```
# 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", force_remount=True).
```

```
# 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','
```

```
# 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
```

```
# Chose parameters for markov sampling
k=5
q=1.2
rej=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)
```

```
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)
```

```
# 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]
```

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

```
letter Z
xbox 4
ybox 10
width 5
height 8
onpix 3
xbar 7
ybar 7
x2bar 4
y2bar 15
xybar 9
x2ybar 6
xy2bar 8
xedge 0
xedgey 8
yedge 8
yedgex 8
```

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

```
predProb=[]
```

```
# 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
```

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

    l=0
    rej=0
    print("Entering...")
    while l<m:
        # 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=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['letter'],y1,p])
            markov=markov.append(z1)
```

```
            markov=markov.append(z1)
            z0=z1
            l+=1
            flg=True
            rej=0
        elif p==1 and y1==y0:
            n=np.exp(-getNo(y1)*getNo(z1['letter']))
            d=np.exp(-getNo(y0)*getNo(z0['letter']))

            p=n/d
            p=min(p,1)
        if not(flg) and np.random.random() < p:
            predProb.append([z1['letter'],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
        if at<0:
            t-=1
        else:
            model_linear=train(markov)

markov
```

```
Entering...
0.26269230769230767
Entering...
0.2823076923076923
Entering...
0.2742307692307692
```

	letter	xbox	ybox	width	height	onpix	xbar	ybar	x2bar	y2bar	xybar	x2ybar	xy2bar	xedge	xedgey	yedge	yedgex
17445	M	3	6	4	4	2	8	6	11	1	6	9	8	7	6	0	8
13133	E	6	11	6	8	4	3	7	6	11	7	6	14	0	8	8	7
9994	B	8	11	6	6	4	9	6	6	6	11	4	9	6	6	6	10
19410	F	7	13	6	7	4	8	9	2	5	11	5	4	3	10	7	7

Save data from generated markov chain

...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...

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

14214	F	3	4	3	3	3	6	8	3	7	11	7	9	2	8	4	8

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

2695	L	3	9	4	6	2	7	4	0	9	9	2	11	0	7	2	8

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

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

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

SVM

```
train = pd.read_csv("/content/drive/MyDrive/DM/SVMBMSamplesLetters.csv")
test = pd.read_csv("/content/drive/MyDrive/DM/SVMBMLettersRemaining.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', 'x2ybar', 'xy2bar', 'xedge','xe
test.columns = ['letter', 'xbox', 'ybox', 'width', 'height', 'onpix', 'xbar','ybar', 'x2bar', 'y2bar', 'xybar', 'x2ybar', 'xy2bar', 'xedge','xed
```

```
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: 0.8165264492336061
```

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: 0.8200414889938918
```

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: 0.8796819177134955
```

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
```

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

```
accuracy: 0.7778033882678345
```

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")
```

```
accuracy: 0.033248818716146135
```