# Markov Sampling for Pascal dataset

```
# 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)
```

Step-I

```
# initialise parameters
markov= pd.DataFrame(columns = pascal.columns)
uniqCls=list(np.sort(pascal['label'].unique()))
```

```
classCNT=len(uniqCls)
limit=250
m=classCNT*limit
mcls={i:0 for i in uniqCls}
```

```
# Chose parameters for markov sampling
k=5
q=1.2
acc=0
```

```
# Train a linear Model on N[here 1000] 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 = 0.7, random_state = 101)
model_linear = SVC(kernel='linear')
model_linear.fit(X_train, y_train)

# predict
y_pred = model_linear.predict(X_test)
```

```
y_pred
```

```
array([4., 4., 0., ..., 6., 2., 1.])
```

Step-II

```
# 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]
if m%classCNT==0:
    mcls[z0['label']]+=1
```

```
predProb=[]
```

```
# Utility Function for loop condition
def exist(dic,limit):
    for i,val in dic.items():
        if val<limit:
            return True
    return False
```

```
# Utility loss Function
def lossF(actual,pred):
    if actual==pred:
```

```
    if actual==pred:
        return 1.0
    return np.exp(-2)
```

```
lst=[]
```

Step-III TO Step-VI

```
# Run loop till limit is reached for each class
while exist(mcls,limit):

    # choosing a random sample
    i=np.random.randint(pascal.shape[0])
    while i in lst:
        i=np.random.randint(pascal.shape[0])
    lst.append(i)
    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=n/d

    # Deciding of acceptance of chosen sample and its probability in markov chain
    if acc==k:
        acc=0
        p2=q*p
        p2=min(p2,1)
        predProb.append([z1['label'],y1,p2])
        markov=markov.append(z1)
        z0=z1
        mcls[z1['label']]+=1
        acc+=1
    elif p==1 and z0['label']==z1['label']:
        n=np.exp(-y1*z1['label'])
        d=np.exp(-y0*z0['label'])

        p1=n/d
        p1=min(p1,1)
        predProb.append([z1['label'],y1,p1])
        markov=markov.append(z1)
        z0=z1
        mcls[z1['label']]+=1
        acc+=1
    elif p<1:
        predProb.append([z1['label'],y1,p])
        markov=markov.append(z1)
        z0=z1
        mcls[z1['label']]+=1
```

```
        acc+=1
    elif p==1 and z0['label']!=z1['label']:
        predProb.append([z1['label'],y1,p])
        markov=markov.append(z1)
        z0=z1
        mcls[z1['label']]+=1
        acc+=1
markov
```

| | 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 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **3280** | 54.0 | 52.0 | 83.0 | 183.0 | 188.0 | 189.0 | 189.0 | 191.0 | 186.0 | 186.0 | 183.0 | 184.0 | 187.0 | 190.0 | 190.0 | 188.0 | 190.0 | 188.0 | 188.0 | 190.0 | 191.0 | 195.0 | 190.0 | 192.0 | 191.0 | 194.0 | 193.0 | 193.0 | 191.0 |
| **3009** | 20.0 | 64.0 | 115.0 | 116.0 | 116.0 | 119.0 | 119.0 | 117.0 | 119.0 | 114.0 | 118.0 | 121.0 | 119.0 | 124.0 | 122.0 | 121.0 | 120.0 | 121.0 | 122.0 | 121.0 | 122.0 | 123.0 | 123.0 | 124.0 | 126.0 | 123.0 | 122.0 | 125.0 | 123.0 |
| **4378** | 184.0 | 207.0 | 203.0 | 204.0 | 190.0 | 184.0 | 93.0 | 169.0 | 155.0 | 149.0 | 147.0 | 165.0 | 158.0 | 104.0 | 145.0 | 155.0 | 154.0 | 170.0 | 168.0 | 149.0 | 144.0 | 150.0 | 174.0 | 158.0 | 117.0 | 129.0 | 157.0 | 146.0 | 144.0 |
| **19** | 139.0 | 156.0 | 151.0 | 147.0 | 165.0 | 152.0 | 150.0 | 200.0 | 131.0 | 151.0 | 212.0 | 143.0 | 166.0 | 151.0 | 155.0 | 215.0 | 151.0 | 179.0 | 154.0 | 169.0 | 206.0 | 218.0 | 242.0 | 213.0 | 238.0 | 203.0 | 230.0 | 184.0 | 208.0 |
| **457** | 200.0 | 200.0 | 202.0 | 201.0 | 202.0 | 204.0 | 204.0 | 205.0 | 205.0 | 206.0 | 205.0 | 204.0 | 206.0 | 204.0 | 205.0 | 206.0 | 207.0 | 204.0 | 203.0 | 202.0 | 200.0 | 200.0 | 199.0 | 199.0 | 202.0 | 202.0 | 203.0 | 206.0 | 207.0 |
| **...** | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... |
| **2289** | 30.0 | 25.0 | 22.0 | 26.0 | 12.0 | 24.0 | 24.0 | 22.0 | 15.0 | 23.0 | 16.0 | 24.0 | 23.0 | 31.0 | 56.0 | 68.0 | 68.0 | 59.0 | 31.0 | 29.0 | 32.0 | 30.0 | 37.0 | 32.0 | 41.0 | 36.0 | 49.0 | 50.0 | 59.0 |
| **3903** | 11.0 | 12.0 | 13.0 | 12.0 | 12.0 | 12.0 | 12.0 | 12.0 | 12.0 | 12.0 | 12.0 | 12.0 | 12.0 | 12.0 | 11.0 | 11.0 | 11.0 | 10.0 | 34.0 | 67.0 | 71.0 | 61.0 | 55.0 | 58.0 | 58.0 | 57.0 | 53.0 | 56.0 | 54.0 |
| **2666** | 62.0 | 71.0 | 27.0 | 47.0 | 40.0 | 49.0 | 39.0 | 45.0 | 54.0 | 27.0 | 32.0 | 39.0 | 82.0 | 69.0 | 68.0 | 51.0 | 34.0 | 37.0 | 36.0 | 28.0 | 54.0 | 77.0 | 91.0 | 107.0 | 105.0 | 106.0 | 37.0 | 118.0 | 149.0 |
| **490** | 184.0 | 183.0 | 185.0 | 186.0 | 184.0 | 183.0 | 183.0 | 186.0 | 188.0 | 188.0 | 187.0 | 186.0 | 189.0 | 191.0 | 188.0 | 185.0 | 192.0 | 190.0 | 189.0 | 191.0 | 191.0 | 189.0 | 190.0 | 193.0 | 191.0 | 194.0 | 191.0 | 192.0 | 192.0 |
| **1732** | 210.0 | 199.0 | 153.0 | 221.0 | 224.0 | 227.0 | 229.0 | 178.0 | 176.0 | 225.0 | 181.0 | 226.0 | 188.0 | 230.0 | 223.0 | 218.0 | 237.0 | 234.0 | 238.0 | 233.0 | 234.0 | 183.0 | 235.0 | 233.0 | 235.0 | 234.0 | 234.0 | 234.0 | 234.0 |

2417 rows × 22501 columns

```
markov
```

```
predProb
```

```
    [0.0, 0.0, 1],
    [4.0, 5.0, 1.0],
    [2.0, 4.0, 1.0],
    [3.0, 4.0, 1.0],
    [4.0, 6.0, 1.0],
    [3.0, 4.0, 1],
    [1.0, 2.0, 1.0],
    [4.0, 5.0, 0.1353352832366127],
    [1.0, 0.0, 1.0],
    [0.0, 0.0, 1.0],
    [0.0, 2.0, 1],
    [3.0, 4.0, 1.0],
    [2.0, 4.0, 1.0],
    [4.0, 6.0, 1.0],
    [5.0, 4.0, 1.0],
    [0.0, 0.0, 1],
    [2.0, 0.0, 1.0],
    [5.0, 6.0, 1.0],
    [6.0, 3.0, 1.0],
    [1.0, 0.0, 1.0],
    [0.0, 0.0, 1],
    [3.0, 4.0, 1.0],
    [5.0, 2.0, 1.0],
    [4.0, 1.0, 1.0],
    [6.0, 3.0, 1.0],
    [2.0, 4.0, 1],
    [4.0, 2.0, 1.0],
    [4.0, 0.0, 1],
    [4.0, 2.0, 0.018315638888734182],
    [6.0, 2.0, 1.0],
    [2.0, 3.0, 1],
    [6.0, 0.0, 1.0],
    [4.0, 5.0, 1.0],
    [0.0, 4.0, 1.0],
    [4.0, 6.0, 1.0],
    [4.0, 4.0, 1],
    [4.0, 2.0, 0.018315638888734182],
    [1.0, 2.0, 1.0],
    [1.0, 4.0, 0.1353352832366127],
    [4.0, 4.0, 1.0],
    [0.0, 2.0, 1],
    [0.0, 4.0, 1.0],
    [2.0, 6.0, 1.0],
    [1.0, 0.0, 1.0],
    [2.0, 0.0, 0.1353352832366127],
    [3.0, 2.0, 1],
    [2.0, 0.0, 1.0],
    [0.0, 4.0, 1.0],
    [5.0, 4.0, 1.0],
    [2.0, 4.0, 1.0],
    [4.0, 4.0, 1],
    [5.0, 2.0, 1.0],
    [4.0, 6.0, 1.0],
    [4.0, 5.0, 1.1253517471925913e-07],
    [4.0, 2.0, 0.018315638888734182],
    [6.0, 4.0, 1],
    [6.0, 0.0, 1],
    [0.0, 3.0, 1.0],
    [2.0, 6.0, 1.0],
    [6.0, 2.0, 1.0]
```

Save data from generated markov chain

```
markov.to_csv("/content/drive/MyDrive/DM/markovSamplesPascal1.csv")
```

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

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

```
markov.to_csv("/content/drive/MyDrive/DM/markovSamplesPascalProbability1.csv")
```

```
for i in lst:
    pascal=pascal.drop([i])
pascal.to_csv('/content/drive/MyDrive/DM/remainingPascal1.csv')
```

## SVM

```
train = pd.read_csv("/content/drive/MyDrive/DM/markovSamplesPascal1.csv")
test = pd.read_csv("/content/drive/MyDrive/DM/remainingPascal1.csv")
```

```
print(train.shape,test.shape)
```

```
(2417, 22502) (879, 22502)
```

```
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: 0.20136518771331058
```

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: 0.2867988190636862
```

Chi-squared kernel

```
from sklearn.metrics.pairwise import 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: 0.2193167439898777
```

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: 0.18430034129692832
```

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.
```