Aman Chauhan - achauha3
Khantil Choksi – khchoksi

Q1 D-Separation

(1) d-sep(B,D / A)

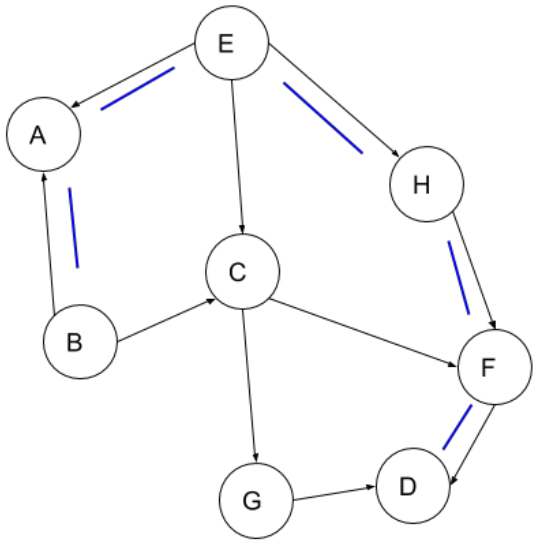
$X_1 = B$

$X_3 = D$

$X_2 = A$

B → A ← E → H → F → D

- A is converging connection and it is in X_2 ; therefore doesn't block information
- E is diverging connection and it is not in X_2 ; therefore doesn't block information
- H is serial connection and it is not in X_2 ; therefore doesn't block information
- F is serial connection and it is not in X_2 ; therefore doesn't block information
- Therefore, for this given path, B and D are not-separated given A. (According to blue path shown in figure)



There are also other paths present which is also not blocking information.

B → A ← E → C → F → D

B → A ← E → C ← G → D

B → C ← G → D

Conclusion: B and D are not d-separated given {A}.

(2) d-sep(A,D / {C,H})

$X_1 = A$

$X_3 = D$

$X_2 = \{C,H\}$

A ← E → H → F → D

- E is diverging connection and it is not in X_2 ; therefore doesn't block information
- H is serial connection and in X_2 ; therefore does block information
- Therefore, A and D are separated by this path.

Consider this alternative path:

A ← E → C → F → D

- E is diverging connection and it is not in X_2 ; therefore doesn't block information
- C is serial connection and in X_2 ; therefore does block information
- Therefore, A and D are separated by this path.

Consider this alternative path:

A ← E → C → G → D

- E is diverging connection and it is not in X_2 ; therefore doesn't block information
- C is serial connection and in X_2 ; therefore does block information
- Therefore, A and D are separated by this path.

Consider the following alternative path:

A ← B → C → G → D

A ← E → C → F → D

In all above paths,

- C is serial connection and in X_2 ; therefore does block information
- Therefore, A and D are separated by this path.

Conclusion: Therefore, A and D are d-separated given {C,H}.

(3) d-sep(A,B / {F,E})

$X_1 = A$

$X_3 = B$

$X_2 = \{F,E\}$

A → E → G → F → B

- E is serial connection and it is in X_2 ; therefore does block information

- Therefore, A and B are d-separated given {F,E}.

Consider this path:
A → E ← H → F → B

- E is converging connection and it is in X_2 ; therefore doesn't block information.
- H is diverging connection and not in X_2 ; therefore doesn't block information.
- F is serial connection and not in X_2 ; therefore doesn't block information.
- Therefore, A and B are not d-separated.

Conclusion: Therefore, we found this one path which proves A and B are not d-separated given {F,E}.

(4) d-sep(C,D / {B})

X_1 = C
X_3 = D
X_2 = {B}

C ← E → G → F ← D

- E is diverging connection and it is in X_2 ; therefore doesn't block information.
- G is serial connection and not in X_2 ; therefore doesn't block information.
- F is converging connection and not in X_2 ; therefore does block information.
- Therefore, C and D are d-separated given {B}.

Consider this path:
C ← E → H → F ← D

- E is diverging connection and it is in X_2 ; therefore doesn't block information.
- H is serial connection and not in X_2 ; therefore doesn't block information.
- F is converging connection and not in X_2 ; therefore does block information.
- Therefore, C and D are d-separated given {B}.

Conclusion: C and D are d-separated given {B}.

Q2

(a) Compute P(E)

P(E) = P(E | B) * P(B) + P(E |~ B) * P(~B)

P(B) = P(B | A) * P(A) + P(B | ~A) * P(~A)

P(E) = P(E | B) * P(B | A) * P(A) +
P(E | B) * P(B | ~A) * P(~A) +
P(E |~ B) * P(~B | A) * P(A) +
P(E |~ B) * P(~B | ~A) * P(~A)

= 0.6 * 0.2 * 0.75 +
0.6 * 0.5 * 0.25 +
0.3 * 0.8 * 0.75 +
0.3 * 0.5 * 0.25
= 0.09 + 0.075 + 0.18 + 0.0375
= 0.3825

(b) Compute P(~B,C,D,E)

P(~B,C,D,E) = P(D|~B,C) * P(C) * P(E | ~B) * P(~B)
= P(D|~B,C) * P(C | A) * P(E | ~B) * P(~B | A) * P(A)
+ P(D|~B,C) * P(C | ~A) * P(E | ~B) * P(~B | ~A) * P(~A)
= 0.1 * 0.7 * 0.3 * 0.8 * 0.75 + (0.1 * 0.25 * 0.3 * 0.5 * 0.25)
= 0.0126 + 0.0009375
= 0.0135375

(c) Compute P(D | A)

P(A,B,C,D) = P(D | B,C) * P(B|A) * P(C|A) * P(A)
= 0.3 * 0.2 * 0.7 * 0.75
= 0.0315

P(A,B,~C,D) = P(D | B,~C) * P(B|A) * P(~C|A) * P(A)
= 0.25 * 0.2 * 0.3 * 0.75
= 0.01125
P(A,~B,C,D) = P(D | ~B,C) * P(~B|A) * P(C|A) * P(A)
= 0.1 * 0.8 * 0.7 * 0.75
= 0.042

P(A,~B,~C,D) = P(D | ~B,~C) * P(~B|A) * P(~C|A) * P(A)

= 0.35 * 0.8 * 0.3 * 0.75
= 0.063

P(D,A) = P(A,B,C,D) + P(A,B,~C,D) + P(A,~B,C,D) + P(A,~B,~C,D)
= 0.0315 + 0.01125 + 0.042 + 0.063
= 0.14775

P(D | A) = P(D,A) / P(A)
= 0.14775/ 0.75
= 0.197

Q3 - Linear Regression

Import Libraries

```
from sklearn.model_selection import LeaveOneOut, cross_val_predict
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error
import numpy as np
import math
import sys
import os
```

(a) Given the following three data points of (x, y): (1, 2), (2, 1), (0, -1), try to use a linear regression $y = \beta_0 + \beta_1 x$ to predict y. Determine the values of β_1 and β_0 and show each step of your work.

x	y	xy	x^2	y^2
1	2	2	1	4
2	1	2	4	1
0	-1	0	0	1
$\sum_i x_i = 3$	$\sum_i y_i = 2$	$\sum_i x_i y_i = 4$	$\sum_i x_i^2 = 5$	$\sum_i y_i^2 = 6$

We want to predict the dependent random variable y , and it is given by $y' = \beta_0 + \beta_1 x$

$\beta_0 = \frac{\sum_i y_i \sum_i x_i^2 - \sum_i x_i \sum_i x_i y_i}{n(\sum_i x_i^2) - (\sum_i x_i)^2} = \frac{2*5-3*4}{3*5-3*3} = \frac{-1}{3}$

$\beta_1 = \frac{n(\sum_i x_i y_i) - \sum_i x_i \sum_i y_i}{n(\sum_i x_i^2) - (\sum_i x_i)^2} = \frac{3*4-3*2}{3*5-3*3} = 1$

Thus the equation is $y' = \frac{-1}{3} + x$

(b) Linear Regression Programming Assignment

Apply the following three linear regressions: (1) $y = \alpha_0 + \alpha_1 x_1 + \alpha_2 x_2 + \alpha_3 x_3 + \alpha_4 x_4$ ***(2)*** $y = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \beta_3 x_3 + \beta_4 x_4$ ***(3)*** $y = \gamma_0 + \gamma_1 x_1 + \gamma_2 x_2 + \gamma_3 x_3 + \gamma_4 x_4$ ***to the provided data file "hw3q3(b).csv", which is from a combined cycle power plant dataset (https://archive.ics.uci.edu/ml/datasets/Combined+Cycle+Power+Plant). In the given data file, xi are features and y is the prediction target which indicates hourly electrical energy output.***

(i) Load the data. Fit the whole dataset to the three linear regression models, respectively. Report the coefficients (alphas, betas, gammas) of the three models.

```
def data_and_headers(filename):
    data = None
    with open(filename) as fp:
        data = [x.strip().split(',') for x in fp.readlines()]
    headers = data[0]
    headers = np.asarray(headers)
    class_field = len(headers) - 1
    data_x = [[float(x[i]) for i in range(class_field)] for x in data[1:]]
    data_x = np.asarray(data_x)
    data_y = [[float(x[i]) for i in range(class_field, class_field + 1)] for x in data[1:]]
    data_y = np.asarray(data_y)
    return headers, data_x, data_y
```

```
headers, features_x, labels_y = data_and_headers('Data' + os.sep + 'hw3q3(b).csv')
```

```
modela = LinearRegression().fit(features_x, labels_y.flatten())
modelb = LinearRegression().fit(features_x**2, labels_y.flatten())
modelc = LinearRegression().fit(features_x**3, labels_y.flatten())
print('Coefficients of Simple LR - \t{}, Intercept - {:.4f}'.format(modela.coef_,modela.intercept_))
print('Coefficients of Quadratic LR - \t{}, Intercept - {:.4f}'.format(modelb.coef_,modelb.intercept_))
print('Coefficients of Cubic LR - \t{}, Intercept - {:.4f}'.format(modelc.coef_,modelc.intercept_))
```

Output -

```
Coefficients of Simple LR -     [-12.38926535   2.80059786 -12.32760055
-64.67916351], Intercept - 500.2071
Coefficients of Quadratic LR -  [ -6.05581029   7.28426322 -15.38358205
-54.34236701], Intercept - 477.0979
Coefficients of Cubic LR -  [  1.24877688  16.3800381  -24.20328807
-43.63328247], Intercept - 466.2837
```

The above output can be interpreted as follows, where $c_i \in \{\alpha_i, \beta_i, \gamma_i\}$ and $i \in \{0, 1, 2, 3, 4\}$-

Equation	c_0	c_1	c_2	c_3	c_4
$y = \alpha_0 + \alpha_1 x_1 + \alpha_2 x_2 + \alpha_3 x_3 + \alpha_4 x_4$	500.2071	-12.3892	2.8005	-12.3276	-64.6791
$y = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \beta_3 x_3 + \beta_4 x_4$	477.0979	-6.0558	7.2842	-15.3836	-54.3423
$y = \gamma_0 + \gamma_1 x_1 + \gamma_2 x_2 + \gamma_3 x_3 + \gamma_4 x_4$	466.2837	1.2487	16.3800	-24.2032	-43.6333

(ii) Use leave-one-out cross validation to determine the RMSE (root mean square error) for the three models. Specifically, in each fold, fit the training data to the model to determine the coefficients, then apply the coefficients to get predicted label for testing data (You don't need to report the coefficients in each fold). Report RMSE for the three models. Based on the RMSE, which model is the best for fitting the given data?

```
model1 = LinearRegression()
model2 = LinearRegression()
model3 = LinearRegression()
loocv = LeaveOneOut()
ypred1 = cross_val_predict(model1, features_x, labels_y.flatten(),
cv=loocv)
ypred2 = cross_val_predict(model2, features_x**2, labels_y.flatten(),
cv=loocv)
ypred3 = cross_val_predict(model3, features_x**3, labels_y.flatten(),
cv=loocv)
print('Normal LR RMSE -
{:.4f}'.format(math.sqrt(mean_squared_error(labels_y.flatten(), ypred1))))
print('Quadartic LR RMSE -
{:.4f}'.format(math.sqrt(mean_squared_error(labels_y.flatten(), ypred2))))
print('Cubic LR RMSE -
{:.4f}'.format(math.sqrt(mean_squared_error(labels_y.flatten(), ypred3))))
```

Output -

```
Normal LR RMSE - 4.4927
Quadartic LR RMSE - 6.4587
Cubic LR RMSE - 8.0864
```

Based on the RMSE, simple linear regression is the best for fitting the given data.

Q4 - Aritificial Neural Networks

Train, validate, and test a neural network model using the dataset in hw3q4.zip, which contains training data (75%), validation data (12.5%), and test data (12.5%). There are two output classes in this data set. You can either choose matlab or a python neural networks package, Keras for this problem.

Import Libraries

```
import matplotlib.pyplot as plt
import numpy as np
import zipfile
import math
import sys
import os
np.random.seed(7)
from tensorflow.keras.layers import Activation
from tensorflow.keras.layers import Input
from tensorflow.keras.layers import Dense
from tensorflow.keras import Model
import tensorflow as tf
tf.set_random_seed(7)
%matplotlib inline
```

(a) Please briefly describe how to construct your working environments (e.g. language, package version, backend for neural networks, installation, etc.) in your report, and write how to execute your codes on 'readme' file.

Language

- python 3.7

Prerequisites

- tensorflow 1.8+ (CPU only)

- matplotlib
- numpy

Installation

Install the above packages using the following commands -

- tensorflow 1.8+ (CPU only) - `pip install tensorflow`
- matplotlib - `pip install matplotlib`
- numpy - `pip install numpy`

Data

hw3q4.zip should be in the same directory level as this notebook.

(b) Keras Model

Helper Functions

```
def get_data(zipname,filename):
    zf = zipfile.ZipFile(zipname)
    data = None
    with zf.open(filename, 'r') as fp:
        data = np.asarray([[float(v.strip()) for v in
x.decode().strip().split(',')] for x in fp.readlines()])
    return data
```

```
def ANN(n_neurons, output_dims):
    input_layer = Input(batch_shape = (None, 61), name='input_layer')
    layer = Dense(n_neurons, name='hidden_layer')(input_layer)
    layer = Activation(activation='relu', name='relu')(layer)
    layer = Dense(output_dims, name='output_layer')(layer)
    output_layer = Activation(activation='sigmoid', name='sigmoid')(layer)
    model = Model(inputs=input_layer, outputs=output_layer, name='ann')
    model.compile('adam', 'mse', ['accuracy'])
    return model
```

Read Data

```
train_x = get_data('Data' + os.sep + 'hw3q4.zip','hw3q4' + os.sep +
'X_train.csv')
train_y = get_data('Data' + os.sep + 'hw3q4.zip','hw3q4' + os.sep +
'Y_train.csv')
val_x = get_data('Data' + os.sep + 'hw3q4.zip', 'hw3q4' + os.sep +
'X_val.csv')
val_y = get_data('Data' + os.sep + 'hw3q4.zip', 'hw3q4' + os.sep +
'Y_val.csv')
test_x = get_data('Data' + os.sep + 'hw3q4.zip', 'hw3q4' + os.sep +
'X_test.csv')
test_y = get_data('Data' + os.sep + 'hw3q4.zip', 'hw3q4' + os.sep +
'Y_test.csv')
```

(1) Construct neural networks using the given training dataset (X train, Y train) using different number of hidden neurons. Set the parameters as follows: activation function for hidden layer='relu', activation for output layer ='sigmoid', loss function ='mse', metrics= 'accuracy', epochs=10, batch size=50. For each model, change the number of hidden neurons in the order of 2, 4, 6, 8, 10.

```
hidden_neurons = [2,4,6,8,10]
models = [ANN(hidden_neurons[i], train_y.shape[1]) for i in
range(len(hidden_neurons))]
histories = []
for i in range(len(models)):
    print('Training Model with {} neurons in hidden
layer'.format(hidden_neurons[i]))
    history = models[i].fit(x=train_x,
                  y=train_y,
                  batch_size=50,
                  epochs=10,
                  shuffle=True)
    histories.append(history)
    print()
```

```
Training Model with 2 neurons in hidden layer
Epoch 1/10
1500/1500 [==============================] - 0s 273us/step - loss: 0.2538
- acc: 0.5333
Epoch 2/10
```

```
1500/1500 [==============================] - 0s 25us/step - loss: 0.2481
- acc: 0.5533
Epoch 3/10
1500/1500 [==============================] - 0s 24us/step - loss: 0.2432
- acc: 0.5727
Epoch 4/10
1500/1500 [==============================] - 0s 25us/step - loss: 0.2387
- acc: 0.5907
Epoch 5/10
1500/1500 [==============================] - 0s 24us/step - loss: 0.2338
- acc: 0.6087
Epoch 6/10
1500/1500 [==============================] - 0s 25us/step - loss: 0.2285
- acc: 0.6193
Epoch 7/10
1500/1500 [==============================] - 0s 22us/step - loss: 0.2229
- acc: 0.6307
Epoch 8/10
1500/1500 [==============================] - 0s 22us/step - loss: 0.2164
- acc: 0.6347
Epoch 9/10
1500/1500 [==============================] - 0s 22us/step - loss: 0.2095
- acc: 0.6480
Epoch 10/10
1500/1500 [==============================] - 0s 25us/step - loss: 0.2022
- acc: 0.6733

Training Model with 4 neurons in hidden layer
Epoch 1/10
1500/1500 [==============================] - 0s 195us/step - loss: 0.3337
- acc: 0.4927
Epoch 2/10
1500/1500 [==============================] - 0s 25us/step - loss: 0.3077
- acc: 0.5280
Epoch 3/10
1500/1500 [==============================] - 0s 24us/step - loss: 0.2827
- acc: 0.5680
Epoch 4/10
1500/1500 [==============================] - 0s 25us/step - loss: 0.2596
- acc: 0.5967
Epoch 5/10
1500/1500 [==============================] - 0s 37us/step - loss: 0.2387
- acc: 0.6307
Epoch 6/10
1500/1500 [==============================] - 0s 27us/step - loss: 0.2213
- acc: 0.6547
```

```
Epoch 7/10
1500/1500 [==============================] - 0s 24us/step - loss: 0.2066
- acc: 0.6827
Epoch 8/10
1500/1500 [==============================] - 0s 23us/step - loss: 0.1944
- acc: 0.7133
Epoch 9/10
1500/1500 [==============================] - 0s 26us/step - loss: 0.1840
- acc: 0.7360
Epoch 10/10
1500/1500 [==============================] - 0s 24us/step - loss: 0.1753
- acc: 0.7580

Training Model with 6 neurons in hidden layer
Epoch 1/10
1500/1500 [==============================] - 0s 209us/step - loss: 0.2943
- acc: 0.4933
Epoch 2/10
1500/1500 [==============================] - 0s 25us/step - loss: 0.2667
- acc: 0.5333
Epoch 3/10
1500/1500 [==============================] - 0s 26us/step - loss: 0.2441
- acc: 0.5673
Epoch 4/10
1500/1500 [==============================] - 0s 25us/step - loss: 0.2260
- acc: 0.6000
Epoch 5/10
1500/1500 [==============================] - 0s 25us/step - loss: 0.2110
- acc: 0.6453
Epoch 6/10
1500/1500 [==============================] - 0s 27us/step - loss: 0.1988
- acc: 0.6727
Epoch 7/10
1500/1500 [==============================] - 0s 26us/step - loss: 0.1883
- acc: 0.7087
Epoch 8/10
1500/1500 [==============================] - 0s 27us/step - loss: 0.1791
- acc: 0.7387
Epoch 9/10
1500/1500 [==============================] - 0s 31us/step - loss: 0.1711
- acc: 0.7573
Epoch 10/10
1500/1500 [==============================] - 0s 23us/step - loss: 0.1643
- acc: 0.7760

Training Model with 8 neurons in hidden layer
```

```
Epoch 1/10
1500/1500 [==============================] - 0s 214us/step - loss: 0.2895 - acc: 0.5273
Epoch 2/10
1500/1500 [==============================] - 0s 25us/step - loss: 0.2614 - acc: 0.5653
Epoch 3/10
1500/1500 [==============================] - 0s 26us/step - loss: 0.2389 - acc: 0.6087
Epoch 4/10
1500/1500 [==============================] - 0s 27us/step - loss: 0.2202 - acc: 0.6460
Epoch 5/10
1500/1500 [==============================] - 0s 27us/step - loss: 0.2051 - acc: 0.6740
Epoch 6/10
1500/1500 [==============================] - 0s 29us/step - loss: 0.1924 - acc: 0.7073
Epoch 7/10
1500/1500 [==============================] - 0s 28us/step - loss: 0.1815 - acc: 0.7347
Epoch 8/10
1500/1500 [==============================] - 0s 25us/step - loss: 0.1725 - acc: 0.7533
Epoch 9/10
1500/1500 [==============================] - 0s 25us/step - loss: 0.1647 - acc: 0.7680
Epoch 10/10
1500/1500 [==============================] - 0s 23us/step - loss: 0.1580 - acc: 0.7747

Training Model with 10 neurons in hidden layer
Epoch 1/10
1500/1500 [==============================] - 0s 268us/step - loss: 0.2728 - acc: 0.5547
Epoch 2/10
1500/1500 [==============================] - 0s 28us/step - loss: 0.2387 - acc: 0.6080
Epoch 3/10
1500/1500 [==============================] - 0s 26us/step - loss: 0.2110 - acc: 0.6700
Epoch 4/10
1500/1500 [==============================] - 0s 25us/step - loss: 0.1890 - acc: 0.7140
Epoch 5/10
```

```
1500/1500 [==============================] - 0s 29us/step - loss: 0.1722
- acc: 0.7473
Epoch 6/10
1500/1500 [==============================] - 0s 22us/step - loss: 0.1596
- acc: 0.7687
Epoch 7/10
1500/1500 [==============================] - 0s 29us/step - loss: 0.1499
- acc: 0.7880
Epoch 8/10
1500/1500 [==============================] - 0s 22us/step - loss: 0.1425
- acc: 0.8053
Epoch 9/10
1500/1500 [==============================] - 0s 28us/step - loss: 0.1364
- acc: 0.8220
Epoch 10/10
1500/1500 [==============================] - 0s 26us/step - loss: 0.1315
- acc: 0.8240
```

(2) Validate each neural network using the given validation dataset (X val, Y val). The validation accuracy is used to determine how many number of hidden neurons are optimal for this problem.

```
eval_results = []
for i in range(len(models)):
    print('Evaluating Model with {} neurons in hidden
layer'.format(hidden_neurons[i]))
    eval_result = models[i].evaluate(x=val_x, y=val_y, batch_size=50)
    print('Mean Squared Error - {:.4f}'.format(eval_result[0]))
    print('Accuracy - {:.4f}'.format(eval_result[1]))
    eval_results.append(eval_result)
    print()
```

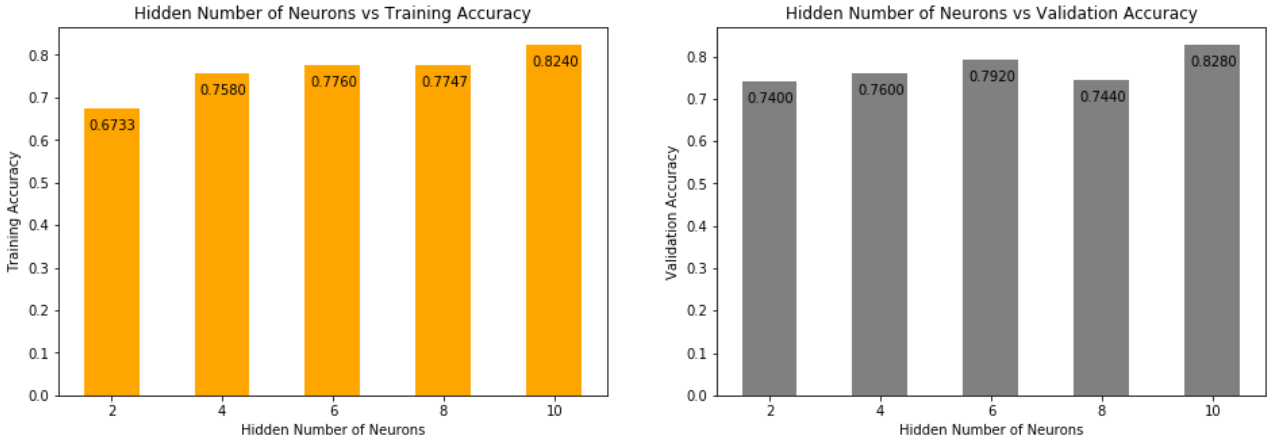
```
Evaluating Model with 2 neurons in hidden layer
250/250 [==============================] - 0s 360us/step
Mean Squared Error - 0.1976
Accuracy - 0.7400

Evaluating Model with 4 neurons in hidden layer
250/250 [==============================] - 0s 346us/step
Mean Squared Error - 0.1650
```

```
Accuracy - 0.7600

Evaluating Model with 6 neurons in hidden layer
250/250 [==============================] - 0s 372us/step
Mean Squared Error - 0.1585
Accuracy - 0.7920

Evaluating Model with 8 neurons in hidden layer
250/250 [==============================] - 0s 341us/step
Mean Squared Error - 0.1595
Accuracy - 0.7440

Evaluating Model with 10 neurons in hidden layer
250/250 [==============================] - 0s 318us/step
Mean Squared Error - 0.1238
Accuracy - 0.8280
```

(c) Plot a figure, where the horizontal x-axis is the number of hidden neurons, and the vertical y-axis is the accuracy. Please plot both training and validation accuracy in your figure. (Note that the exact accuracy could be slightly different according to your working environments, however you can analyze the trend.

```
fig, ax = plt.subplots(nrows=1, ncols=2, figsize=(16,5))
ax[0].set_title('Hidden Number of Neurons vs Training Accuracy')
ax[0].set_ylabel('Training Accuracy')
ax[0].set_xlabel('Hidden Number of Neurons')
ax[0].bar(hidden_neurons,
          [histories[i].history['acc'][-1] for i in
range(len(hidden_neurons))],
          width=1,
          color='orange')
for i in range(len(hidden_neurons)):
    ax[0].annotate('{:.4f}'.format(histories[i].history['acc'][-1]),
                   (hidden_neurons[i]-0.4, histories[i].history['acc']
[-1]-0.05))

ax[1].set_title('Hidden Number of Neurons vs Validation Accuracy')
```

```
ax[1].set_ylabel('Validation Accuracy')
ax[1].set_xlabel('Hidden Number of Neurons')
ax[1].bar(hidden_neurons,
          [eval_results[i][1] for i in range(len(hidden_neurons))],
          width=1,
          color='gray')
for i in range(len(hidden_neurons)):
    ax[1].annotate('{:.4f}'.format(eval_results[i][1]),
                   (hidden_neurons[i]-0.4, eval_results[i][1]-0.05))
```



(d) Provide a simple analysis about your results and choose the optimal number of hidden neuron from the analysis.

```
model = models[4]
```

Based on the above plots, we can clearly see that we get the best validation accuracy when number of neurons in hidden layer is 10. Thus we select the model with 10 neurons in hidden layers.

(e) Report the test accuracy using the given test dataset (X test, Y test) on the neural network with the optimal number of hidden neurons.

```
print('Testing Model with 10 neurons in hidden layer')
eval_result = model.evaluate(x=test_x, y=test_y, batch_size=50)
print('Mean Squared Error - {:.4f}'.format(eval_result[0]))
print('Accuracy - {:.4f}'.format(eval_result[1]))
print()
```

```
Testing Model with 10 neurons in hidden layer
250/250 [==============================] - 0s 19us/step
Mean Squared Error - 0.1594
Accuracy - 0.7880
```

Test Accuracy is 0.7880 with 10 neurons in Hidden Layer.

Q5 SVM Theory

(A)

Weight Vector
Given Datapoints:

X_1	X_2	Y (Class Label)
1	4	1
3	2	1
5	4	2
5	6	2

(I)
Class 1 (label 1)
Positive Hyperplane : $WX_1 + W_0 = 1$

Class 2 (label -1)
Negative Hyperplane : $WX_1 + W_0 = -1$

Let's say Weight vector w = $[w_1\ w_2]$

As per given hint that, $w_1 = w_2$

Considering support vectors to be (3,2), (1,4) and (5,4)

For support vector (3,2) -> Class 1 (label 1),

$[w1\ w2]\begin{bmatrix}3\\2\end{bmatrix} + W_0 = 1$ -> $3\ w_1 + 2\ w_2 + W_0 = 1$ ..... (i)

For support vector (3,2) -> Class 1 (label 1),

$[w1\ w2]\begin{bmatrix}1\\4\end{bmatrix} + W_0 = 1$ -> $w_1 + 4\ w_2 + W_0 = 1$ ..... (ii)

For support vector (5,4) -> Class 2 (label -1),

$[w1\ w2]\begin{bmatrix}5\\4\end{bmatrix} + W_0 = -1$ -> $5w_1 + 4\ w_2 + W_0 = -1$ ..... (iii)

(iii) - (ii)
$4\ w_1 = -2$

w_1 = - 0.5 ……. (iv)

Substituting value of w_1 into (i) and (ii)
- 3.5 + 2 w_2 + W_0 = 1 ……. (v)
- 0.5 + 4 w_2 + W_0 = 1 ……. (vi)

(vi) - (v)
1 + $2w_2$ = 0
w_2 = - 0.5 ……. (vii)

substituting values of w_1 and w_2 into (i)
3 (-0.5) + 2 (-0.5) + w_0 = 1

w_0 = 1 + 1.5 + 1
w_0 = 3.5

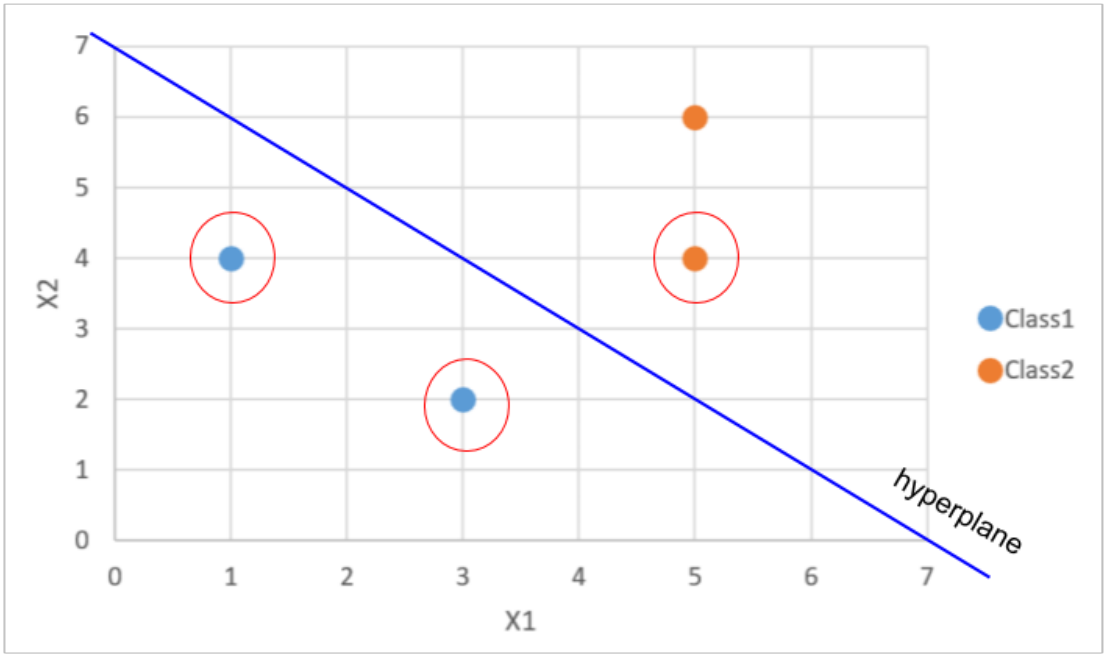
Therefore, weighted vector, W = [-0.5 -0.5]
Bias W_0 = 3.5

(ii) Support Vectors and Decision Boundary

Decision Boundary : $WX_1 + W_0 = 0$

$$[-0.5 \ -0.5]\begin{bmatrix}X1\\X2\end{bmatrix} + W_0 = 0$$

- $0.5X_1$ - $0.5X_2$ + 3.5 = 0
X_2 = $-X_1$ + 7



Q5

(B)

Data ID	x_1	x_2	y
X^1	0	2	-1
X^2	2	0	-1
X^3	0	0	1
X^4	2	2	1

(i) Kernel Function

$K(X^i, X^j) = (1 + X^i . X^j)^2$...(1)

We know $X^i . X^j = [X_1^i, X_2^i] . [X_1^j, X_2^j] = X_1^i . X_1^j + X_2^i . X_2^j$

$(1 + X^i . X^j)^2$

$= 1 + (X^i . X^j)^2 + 2.(X^i . X^j)$

$= 1 + (X_1^i . X_1^j + X_2^i . X_2^j)^2 + 2.(X_1^i . X_1^j + X_2^i . X_2^j)$

$= 1 + (X_1^i . X_1^j)^2 + (X_2^i . X_2^j)^2 + 2.(X_1^i . X_1^j . X_2^i . X_2^j) + 2.(X_1^i . X_1^j) + 2.(X_2^i . X_2^j)$ ...(2)

We know $X^i . X^j = \phi(X^i) . \phi(X^j)$ ...(3)

From (1), (2) and (3),

$\phi(X^i) = 1 + (X_1^i)^2 + (X_2^i)^2 + \sqrt{2} . (X_1^i . X_2^i) + \sqrt{2} . (X_1^i) + \sqrt{2} . (X_2^i)$

$\phi(X^j) = 1 + (X_1^j)^2 + (X_2^j)^2 + \sqrt{2} . (X_1^j . X_2^j) + \sqrt{2} . (X_1^j) + \sqrt{2} . (X_2^j)$

$\phi(v) =< 1, v_1^2, v_2^2, \sqrt{2}v_1v_2, \sqrt{2}v_1, \sqrt{2}v_2 >$

(ii) Transformed Space

Data ID	**1**	x_1^2	x_2^2	$\sqrt{2}x_1x_2$	$\sqrt{2}x_1$	$\sqrt{2}x_2$	y
X^1	1	0	4	0	0	$2\sqrt{2}$	-1
X^2	1	4	0	0	$2\sqrt{2}$	0	-1
X^3	1	0	0	0	0	0	1
X^4	1	4	4	$4\sqrt{2}$	$2\sqrt{2}$	$2\sqrt{2}$	1

Q-5 (b)

(iii)

Data ID	1	2	3	4
(x_1, x_2)	(0, 2)	(2, 0)	(0, 0)	(2, 2)
label y	−1	−1	1	1

→ Dot products of data-points:

Data ID	1	2	3	4
1	4	0	0	4
2	0	4	0	4
3	0	0	0	0
4	4	4	0	8

→ Kernel values $K(x_i, x_j) = (1 + x_i \cdot x_j)^2$

	1	2	3	4
1	25	1	1	25
2	1	25	1	25
3	1	1	1	1
4	25	25	1	81

$$L = \sum \alpha_i - \frac{1}{2} \sum_i \sum_j \alpha_i \alpha_j x_i x_j y_i y_j$$

$$= \sum \alpha_i - \frac{1}{2} \sum_i \sum_j \alpha_i \alpha_j \cdot k(x_i, x_j)\, y_i y_j$$

$$= \alpha_1 + \alpha_2 + \alpha_3 + \alpha_4$$

$$- \frac{1}{2} \Big[\alpha_1^2 \cdot 25 \cdot 1 + \alpha_1 \cdot \alpha_2 - \alpha_1 \alpha_3 - 25 \alpha_1 \alpha_4$$

$$+ \alpha_1 \cdot \alpha_2 + 25 \alpha_2^2 - \alpha_2 \cdot \alpha_3 - 25 \alpha_2 \alpha_4$$

$$- \alpha_3 \alpha_1 - \alpha_2 \alpha_3 + \alpha_3^2 + \alpha_3 \alpha_4$$

$$- 25 \alpha_1 \alpha_4 - 25 \alpha_2 \alpha_4 + \alpha_3 \alpha_4 + 81 \alpha_4^2 \Big]$$

$$\therefore L = \alpha_1 + \alpha_2 + \alpha_3 + \alpha_4$$

$$- \frac{25}{2} \alpha_1^2 - \frac{25}{2} \alpha_2^2 - \frac{\alpha_3^2}{2} - \frac{81}{2} \alpha_4^2$$

$$- \alpha_1 \alpha_2 + \alpha_1 \alpha_3 + 25 \alpha_1 \alpha_4 + \alpha_2 \alpha_3$$

$$+ 25 \alpha_2 \alpha_4 - \alpha_3 \alpha_4$$

$$\therefore \frac{dL}{d\alpha_1} = 1 - 25\alpha_1 - \alpha_2 + \alpha_3 + 25\alpha_4 = 0 \quad - (1)$$

$$\therefore \frac{dL}{d\alpha_2} = 1 - 25\alpha_2 - \alpha_1 + \alpha_3 + 25\alpha_4 = 0 \quad - (2)$$

$$\therefore \frac{dL}{d\alpha_3} = 1 - \alpha_3 + \alpha_1 + \alpha_2 - \alpha_4 = 0 \quad - (3)$$

$$\therefore \frac{dL}{d\alpha_4} = 1 - 81\alpha_4 + 25\alpha_1 + 25\alpha_2 - \alpha_3 = 0 \quad - (4)$$

(1) − (2)

$$-24\alpha_1 + 24\alpha_2 = 0$$

$$\therefore \boxed{\alpha_1 = \alpha_2} \quad - (5)$$

(2) + (4)

$$2 + 24\alpha_1 - 56\alpha_4 = 0$$

$$1 + 12\alpha_1 - 28\alpha_4 = 0$$

$$\therefore \boxed{12\alpha_1 = 28\alpha_4 - 1} \quad - (6)$$

(2) + (3)

$$2 - 24\alpha_2 + 4\alpha_4 = 0$$

$$1 - 12\alpha_2 + 12\alpha_4 = 0$$

$$\therefore \boxed{12\alpha_4 = 12\alpha_2 - 1} \quad - (7)$$

(6), (7) and (5)

$$12\alpha_1 = 28\alpha_4 - 1 \quad - (6)$$

$$12\alpha_4 = 12\alpha_1 - 1 \quad - (7)$$

Substituting value of (6) into (7)

$$12\alpha_4 = (28\alpha_4 - 1) - 1$$

$$\therefore 16\alpha_4 = 2$$

$$\therefore \boxed{\alpha_4 = \frac{1}{8}} \quad - (8)$$

Substituing value of α_4 in (7).

$$12\left(\frac{1}{8}\right) = 12\alpha_1 - 1$$

$$\therefore \frac{3}{2} = 12\alpha_1 - 1$$

$$\therefore 12\alpha_1 = \frac{3}{2} + 1 = 5/2$$

$$\therefore \boxed{\alpha_1 = \frac{5}{24} = \alpha_2} \quad - (9)$$

Substituiting values $\alpha_1, \alpha_2, \alpha_4$ in (3)

$$1 + \frac{5}{24} + \frac{5}{24} - \frac{1}{8} = \alpha_3$$

$$\therefore \alpha_3 = \frac{24 + 10 - 3}{24} = \frac{31}{24}$$

$$\therefore \boxed{\alpha_3 = \frac{31}{24}}$$

For x' datapoint, (support vector).

$$w \cdot x_1 + w_0 = y_1$$

$$w \cdot \phi(x_1) + w_0 = y_1$$

$$\therefore \sum_{i=1 \text{ to } 4} \alpha_i y_i \phi(x_i) \cdot \phi(x_1) + w_0 = y_1$$

$$\therefore \alpha_1 y_1 k(x_1, x_1) + \alpha_2 y_2 k(x_2, x_1) + \alpha_3 y_3 k(x_3, x_1)$$

$$+ \alpha_4 y_4 k(x_4, x_1) + w_0 = y_1$$

$$\therefore \left(\frac{5}{24}\right)(-1)(25) + \frac{5}{24}(-1)(1) + \frac{31}{24}(1)(1) + \frac{1}{8}(1)(25)$$

$$+ w_0 = -1$$

$$\therefore \frac{-125 - 5 + 31 + 75}{24} + w_0 = -1$$

$$\therefore \frac{-24}{24} + w_0 = -1$$

$$\therefore \boxed{w_0 = 0}$$

Decision Boundary

$W.X + W_0 = 0$

$\sum_{i=1}^{4} \alpha_i . y_i . \phi(X^i) . \phi(X) + W_0 = 0$

$\alpha_1 . y_1 . \phi(< 1, 0, 4, 0, 0, 2\sqrt{2} >) . \phi(< 1, x_1^2, x_2^2, \sqrt{2}. x_1 . x_2, \sqrt{2}. x_1, \sqrt{2}. x_2 >)$
$+\alpha_2 . y_2 . \phi(< 1, 4, 0, 0, 2\sqrt{2}, 0 >) . \phi(< 1, x_1^2, x_2^2, \sqrt{2}. x_1 . x_2, \sqrt{2}. x_1, \sqrt{2}. x_2 >)$
$+\alpha_3 . y_3 . \phi(< 1, 0, 0, 0, 0, 0 >) . \phi(< 1, x_1^2, x_2^2, \sqrt{2}. x_1 . x_2, \sqrt{2}. x_1, \sqrt{2}. x_2 >)$
$+\alpha_4 . y_4 . \phi(< 1, 4, 4, 4\sqrt{2}, 2\sqrt{2}, 2\sqrt{2} >) . \phi(< 1, x_1^2, x_2^2, \sqrt{2}. x_1 . x_2, \sqrt{2}. x_1, \sqrt{2}. x_2 >) + W_0 = 0$

$1 - \frac{1}{3}x_1^2 - \frac{1}{3}x_2^2 + 8.x_1 . x_2 + 8.x_1 + 8.x_2 = 0$

$x_1^2 + x_2^2 + 24.x_1 . x_2 + 24.x_1 + 24.x_2 - 3 = 0$ is the decision boundary.

Q6 - SVM Programming

In this question, you will employ SVM to solve a classification problem for the provided data file "hw3q6.csv". Each row in the data file indicates a sample. The first 12 columns are features and the last column "Class" indicates the label, with 1 and 0 indicating the positive and negative samples, respectively.

Import Libraries

```
from sklearn.metrics import accuracy_score, f1_score, precision_score, recall_score
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.svm import SVC

import matplotlib.pyplot as plt
import numpy as np
import sys
import os

%matplotlib inline
```

(a) Load data. Report the size of positive and negative samples in dataset

```
def data_and_headers(filename):
    data = None
    with open(filename) as fp:
        data = [x.strip().split(',') for x in fp.readlines()]
    headers = data[0]
    headers = np.asarray(headers)
    class_field = len(headers) - 1
    data_x = [[float(x[i]) for i in range(class_field)] for x in data[1:]]
    data_x = np.asarray(data_x)
    data_y = [[int(x[i]) for i in range(class_field, class_field + 1)] for x in data[1:]]
    data_y = np.asarray(data_y)
    return headers, data_x, data_y
```

```
headers, features_x, labels_y = data_and_headers('Data' + os.sep +
'hw3q6.csv')
```

```
print('Data')
print('Number of features - ' + str(features_x.shape[1]))
print('Total Number of observations - ' + str(features_x.shape[0]))
print('Number of Positive Samples - ' + str(labels_y[labels_y==1].shape[0]))
print('Number of Negative Samples - ' + str(labels_y[labels_y==0].shape[0]))
print()
```

```
Data
Number of features - 12
Total Number of observations - 200
Number of Positive Samples - 90
Number of Negative Samples - 110
```

(b) Use stratified random sampling to divide the dataset into training data (75%) and testing data (25%). Report the number of positive and negative samples in both training and testing data.

```
train_x, test_x, train_y, test_y = train_test_split(features_x,
                                                    labels_y,
                                                    test_size=0.25,
                                                    random_state=10,
                                                    shuffle=True,
                                                    stratify=labels_y)
```

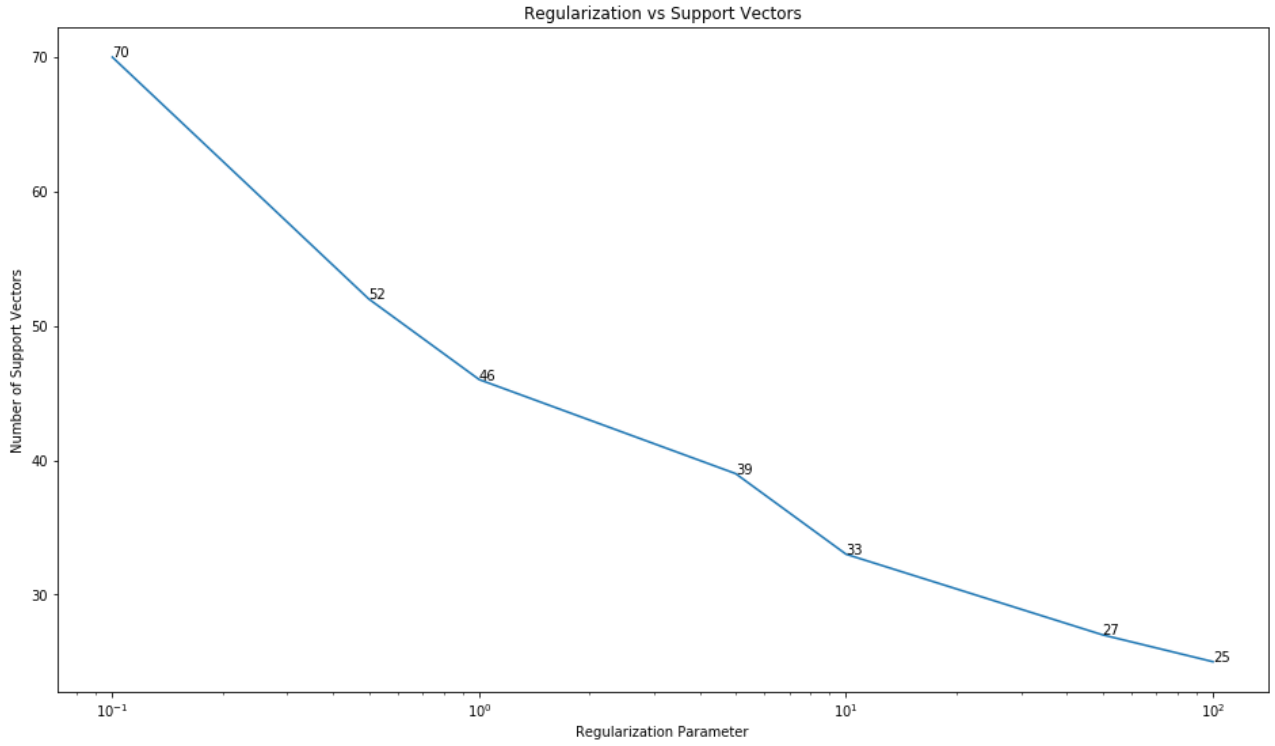
```
print('Training Data')
print('Number of features - ' + str(train_x.shape[1]))
print('Total Number of observations - ' + str(train_x.shape[0]))
print('Number of Positive Samples - ' + str(train_y[train_y==1].shape[0]))
print('Number of Negative Samples - ' + str(train_y[train_y==0].shape[0]))
print()
print('Testing Data')
print('Number of features - ' + str(test_x.shape[1]))
print('Total Number of observations - ' + str(test_x.shape[0]))
print('Number of Positive Samples - ' + str(test_y[test_y==1].shape[0]))
print('Number of Negative Samples - ' + str(test_y[test_y==0].shape[0]))
print()
```

```
Training Data
Number of features - 12
Total Number of observations - 150
Number of Positive Samples - 67
Number of Negative Samples - 83

Testing Data
Number of features - 12
Total Number of observations - 50
Number of Positive Samples - 23
Number of Negative Samples - 27
```

(c) Take SVM with linear kernel as classifier (third-party packages are allowed to use) and set the regularization parameter C as: [0.1, 0.5, 1, 5, 10, 50, 100], respectively. For each value of C, train a SVM classifier with the training data and get the number of support vectors (SVs). Generate a plot with C as the horizontal axis and number of SVs as the vertical axis. Give a brief analysis for the plot.

```
C = [0.1, 0.5, 1, 5, 10, 50, 100]
models = [SVC(C[i],'linear',random_state=10).fit(train_x,train_y.flatten())
for i in range(len(C))]
```

```
fig, ax = plt.subplots(figsize=(16,9))
ax.set_title('Regularization vs Support Vectors')
ax.set_ylabel('Number of Support Vectors')
ax.set_xlabel('Regularization Parameter')
ax.set_xscale('log')
plty = [x.support_.shape[0] for x in models]
#ax.bar(C, plty, color='orange')
ax.plot(C, plty)
for i, txt in enumerate(C):
    ax.annotate(plty[i], (C[i], plty[i]))
plt.show()
```



(d) Compare 4 different kernel functions, including linear, polynomial, radial basic function (Gaussian kernel), and sigmoid kernel. Make a table to record the accuracy, precision, recall and f-measure of the classification results for the 4 kernel functions. Try to tune the parameters via grid search and report your best results with the optimal parameters. Based on the results, which kernel function will you choose?

Linear Kernel

```
param_grid = [
    {'C':[0.001, 0.005, 0.01, 0.05, 0.1, 0.5, 1, 5, 10, 50, 100],
     'kernel':['linear']}
]

lsvc = SVC(random_state=10)
linear_clf = GridSearchCV(lsvc, param_grid,
                  ['f1', 'accuracy', 'recall', 'precision'],
                  cv=5, refit='accuracy', verbose=1, n_jobs=4)
linear_clf.fit(train_x, train_y.flatten())
y_pred = linear_clf.predict(test_x)
```

```
Fitting 5 folds for each of 11 candidates, totalling 55 fits
[Parallel(n_jobs=4)]: Using backend LokyBackend with 4 concurrent workers.
[Parallel(n_jobs=4)]: Done  55 out of  55 | elapsed:    0.3s finished
```

```
print('Best Parameters - ')
print(linear_clf.best_params_)
print()
print('Linear Kernel Classification Results on Test Set with Optimal Parameters')
print('Accuracy - \t{:.4f}'.format(accuracy_score(test_y.flatten(), y_pred)))
print('F1 Score - \t{:.4f}'.format(f1_score(test_y.flatten(), y_pred)))
print('Precision - \t{:.4f}'.format(precision_score(test_y.flatten(), y_pred)))
print('Recall - \t{:.4f}'.format(recall_score(test_y.flatten(), y_pred)))
```

```
Best Parameters -
{'C': 10, 'kernel': 'linear'}

Linear Kernel Classification Results on Test Set with Optimal Parameters
Accuracy -  0.9600
F1 Score -  0.9565
Precision -     0.9565
Recall -    0.9565
```

Radial Basis Function

```
param_grid = [
    {'C':[0.001, 0.005, 0.01, 0.05, 0.1, 0.5, 1, 5, 10, 50, 100],
     'kernel':['rbf'],
     'gamma': [0.1,0.2,0.3,0.4,0.5,0.6,0.7,0.8,0.9,1.0]}
]

rsvc = SVC(random_state=10)
radial_clf = GridSearchCV(rsvc, param_grid,
                  ['f1', 'accuracy', 'recall', 'precision'],
                  cv=5, refit='accuracy', verbose=1, n_jobs=4)
radial_clf.fit(train_x, train_y.flatten())
y_pred = radial_clf.predict(test_x)
```

```
Fitting 5 folds for each of 110 candidates, totalling 550 fits
[Parallel(n_jobs=4)]: Using backend LokyBackend with 4 concurrent workers.
[Parallel(n_jobs=4)]: Done 550 out of 550 | elapsed:    1.4s finished
```

```
print('Best Parameters - ')
print(radial_clf.best_params_)
print()
print('Radial Basis Function Kernel Classification Results on Test Set with Optimal Parameters')
print('Accuracy - \t{:.4f}'.format(accuracy_score(test_y.flatten(), y_pred)))
print('F1 Score - \t{:.4f}'.format(f1_score(test_y.flatten(), y_pred)))
print('Precision - \t{:.4f}'.format(precision_score(test_y.flatten(), y_pred)))
print('Recall - \t{:.4f}'.format(recall_score(test_y.flatten(), y_pred)))
```

```
Best Parameters -
{'C': 1, 'gamma': 0.2, 'kernel': 'rbf'}

Radial Basis Function Kernel Classification Results on Test Set with Optimal Parameters
Accuracy -  0.9800
F1 Score -  0.9778
Precision -     1.0000
Recall -    0.9565
```

Polynomial Kernel

```
param_grid = [
    {'C':[0.001, 0.005, 0.01, 0.05, 0.1, 0.5, 1, 5, 10, 50, 100],
     'kernel':['poly'],
     'degree':[2,3,4,5,6],
     'gamma': [0.1,0.2,0.3,0.4,0.5,0.6,0.7,0.8,0.9,1.0],
     'coef0': [0.1,0.5,1.0,1.5,2.0,5.0,10.0,20.0]}
]

psvc = SVC(random_state=10)
poly_clf = GridSearchCV(psvc, param_grid,
                  ['f1', 'accuracy', 'recall', 'precision'],
                  cv=5, refit='accuracy', verbose=1, n_jobs=4)
poly_clf.fit(train_x, train_y.flatten())
y_pred = poly_clf.predict(test_x)
```

```
Fitting 5 folds for each of 4400 candidates, totalling 22000 fits
[Parallel(n_jobs=4)]: Using backend LokyBackend with 4 concurrent workers.
[Parallel(n_jobs=4)]: Done 756 tasks      | elapsed:    2.2s
[Parallel(n_jobs=4)]: Done 4056 tasks      | elapsed:    9.3s
[Parallel(n_jobs=4)]: Done 9556 tasks      | elapsed:   20.5s
[Parallel(n_jobs=4)]: Done 17256 tasks      | elapsed:   34.6s
[Parallel(n_jobs=4)]: Done 22000 out of 22000 | elapsed:   43.7s finished
```

```
print('Best Parameters - ')
print(poly_clf.best_params_)
print()
print('Polynomial Kernel Classification Results on Test Set with Optimal Parameters')
print('Accuracy - \t{:.4f}'.format(accuracy_score(test_y.flatten(), y_pred)))
print('F1 Score - \t{:.4f}'.format(f1_score(test_y.flatten(), y_pred)))
print('Precision - \t{:.4f}'.format(precision_score(test_y.flatten(), y_pred)))
print('Recall - \t{:.4f}'.format(recall_score(test_y.flatten(), y_pred)))
```

```
Best Parameters -
{'C': 0.001, 'coef0': 5.0, 'degree': 4, 'gamma': 0.8, 'kernel': 'poly'}

Polynomial Kernel Classification Results on Test Set with Optimal Parameters
Accuracy -  0.9600
F1 Score -  0.9565
Precision -     0.9565
Recall -    0.9565
```

Sigmoid Kernel

```
param_grid = [
    {'C':[0.001, 0.005, 0.01, 0.05, 0.1, 0.5, 1, 5, 10, 50, 100],
     'kernel':['sigmoid'],
     'gamma': [0.1,0.2,0.3,0.4,0.5,0.6,0.7,0.8,0.9,1.0],
     'coef0': [0.1,0.5,1.0,1.5,2.0,5.0,10.0,20.0]}
]

ssvc = SVC(random_state=10)
sig_clf = GridSearchCV(ssvc, param_grid,
                   ['f1', 'accuracy', 'recall', 'precision'],
                   cv=5, refit='accuracy', verbose=1, n_jobs=4)
sig_clf.fit(train_x, train_y.flatten())
y_pred = sig_clf.predict(test_x)
```

```
Fitting 5 folds for each of 880 candidates, totalling 4400 fits
[Parallel(n_jobs=4)]: Using backend LokyBackend with 4 concurrent workers.
[Parallel(n_jobs=4)]: Done 416 tasks      | elapsed:    4.6s
[Parallel(n_jobs=4)]: Done 2216 tasks      | elapsed:   15.1s
[Parallel(n_jobs=4)]: Done 4400 out of 4400 | elapsed:   20.9s finished
```

```
print('Best Parameters - ')
print(sig_clf.best_params_)
print()
print('Sigmoid Kernel Classification Results on Test Set with Optimal Parameters')
print('Accuracy - \t{:.4f}'.format(accuracy_score(test_y.flatten(), y_pred)))
print('F1 Score - \t{:.4f}'.format(f1_score(test_y.flatten(), y_pred)))
print('Precision - \t{:.4f}'.format(precision_score(test_y.flatten(), y_pred)))
print('Recall - \t{:.4f}'.format(recall_score(test_y.flatten(), y_pred)))
```

```
Best Parameters -
{'C': 0.05, 'coef0': 0.1, 'gamma': 0.3, 'kernel': 'sigmoid'}

Sigmoid Kernel Classification Results on Test Set with Optimal Parameters
Accuracy -  0.9000
F1 Score -  0.9020
Precision -     0.8214
Recall -    1.0000
```

Results

Kernel with Tuned Hyperparameter	Accuracy	F1-score	Precision	Recall
Linear **{'C': 10}**	0.9600	0.9565	0.9565	0.9565
RBF **{'C': 1, 'gamma': 0.2}**	0.9800	0.9778	1.0000	0.9565
Polynomial **{'C': 0.001, 'coef0': 5.0, 'degree': 4, 'gamma': 0.8}**	0.9600	0.9565	0.9565	0.9565
Sigmoid **{'C': 0.05, 'coef0': 0.1, 'gamma': 0.3}**	0.9000	0.9020	0.8214	1.0000

Based on the above results, we will choose the RBF kernel, with C=1 and gamma=0.2 as the kernel function. This is because it is giving the best score across 3 categories - accuracy, f1 and precision.