Markov Sampling for Letters 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
letters = pd.read_csv("/content/drive/MyDrive/DM/letter-recognition.csv")
letters.columns = ['letter', 'xbox', 'ybox', 'width', 'height', 'onpix', 'xbar','ybar', 'x2bar', 'y2bar', 'xybar', 'x2ybar', 'xy2bar', 'xedge','
```

Step-I

```
# initialise parameters
markov= pd.DataFrame(columns = letters.columns)
uniqChar=list(np.sort(letters['letter'].unique()))
classCNT=len(uniqChar)
limit=100
m=classCNT*limit
charNo={}
c=0
for i in uniqChar:
    charNo[i]=c
    c+=1
mAZ={i:0 for i in uniqChar}
```

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

```
acc=0
```

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

X_scaled = scale(X)

# train test split
X_train, X_test, y_train, y_test = train_test_split(X_scaled, y, test_size = 0.9, 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(['X', 'M', 'E', ..., 'M', 'B', 'J'], dtype=object)
```

Step-II

```
# Chosing a random sample as first of markov chain
i=np.random.randint(letters.shape[0])
z0=letters.iloc[i]
y0=model_linear.predict(np.array([z0.drop('letter')]))[0]
if m%classCNT==0:
    mAZ[z0['letter']]+=1
```

```
d={}
for i,val in z0.items():
    print(i,val)
    d[i]=val
markov.append(d,ignore_index=True)
markov
```

```
letter 0
xbox 2
ybox 3
width 2
```

```
predProb=[]
```

```
xbar 8
```

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

```
yedgex 8
```

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

```
# Utility Function for getting class index
def getNo(ch):
    return charNo[ch]
```

```
lst=[]
```

Step-III TO Step-VI

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

    # choosing a random sample
    i=np.random.randint(letters.shape[0])
    while i in lst:
        i=np.random.randint(letters.shape[0])
    z1=letters.iloc[i]
    y1=model_linear.predict(np.array([z1.drop('letter')]))[0]
    n=lossF(z1['letter'],y1)
    d=lossF(z0['letter'],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['letter'],y1,p2])
        markov=markov.append(z1)
```

```
        z0=z1
        mAZ[z1['letter']]+=1
        acc+=1
        lst.append(i)
    elif p==1 and z0['letter']==z1['letter']:
        n=np.exp(-getNo(y1)*getNo(z1['letter']))
        d=np.exp(-getNo(y0)*getNo(z0['letter']))

        p1=n/d
        p1=min(p1,1)
        predProb.append([z1['letter'],y1,p1])
        markov=markov.append(z1)
        z0=z1
        mAZ[z1['letter']]+=1
        acc+=1
        lst.append(i)
    elif p<1:
        predProb.append([z1['letter'],y1,p])
        markov=markov.append(z1)
        z0=z1
        mAZ[z1['letter']]+=1
        acc+=1
        lst.append(i)
    elif p==1 and z0['letter']!=z1['letter']:
        predProb.append([z1['letter'],y1,p])
        markov=markov.append(z1)
        z0=z1
        mAZ[z1['letter']]+=1
        acc+=1
        lst.append(i)
```

	letter	xbox	ybox	width	height	onpix	xbar	ybar	x2bar	y2bar	xybar	x2ybar	xy2bar	xedge	xedgey	yedge	yedgex
80	P	8	14	7	8	4	5	10	6	3	12	5	4	4	10	4	8
18504	I	0	0	0	0	0	7	7	4	4	7	6	8	0	8	0	8
16125	A	2	6	4	4	2	8	3	2	2	7	1	8	2	7	3	7
5633	K	6	10	5	5	3	7	8	3	6	9	9	9	6	11	3	6
5007	G	5	10	6	7	5	6	6	6	6	10	7	13	3	9	5	9
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
19048	G	3	7	4	5	3	6	6	5	4	6	6	9	2	8	3	8
13940	A	4	9	6	6	5	10	3	1	2	7	3	9	5	5	3	7
15117	X	4	9	6	7	4	7	7	4	9	6	6	8	3	8	7	8
6381	X	4	9	4	6	1	7	7	4	4	7	6	8	3	8	4	8
6072	E	3	6	4	4	3	6	7	6	8	6	4	9	3	8	6	9

11451 rows × 17 columns

markov

	letter	xbox	ybox	width	height	onpix	xbar	ybar	x2bar	y2bar	xybar	x2ybar	xy2bar	xedge	xedgey	yedge	yedgex
80	P	8	14	7	8	4	5	10	6	3	12	5	4	4	10	4	8
18504	I	0	0	0	0	0	7	7	4	4	7	6	8	0	8	0	8
16125	A	2	6	4	4	2	8	3	2	2	7	1	8	2	7	3	7
5633	K	6	10	5	5	3	7	8	3	6	9	9	9	6	11	3	6
5007	G	5	10	6	7	5	6	6	6	6	10	7	13	3	9	5	9
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
19048	G	3	7	4	5	3	6	6	5	4	6	6	9	2	8	3	8
13940	A	4	9	6	6	5	10	3	1	2	7	3	9	5	5	3	7
15117	X	4	9	6	7	4	7	7	4	9	6	6	8	3	8	7	8
6381	X	4	9	4	6	1	7	7	4	4	7	6	8	3	8	4	8
6072	E	3	6	4	4	3	6	7	6	8	6	4	9	3	8	6	9

11451 rows × 17 columns

predProb

```
[['P', 'C', 1.0],
 ['I', 'C', 1.0],
 ['A', 'Q', 1.0],
 ['K', 'C', 1.0],
 ['G', 'C', 1.0],
 ['R', 'E', 1],
 ['K', 'E', 1.0],
 ['F', 'C', 1.0],
 ['K', 'C', 1.0],
 ['Y', 'E', 1.0],
 ['X', 'E', 1],
 ['P', 'E', 1.0],
 ['O', 'C', 1.0],
 ['Q', 'C', 1.0],
 ['G', 'C', 1.0],
 ['J', 'E', 1],
 ['F', 'E', 1.0],
 ['O', 'C', 1.0],
 ['K', 'C', 1.0],
 ['H', 'C', 1.0],
 ['Q', 'C', 1],
 ['Y', 'C', 1.0],
 ['U', 'C', 1.0],
 ['F', 'E', 1.0],
 ['H', 'C', 1.0],
 ['Z', 'E', 1],
 ['L', 'C', 1.0],
 ['A', 'Q', 1.0],
 ['B', 'E', 1.0],
 ['Z', 'E', 1.0],
 ['A', 'G', 1],
 ['U', 'C', 1.0],
 ['G', 'C', 1.0],
 ['Y', 'E', 1.0],
```

```
    ['V', 'C', 1.0],
    ['A', 'E', 1],
    ['U', 'C', 1.0],
    ['Q', 'E', 1.0],
    ['F', 'E', 1.0],
    ['X', 'E', 1.0],
    ['X', 'E', 1],
    ['Q', 'E', 1.0],
    ['T', 'E', 1.0],
    ['L', 'E', 1.0],
    ['L', 'C', 1.0],
    ['E', 'E', 1],
    ['K', 'E', 1.0],
    ['N', 'E', 1.0],
    ['H', 'E', 1.0],
    ['S', 'E', 1.0],
    ['O', 'C', 1],
    ['W', 'C', 1.0],
    ['O', 'E', 1.0],
    ['T', 'E', 1.0],
    ['L', 'C', 1.0],
    ['L', 'C', 1],
    ['X', 'E', 1.0],
    ['J', 'C', 1.0],
    ['J', 'E', 1.522979744471263e-08],
```

Save data from generated markov chain

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

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

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

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

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

SVM

```
train = pd.read_csv("/content/drive/MyDrive/DM/Letter Dataset Samples/markovSamplesLetter.csv")
test = pd.read_csv("/content/drive/MyDrive/DM/Letter Dataset Samples/remainingLetter.csv")
```

```
train
```

	letter	xbox	ybox	width	height	onpix	xbar	ybar	x2bar	y2bar	xybar	x2ybar	xy2bar	xedge	xedgey	yedge	yedgex
0	P	8	14	7	8	4	5	10	6	3	12	5	4	4	10	4	8
1	I	0	0	0	0	0	7	7	4	4	7	6	8	0	8	0	8
2	A	2	6	4	4	2	8	3	2	2	7	1	8	2	7	3	7
3	K	6	10	5	5	3	7	8	3	6	9	9	9	6	11	3	6
4	G	5	10	6	7	5	6	6	6	6	10	7	13	3	9	5	9
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
11446	G	3	7	4	5	3	6	6	5	4	6	6	9	2	8	3	8
11447	A	4	9	6	6	5	10	3	1	2	7	3	9	5	5	3	7
11448	X	4	9	6	7	4	7	7	4	9	6	6	8	3	8	7	8
11449	X	4	9	4	6	1	7	7	4	4	7	6	8	3	8	4	8
11450	E	3	6	4	4	3	6	7	6	8	6	4	9	3	8	6	9

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

```
(11451, 18) (7618, 18)
```

```
train = train.drop(train.columns[[0]], axis=1)
test = test.drop(test.columns[[0]], axis=1)
train.columns = ['letter', 'xbox', 'ybox', 'width', 'height', 'onpix', 'xbar','ybar', 'x2bar', 'y2bar', 'xybar', 'x2ybar', 'xy2bar', 'xedge','xe
test.columns = ['letter', 'xbox', 'ybox', 'width', 'height', 'onpix', 'xbar','ybar', 'x2bar', 'y2bar', 'xybar', 'x2ybar', 'xy2bar', 'xedge','xed
```

```
X_train = np.array(train.drop("letter", axis = 1))
y_train = np.array(train["letter"])

X_test = np.array(test.drop("letter", axis = 1))
y_test = np.array(test["letter"])
```

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: 0.8475977946967708
```

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: 0.9057495405618272
```

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: 0.9558939354161197
```

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: 0.7760567077973222
```

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.
```