```
# Import all dependencies
import pandas as pd
import numpy as np
from sklearn.svm import SVC
from sklearn.model_selection import train_test_split
from sklearn import metrics
from sklearn.metrics import confusion_matrix
from sklearn.model_selection import KFold
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import GridSearchCV
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.preprocessing import scale
```

```
# mount drive for easy import and export of data
from google.colab import drive
drive.mount('/content/drive')
```

```
    Mounted at /content/drive
```

```
# initialise dataframe with letter dataset
pascal = pd.read_csv("/content/drive/MyDrive/DM/Image-pixels.csv")
```

```
pascal.shape
```

```
    (4382, 22501)
```

```
col=[i for i in range(22500)]
col.append('label')
pascal.columns=col
pascal.columns
```

```
    Index([     0,      1,      2,      3,      4,      5,      6,      7,
                8,      9,
           ...
             22491,  22492,  22493,  22494,  22495,  22496,  22497,  22498,
             22499, 'label'],
          dtype='object', length=22501)
```

```
# initialise parameters
markov= pd.DataFrame(columns = pascal.columns)
uniqCls=list(np.sort(pascal['label'].unique()))
classCNT=len(uniqCls)
limit=250
m=classCNT*limit
```

```
# Chose parameters for markov sampling
k=5
```

```
q=1.2
rej=0
```

```
# Train a linear Model on N[here 2000] size train set
X = pascal.drop("label", axis = 1)
y = pascal['label']

# train test split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.8, random_state = 101)
model_linear = SVC(kernel='linear')
model_linear.fit(X_train, y_train)
```

```
SVC(C=1.0, break_ties=False, cache_size=200, class_weight=None, coef0=0.0,
    decision_function_shape='ovr', degree=3, gamma='scale', kernel='linear',
    max_iter=-1, probability=False, random_state=None, shrinking=True,
    tol=0.001, verbose=False)
```

```
predProb=[]
```

```
# Utility loss Function
def lossF(actual,pred):
    if actual==pred:
        return 1.0
    return np.exp(-2)
```

```
# Utility function for training subsequent models
def train(data):

    X = data.drop("label", axis = 1)
    y = data['label']

    # train test split
    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.01, random_state = 101)
    model_linear = SVC(kernel='linear')
    model_linear.fit(X_train, y_train)

    return model_linear
```

```
lst=[]
```

```
t=0
T=3
# Loop markov chain generator T times
while t<T:
    # Reset parameters for next markov chain
    markov= pd.DataFrame(columns = pascal.columns)
    predProb=[]
    lst=[]
```

```
lst=[]

# Chosing a random sample as first of markov chain
i=np.random.randint(pascal.shape[0])
z0=pascal.iloc[i]
y0=model_linear.predict(np.array([z0.drop('label')]))[0]

l=0
rej=0
print("Entering...")
while l<m:
    # choosing a random sample
    i=np.random.randint(pascal.shape[0])
    while i in lst:
        i=np.random.randint(pascal.shape[0])
    z1=pascal.iloc[i]
    y1=model_linear.predict(np.array([z1.drop('label')]))[0]
    n=lossF(z1['label'],y1)
    d=lossF(z0['label'],y0)
    p=min(1.0,n/d)

    flg=False
    # Deciding of acceptance of chosen sample and its probability in markov chain
    if rej>k:
        p=min(1.0,q*p)
        predProb.append([z1['label'],y1,p])
        markov=markov.append(z1)
        z0=z1
        l+=1
        flg=True
        rej=0
    elif p==1 and y1==y0:
        n=np.exp(-y1*z1['label'])
        d=np.exp(-y0*z0['label'])

        p=n/d
        p=min(p,1)
    if not(flg) and np.random.random() < p:
        predProb.append([z1['label'],y1,p])
        markov=markov.append(z1)
        z0=z1
        l+=1
        flg=True
        rej=0

    if not(flg):
        rej+=1
    lst.append(i)
yTest=[]
```

```
    yPred=[]
    for i in predProb:
        yTest.append(i[0])
        yPred.append(i[1])
    et=(m-metrics.accuracy_score(y_true=yTest, y_pred=yPred,normalize=False))/m
    print(et)
    at=(1/2)*np.log((1-et)/et)
    t+=1
    model_linear=train(markov)

markov
```

```
Entering...
0.5554285714285714
Entering...
0.42514285714285716
Entering...
0.42457142857142854
```

| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 | 20 | 21 | 22 | 23 | 24 | 25 | 26 | 27 | 28 | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **1140** | 222.0 | 211.0 | 223.0 | 222.0 | 223.0 | 219.0 | 215.0 | 220.0 | 220.0 | 223.0 | 220.0 | 222.0 | 226.0 | 226.0 | 226.0 | 227.0 | 226.0 | 230.0 | 228.0 | 225.0 | 226.0 | 219.0 | 228.0 | 230.0 | 229.0 | 228.0 | 228.0 | 230.0 | 229.0 | 22 |
| **3635** | 243.0 | 245.0 | 246.0 | 246.0 | 247.0 | 248.0 | 247.0 | 248.0 | 249.0 | 249.0 | 249.0 | 250.0 | 250.0 | 250.0 | 251.0 | 251.0 | 251.0 | 251.0 | 252.0 | 252.0 | 253.0 | 253.0 | 253.0 | 253.0 | 253.0 | 253.0 | 253.0 | 253.0 | 253.0 | 25 |
| **726** | 222.0 | 171.0 | 117.0 | 94.0 | 90.0 | 93.0 | 85.0 | 62.0 | 41.0 | 46.0 | 31.0 | 40.0 | 29.0 | 44.0 | 37.0 | 32.0 | 32.0 | 29.0 | 27.0 | 25.0 | 30.0 | 21.0 | 29.0 | 34.0 | 19.0 | 23.0 | 28.0 | 28.0 | 23.0 | 2 |
| **876** | 162.0 | 135.0 | 135.0 | 226.0 | 159.0 | 147.0 | 69.0 | 69.0 | 98.0 | 86.0 | 124.0 | 92.0 | 114.0 | 90.0 | 119.0 | 94.0 | 129.0 | 111.0 | 184.0 | 174.0 | 158.0 | 151.0 | 190.0 | 169.0 | 192.0 | 182.0 | 173.0 | 158.0 | 148.0 | 15 |
| **1343** | 247.0 | 246.0 | 248.0 | 247.0 | 245.0 | 243.0 | 243.0 | 242.0 | 243.0 | 243.0 | 244.0 | 244.0 | 245.0 | 245.0 | 245.0 | 244.0 | 244.0 | 245.0 | 245.0 | 244.0 | 246.0 | 244.0 | 245.0 | 244.0 | 244.0 | 245.0 | 245.0 | 246.0 | 247.0 | 24 |
| **...** | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... |
| **113** | 51.0 | 52.0 | 54.0 | 55.0 | 56.0 | 58.0 | 59.0 | 59.0 | 60.0 | 60.0 | 59.0 | 60.0 | 61.0 | 61.0 | 61.0 | 61.0 | 61.0 | 62.0 | 61.0 | 61.0 | 61.0 | 61.0 | 61.0 | 61.0 | 61.0 | 61.0 | 61.0 | 62.0 | 62.0 | 6 |
| **443** | 145.0 | 144.0 | 144.0 | 144.0 | 145.0 | 144.0 | 144.0 | 144.0 | 144.0 | 144.0 | 144.0 | 144.0 | 143.0 | 143.0 | 143.0 | 143.0 | 144.0 | 144.0 | 144.0 | 143.0 | 143.0 | 142.0 | 144.0 | 144.0 | 144.0 | 142.0 | 143.0 | 143.0 | 143.0 | 14 |
| **1449** | 185.0 | 185.0 | 186.0 | 190.0 | 195.0 | 200.0 | 206.0 | 220.0 | 214.0 | 200.0 | 198.0 | 197.0 | 200.0 | 196.0 | 200.0 | 199.0 | 197.0 | 196.0 | 196.0 | 198.0 | 200.0 | 200.0 | 201.0 | 208.0 | 213.0 | 212.0 | 218.0 | 204.0 | 201.0 | 20 |
| **659** | 132.0 | 133.0 | 133.0 | 134.0 | 134.0 | 134.0 | 134.0 | 134.0 | 134.0 | 134.0 | 135.0 | 135.0 | 136.0 | 136.0 | 136.0 | 135.0 | 137.0 | 137.0 | 137.0 | 137.0 | 137.0 | 137.0 | 138.0 | 138.0 | 138.0 | 139.0 | 138.0 | 137.0 | 138.0 | 13 |
| **563** | 182.0 | 182.0 | 183.0 | 183.0 | 184.0 | 183.0 | 183.0 | 185.0 | 182.0 | 184.0 | 184.0 | 183.0 | 183.0 | 183.0 | 185.0 | 184.0 | 184.0 | 185.0 | 186.0 | 185.0 | 188.0 | 183.0 | 186.0 | 186.0 | 187.0 | 186.0 | 186.0 | 187.0 | 188.0 | 18 |

1750 rows × 22501 columns

```
predProb
```

```
[[1.0, 6.0, 1.0],
 [5.0, 5.0, 1.0],
 [1.0, 0.0, 1.0],
 [1.0, 6.0, 1.0],
 [2.0, 0.0, 1.0],
 [1.0, 4.0, 1.0],
 [1.0, 0.0, 1.0],
 [4.0, 0.0, 1.0],
 [6.0, 6.0, 1.0],
 [4.0, 4.0, 1.0],
 [4.0, 4.0, 1.0],
 [0.0, 3.0, 1.0],
 [4.0, 4.0, 1.0],
 [1.0, 1.0, 1.0],
 [5.0, 5.0, 1.0],
```

```
[2.0, 2.0, 1.0],
[0.0, 0.0, 1.0],
[4.0, 4.0, 1.0],
[0.0, 0.0, 1.0],
[3.0, 3.0, 1.0],
[5.0, 6.0, 1.0],
[4.0, 0.0, 1.0],
[4.0, 4.0, 1.0],
[3.0, 3.0, 1.0],
[1.0, 3.0, 1.0],
[2.0, 0.0, 1.0],
[3.0, 0.0, 1.0],
[4.0, 0.0, 1.0],
[0.0, 0.0, 1.0],
[1.0, 1.0, 1.0],
[4.0, 4.0, 1.0],
[6.0, 6.0, 1.0],
[4.0, 0.0, 1.0],
[3.0, 4.0, 1.0],
[2.0, 2.0, 1.0],
[2.0, 0.0, 1.0],
[2.0, 2.0, 1.0],
[1.0, 1.0, 1.0],
[4.0, 4.0, 1.0],
[1.0, 1.0, 1.0],
[2.0, 0.0, 1.0],
[4.0, 4.0, 1.0],
[3.0, 3.0, 1.0],
[3.0, 3.0, 1.0],
[4.0, 4.0, 1.0],
[2.0, 2.0, 1.0],
[5.0, 5.0, 1.0],
[4.0, 0.0, 1.0],
[2.0, 2.0, 1.0],
[4.0, 0.0, 1.0],
[2.0, 2.0, 1.0],
[0.0, 0.0, 1.0],
[0.0, 0.0, 1.0],
[0.0, 0.0, 1.0],
[4.0, 4.0, 1.0],
[3.0, 3.0, 1.0],
[0.0, 0.0, 1.0],
[4.0, 4.0, 1.0],
[6.0, 4.0, 1.0],
[1.0, 1.0, 1.0],
[5.0, 5.0, 1.0],
```

Save data from generated markov chain

```
markov.to_csv("/content/drive/MyDrive/DM/SVMBMSamplesPascal.csv")
```

```
prob=[]
for i in predProb:
    prob.append(i[2])
```

```
markov['probability']=prob
```

```
markov.to_csv("/content/drive/MyDrive/DM/SVMBMSamplesPascalProbability.csv")
```

```
for i in lst:
```

```
    pascal=pascal.drop([1])
pascal.to_csv('/content/drive/MyDrive/DM/SVMBMremainingPascal.csv')
```

## SVM

```
train = pd.read_csv("/content/drive/MyDrive/DM/SVMBMSamplesPascal.csv")
test = pd.read_csv("/content/drive/MyDrive/DM/SVMBMremainingPascal.csv")
```

```
train = train.drop(train.columns[[0]], axis=1)
test = test.drop(test.columns[[0]], axis=1)
```

```
X_train = train.drop("label", axis = 1)
y_train = train["label"]

X_test = test.drop("label", axis = 1)
y_test = test["label"]
```

Linear kernel

```
model_linear = SVC(kernel='linear')
model_linear.fit(X_train, y_train)

# predict
y_pred = model_linear.predict(X_test)
print("accuracy:", metrics.accuracy_score(y_true=y_test, y_pred=y_pred), "\n")
```

```
accuracy: 0.22126436781609196
```

RBF kernel

```
model_linear = SVC(kernel='rbf')
model_linear.fit(X_train, y_train)

# predict
y_pred = model_linear.predict(X_test)
print("accuracy:", metrics.accuracy_score(y_true=y_test, y_pred=y_pred), "\n")
```

```
accuracy: 0.31157635467980294
```

Chi-squared kernel

```
from sklearn.metrics.pairwise import chi2_kernel

model_linear = SVC(kernel=chi2_kernel)
```

```
model_linear = SVC(kernel=chi2_kernel)
model_linear.fit(X_train, y_train)

y_pred = model_linear.predict(X_test)
print("accuracy:", metrics.accuracy_score(y_true=y_test, y_pred=y_pred), "\n")
```

```
accuracy: 0.22536945812807882
```

Hellinger kernel

```
def hellinger(X1, X2):

  return np.sqrt(np.dot(X1,X2.T))


model_linear = SVC(kernel=hellinger)
model_linear.fit(X_train, y_train)

# predict
y_pred = model_linear.predict(X_test)
print("accuracy:", metrics.accuracy_score(y_true=y_test, y_pred=y_pred), "\n")
```

```
accuracy: 0.1925287356321839
```

Intersection kernel

```
from sklearn.metrics.pairwise import euclidean_distances

def intersection(X1,X2):

  # X1= n1 x m
  # X2= n2 x m
  # result= n1xn2

  result = np.zeros((X1.shape[0],X2.shape[0]))
  X2=X2.T

  for i in range(len(X1)):
    # iterate through columns of Y
    for j in range(len(X2[0])):
      # iterate through rows of Y
      val=float('+inf')
      for k in range(len(X2)):
        val = min(val,X1[i][k] * X2[k][j])

      result[i][j]=val
```

```
  return result


model_linear = SVC(kernel=intersection)
model_linear.fit(X_train, y_train)

# predict
y_pred = model_linear.predict(X_test)
print("accuracy:", metrics.accuracy_score(y_true=y_test, y_pred=y_pred), "\n")

# Taking too much time.
```