \qquad if,\ y_i = -1$

2. Hinge Loss:

$$l = max(0, 1 - y_i(w.x_i - b))$$

> where,

$$l = \begin{cases} 0, & \text{if } y.f(x) \geq 1 \\ 1 - y.f(x), & \text{otherwise} \end{cases}$$

3. After adding the Regulariser:

$$L = \lambda||w||^2 + \frac{1}{n}\sum_{i=0}^{n} max(0, 1 - y_i(w.x_i - b))$$

> where, $if \quad y_i.f(x) \geq 1 \quad then,\ L = \lambda||w||^2$ »$else, \quad L = \lambda||w||^2 + 1 - y_i(w.x_i - b)$

4. Gradient: >where, »

$$if, \quad y_i.f(x) \geq 1 \quad then, \frac{\partial L}{\partial w} = 2\lambda w \quad and \quad \frac{\partial L}{\partial b} = 0$$

»

$$else, \quad \frac{\partial L}{\partial w} = 2\lambda w - y_i.x_i \quad and \quad \frac{\partial L}{\partial b} = y_i$$

5. Update rules:

$$w_{i+1} = w_i - \alpha.\partial w \quad and \quad b_{i+1} = b_i - \alpha.\partial b$$

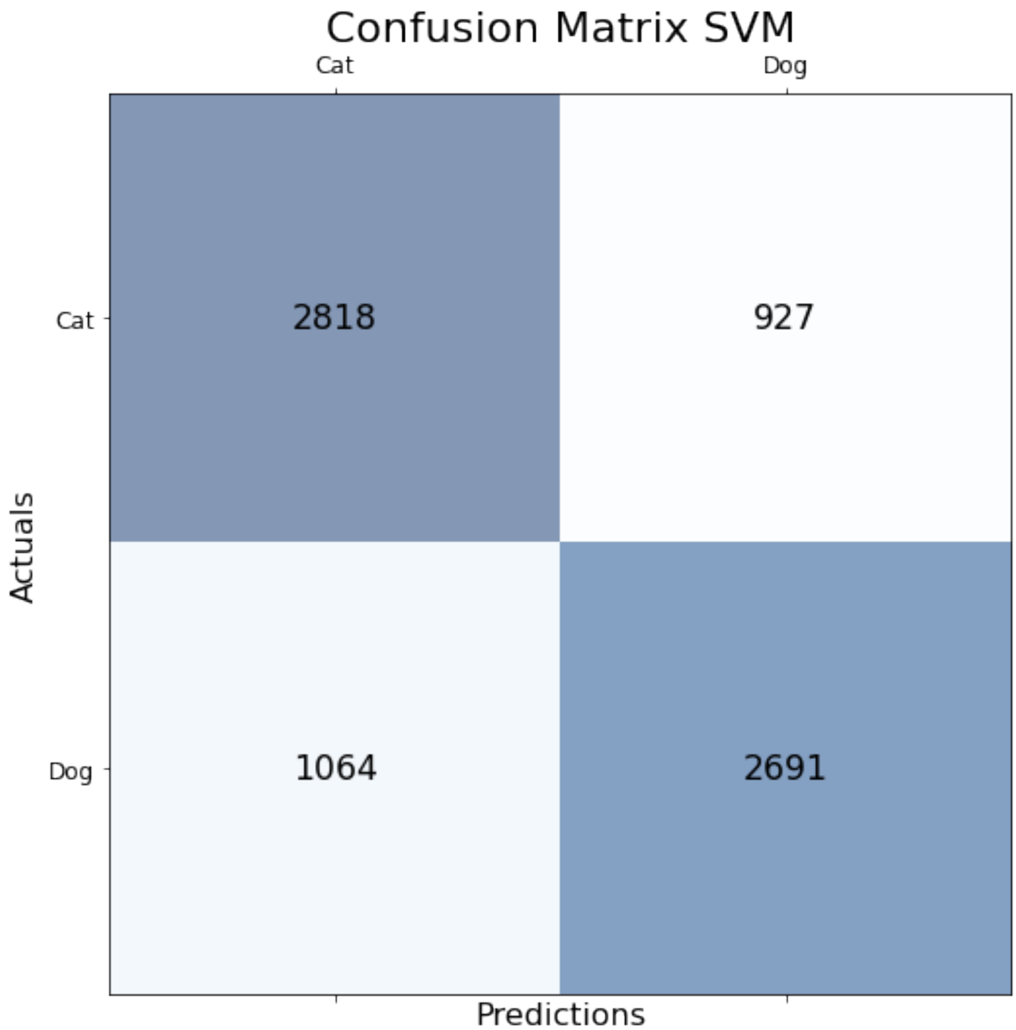
>where, $\alpha \rightarrow learning\ rate$ » $\lambda \ \rightarrow \ regulariser$

[7]:
```python
'''
Support Vector Machine Classifier(Linear):
'''
class SVM:
    #initilize the parameters
    def __init__(self, learning_rate=1.8e-6, lambda_=0.001, n_iters=125):
        self.lr = learning_rate
        self.lambda_ = lambda_
        self.n_iters = n_iters
        self.w = None
        self.b = None
```

```
    # fit method
    def fit(self, X, y):
        n_samples, n_features = X.shape # assign the no of shamples and features

        y_ = np.where(y <= 0, -1, 1)       # converting y=0,1 to y=-1,1

        self.w = np.zeros(n_features)        # initilize the weights
        self.b = 0                             # initilize the bias

        for _ in range(self.n_iters):
            for i, x_i in enumerate(X):
                condition = y_[i] * (np.dot(x_i, self.w) - self.b) >= 1
                # update the weights and bias
                if condition:
                    self.w -= self.lr * (2 * self.lambda_ * self.w)
                else:
                    self.w -= self.lr * (2 * self.lambda_ * self.w - np.
↪dot(x_i, y_[i]))
                    self.b -= self.lr * y_[i]

    # predict method
    def predict(self, X):
        approx = np.dot(X, self.w) - self.b
        return np.where(np.sign(approx) <= 0, 0, 1)
```

[8]:
```
tic = t.process_time()
svm = SVM()
svm.fit(Xtrain, ytrain)
y_pred1 = svm.predict(Xtest)
toc = t.process_time()
a1 = accuracy(ytest, y_pred1)
t1 = np.round(toc-tic,2)
print('SVM model Accuracy on the test_set : ',a1,'%')
print('Processing time taken for the SVM model: ',t1,' sec')
conf1 = confusion_matrix(ytest, y_pred1, 'SVM')
```

```
SVM model Accuracy on the test_set :  73.453 %
Processing time taken for the SVM model:  27.76  sec
```



1.4 K_NearestNeighbour:

1.4.1 Algorithm:

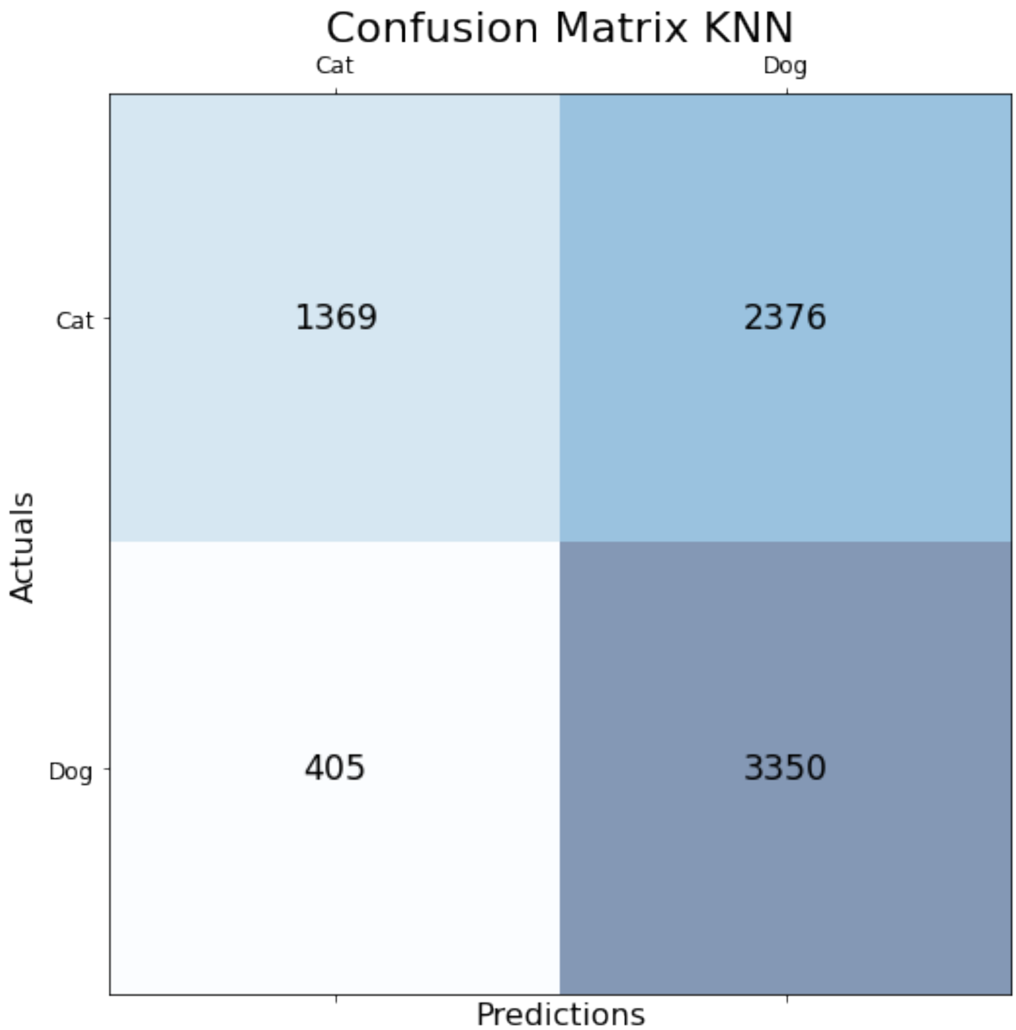
1. Select the number K of the neighbors,
2. Calculate the Euclidean distance of K number of neighbors. > where, $D^2_{a,b} = \sum_{k=0}^{n}(a_k - b_k)^2$
3. Take the K nearest neighbors as per the calculated Euclidean distance.
4. Among these k neighbors, count the number of the data points in each category.
5. Assign the new data points to that category for which the number of the neighbor is maximum.

[9]:
```
'''
KNN Classifier:
'''
```

```python
class knn:
    #initilize the parameters
    def __init__(self, n_neighbours):
        self.k = n_neighbours

    # fit method
    def fit(self, X, y):
        self.X = X
        self.y = y
        return self

    #square euclidean norm
    def eluclidean_dist(self,a,b):
        return np.dot((a-b),(a-b))

    # predict method
    def predict(self, X, y):
        y_pred = []
        for row in X:
            # compute euclidean distance
            dist = []
            for X_row in self.X:
                dist.append(self.eluclidean_dist(X_row,row))

            # get k nearest samples, lables
            neighbours = list(self.y[np.argsort(dist)[:self.k]])
            vote = np.bincount(neighbours)                        # voting the␣
 ↪closest neighbours
            pred = np.argmax(vote)                                 #assign to␣
 ↪the mojority
            y_pred.append(pred)
        return np.array(y_pred)
```

[10]:
```python
tic = t.process_time()
knn = knn(5)
knn.fit(Xtrain,ytrain)
y_pred2 = knn.predict(Xtest, ytest)
toc = t.process_time()
a2 = accuracy(ytest, y_pred2)
t2 = np.round(toc-tic,2)
print('KNN model Accuracy on the test_set : ',a2,'%')
print('Processing time taken for the KNN model: ',t2,' sec')
conf2 = confusion_matrix(ytest, y_pred2, 'KNN')
```

```
KNN model Accuracy on the test_set :  62.92 %
Processing time taken for the KNN model:  407.96  sec
```



1.5 Logistic Regression:

1.5.1 Algorithm:

1. Linear model:
$$y = w^T x + b$$
2. Approximation:
$$\hat{y} = \frac{1}{1 + e^{-(w^T x + b)}}$$
3. Cross entropy:
$$L = \frac{1}{N} \sum_{i=1}^{n} [y_i \log(\hat{y}_i) + (1 - y_i) \log(1 - \hat{y}_i)]$$

4. Gradient Descent:

$$\frac{\partial L}{\partial w} = \frac{1}{N}\sum_{i=0}^{n} 2x_i^T(\hat{y} - y_i)$$

>and,

$$\frac{\partial L}{\partial b} = \frac{1}{N}\sum_{i=0}^{n} 2(\hat{y} - y_i)$$

5. Update rules:

$$w_{i+1} = w_i - \alpha.\partial w \quad and \quad b_{i+1} = b_i - \alpha.\partial b$$

>where, $\alpha \rightarrow$ *learning rate*

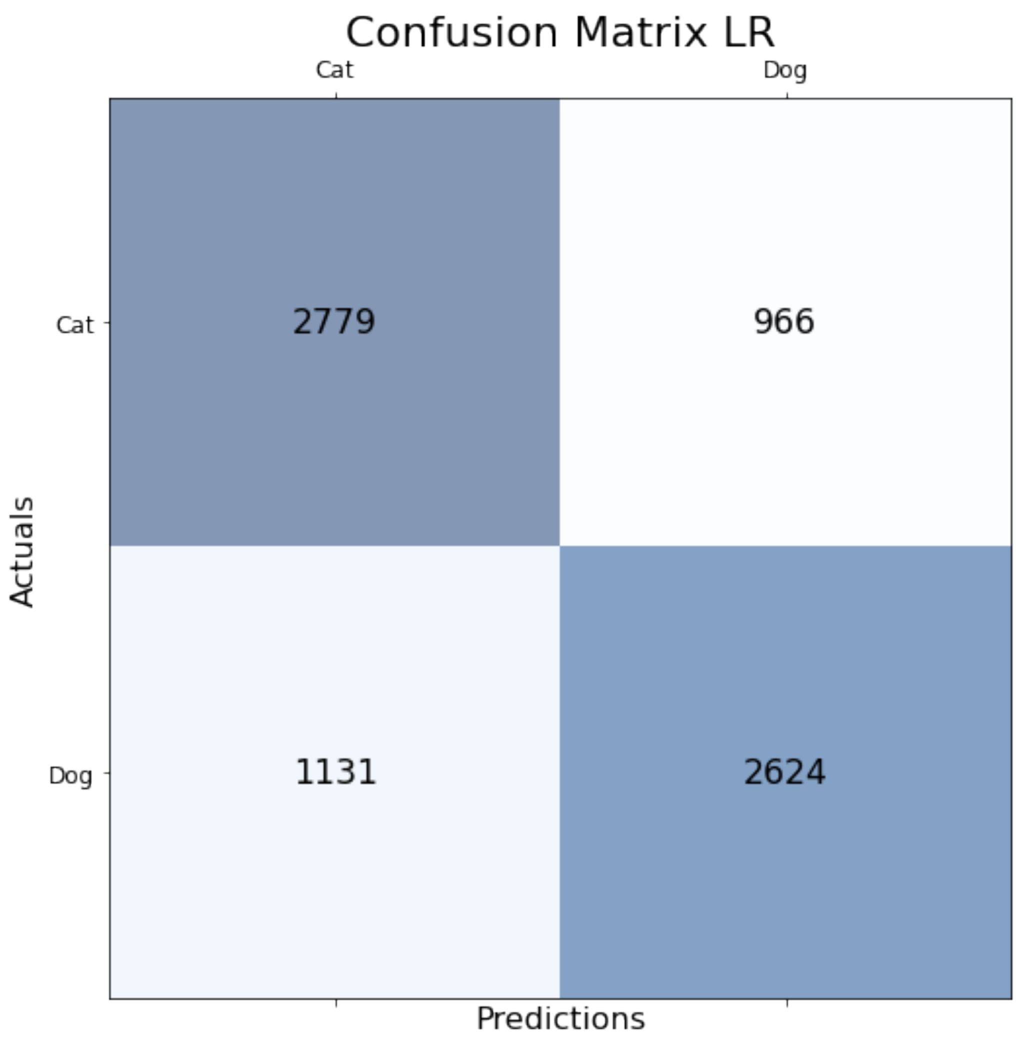
[11]:
```
'''
Logistics Regression:
'''
class LogisticRegression:
    # initilise the parameters
    def __init__(self, learning_rate, n_iter=125):
        self.n_iter = n_iter
        self.lr = learning_rate
        self.w = None
        self.b = None

    # fit methode
    def fit(self,X,y):
        n_samples, n_features = X.shape
        self.w = np.zeros(n_features)  #initilise the weights
        self.b = 0                    #initilise the bias

        # gradient descent
        for _ in range(self.n_iter):
            y_pred = self.sigmoid(X)

            #calculate gradient of weights and bias
            dw = (1/n_samples) * np.dot(X.T, (y_pred-y))
            db = (1/n_samples) * np.sum(y_pred-y)
            #update weights
            self.w -= self.lr*dw
            self.b -= self.lr*db
        return

    # sigmoid function
    def sigmoid(self, x):
        z = np.dot(x,self.w)+self.b
        return (1/(1+np.exp(-z)))

    # predict methode
```

```python
    def predict(self, x):
        y = self.sigmoid(x)
        return np.where(y <= 0.5, 0, 1)
```

[12]:

```python
tic = t.process_time()
LR = LogisticRegression(learning_rate=0.01)
LR.fit(Xtrain,ytrain)
y_pred3 = LR.predict(Xtest)
toc = t.process_time()

a3 = accuracy(ytest, y_pred3)
t3 = np.round(toc-tic,2)
print('Logistic Regression model Accuracy on the test_set : ',a3,'%')
print('Processing time taken for the LR model: ',t3,' sec')
conf3 = confusion_matrix(ytest, y_pred3, 'LR')
```

```
Logistic Regression model Accuracy on the test_set :  72.04 %
Processing time taken for the LR model:  5.02  sec
```



[13]:
```
"""
Side by side comparison of all three machinelearning models:
"""
# initialize list of lists
data2 = [['Support Vector Machine', t1, a1, conf1[0], conf1[1], conf1[2],␣
 ↪conf1[3]], ['K-Nearest Neighbour', t2, a2, conf2[0], conf2[1], conf2[2],␣
 ↪conf3[3]], ['Logistics Regression', t3, a3, conf3[0], conf3[1], conf3[2],␣
 ↪conf3[3]]]

# Create the pandas DataFram3
df = pd.DataFrame(data2, columns = ['Classification Model', 'Procesing␣
 ↪time(Sec)', 'Accuracy{%}','True Cat', 'False Cat', 'False Dog', 'True Dog',])
```

```python
# print dataframe.
display(df)
```

```
    Classification Model  Procesing time(Sec)  Accuracy{%}  True Cat  \
0  Support Vector Machine                27.76       73.453      2818
1     K-Nearest Neighbour               407.96       62.920      1369
2    Logistics Regression                 5.02       72.040      2779

   False Cat  False Dog  True Dog
0        927       1064      2691
1       2376        405      2624
2        966       1131      2624
```

1.6 Conclusion:

1. Traning the model after performing the feature extraction give far better result, compare to the using the raw pixel values as the feature vector.
2. Also tested using PCA. But PCA delivered poor result as we loose too much infromation from the data sets after peforming PCA.
3. Also tested using Keras resnet 50 feature extractor which delivered promissing results, but as it uses neuralnetwork for feature extraction so it is not use in the final answer {All the testing files are to be attached into the 'Experiment' folder}.
4. From the model output we can conclude that,
    1. Time complexity is highest for the KNN model compare to svm and logistic regression.
    2. For this image classification svm delivered the best accuracy followed by Logistics Regression and KNN.
    3. For similar no of itteration and learning rate SVM is superior than logistics regression.
