K-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')
```

```
Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.mount("/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', 'xyb
```

Step-I

```
# initialise parameters
markov= pd.DataFrame(columns = letters.columns)
uniqChar=list(np.sort(letters['letter'].unique()))
classCNT=len(uniqChar)
```

```
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
```

```
# 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 J
xbox 3
ybox 6
width 4
height 4
onpix 1
xbar 8
ybar 6
x2bar 5
y2bar 6
xybar 15
x2ybar 7
xy2bar 12
xedge 1
xedgey 6
yedge 1
yedgex 7
```

letter	xbox	ybox	width	height	onpix	xbar	ybar	x2bar	y2bar	xybar	x2ybar	xy2bar	xedge	xedgey	yed

```
predProb=[]
```

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

```
            return True
    return False
```

```
# 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]
```

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

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

    # 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)

    return model_linear
```

```
lst=[]
```

Step-III TO Step-VII

```
# Loop markov chain generator k times
for km in range(2):
    # Reset parameters for next markov chain
    if km!=0:
        predProb=[]
```

```
        model_linear=train(markov)
        markov= pd.DataFrame(columns = letters.columns)
        mAZ={i:0 for i in uniqChar}
    lst=[]

    # 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
    print("Entering...")
    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)
```

```
Entering...
Entering...
---------------------------------------------------------------------------
KeyboardInterrupt                         Traceback (most recent call last)
<ipython-input-19-eed1c805cd77> in <module>()
```

```
1 markov
```

```
---> 23          z1=letters.iloc[i]
```

```
1 predProb
```

Save data from generated markov chain

```
raise cast_failure)
```

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

```
--> 539          if maybe_castable(arr) and not copy and dtype is None:
```

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

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

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

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

SVM

```
1 train = pd.read_csv("/content/drive/MyDrive/DM/KmarkovSamplesLetters.csv")
2 test = pd.read_csv("/content/drive/MyDrive/DM/KmarkovLettersRemaining.csv")
```

```
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
test.columns = ['letter', 'xbox', 'ybox', 'width', 'height', 'onpix', 'xbar','ybar', 'x2bar', 'y2bar', 'xybar'
```

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

X_test = test.drop("letter", axis = 1)
y_test = 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: 0.8147690305790501
```

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: 0.8815875081327261
```

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: 0.8559531554977229
```

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: 0.7474951203643462
```

Intersection kernel

```
from sklearn.metrics.pairwise import euclidean_distances

def intersection(X1,X2):
```

```
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.
```

```
---------------------------------------------------------------------------
KeyboardInterrupt                         Traceback (most recent call last)
<ipython-input-27-8790cc471789> in <module>()
     24
     25 model_linear = SVC(kernel=intersection)
---> 26 model_linear.fit(X_train, y_train)
     27
     28 # predict

                                3 frames
<ipython-input-27-8790cc471789> in intersection(X1, X2)
     16       val=float('+inf')
     17       for k in range(len(X2)):
---> 18         val = min(val,X1[i][k] * X2[k][j])
     19
     20       result[i][j]=val

KeyboardInterrupt:
```

SEARCH STACK OVERFLOW