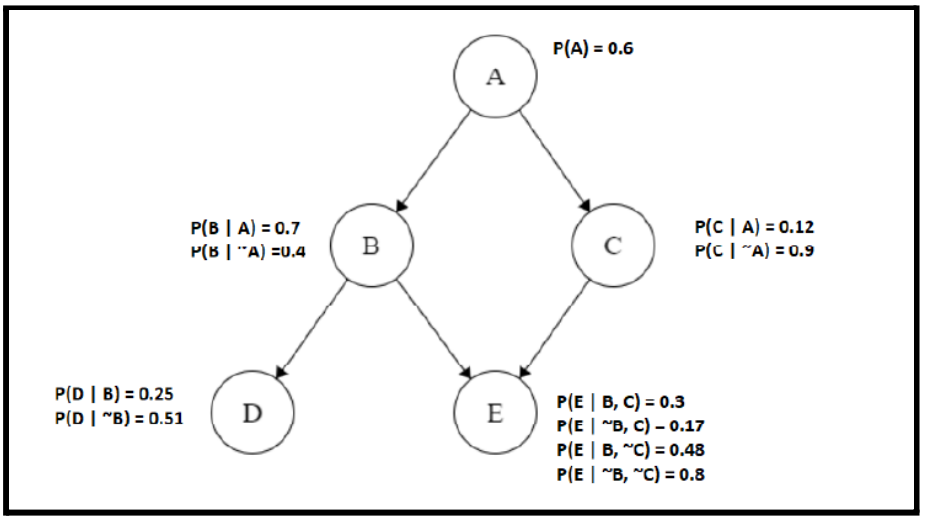
ALDA Fall 2021 – Homework 4

GITHUB Repository - https://github.ncsu.edu/sjanard/engr-ALDA-fall2021-H12

Homework Team 12 – HW12

Sudharsan Janardhanan – sjanard
Sriram Sudharsan – ssudhar
Pradyumna Vemuri – pvemuri

1. (15 points) [BN Inference] [John Wesley Hostetter (Designed) & Tyrone Wu (Graded)]
Compute the following probabilities according to the Bayesian net shown in Figure 1.



(a) (4 points) Compute P(A , ~C, D, ~E). Show your work.

P(A, ~C, D, E)
=P(A) * P(~C|A) * [P(D|B) * P(B) + P(D|~B) * P(~B)] * [P(~E|B, ~C) * P(B) + P(~E|~B, ~C) * P(~B)]

Given:
P(A) = 0.6 (Given)
P(C|A) = 0.12 (Given)
P(~C|A) = 1-P(C|A)
= 1 - 0.12
= 0.88
P(~C|A) = 0.88
P(~B) = 1 - P(B)
P(B|A) = 0.7 (Given)
P(B|~A) = 0.4 (Given)
P(A) = 0.6 (Given)
P(~A) = 1-0.6 = 0.4
Substituting the above values in P(B)
P(B) = P(B | A) * P(A) + P(B | ~A) * P(~A)
= (0.7) * (0.6) + (0.4) * (0.4)
= 0.42 + 0.16
= 0.58
P(B) = 0.58
P(~B) = 1 - 0.58 = 0.42

P(D | B) = 0.25 (Given)
P(D | ~B) = 0.51 (Given)
P(E|B,~C) = 0.48
P(~E|B,~C) = 1 - P(E|B,~C)
= 1 - 0.48
= 0.52

P(~E|B,~C) = 0.52
P(~E|~B,~C) = 1 - P(E|~B,~C)
= 1 - 0.8
= 0.2
P(~E|~B,~C) = 0.2 (Given)

P(A, ~C, D, E)
=P(A) * P(~C|A) * [(P(D|B)) * (P(B)) + (P(D|~B)) * (P(~B))] * [(P(~E|B, ~C)) * (P(B)) + (P(~E|~B, ~C)) * (P(~B))]
P(A,~C,D,~E) = (0.6)*(0.88)*([0.25*0.58+0.51*0.42])*([(0.52)*(0.58)+(0.2)*(0.42)])
= (0.6)*(0.88)*[(0.145)+(0.2142)]*[(0.3016)+(0.084)]
= 0.6*0.88*0.3592*0.3856
= 0.07313
P(A,~C,D,~E) = 0.0731

(b) (5 points) Compute P(A| B). Show your work.

We can infer P(A|B) + P(~A|B) = 1
//Substituting relevant formulae, we get:
(P(B|A)P(A))/P(B) + (P(B|~A)P(~A))/P(B) = 1

P(B|~A) = 0.4
P(B|A) = 0.7
P(~A) = 0.4
P(A) = 0.6

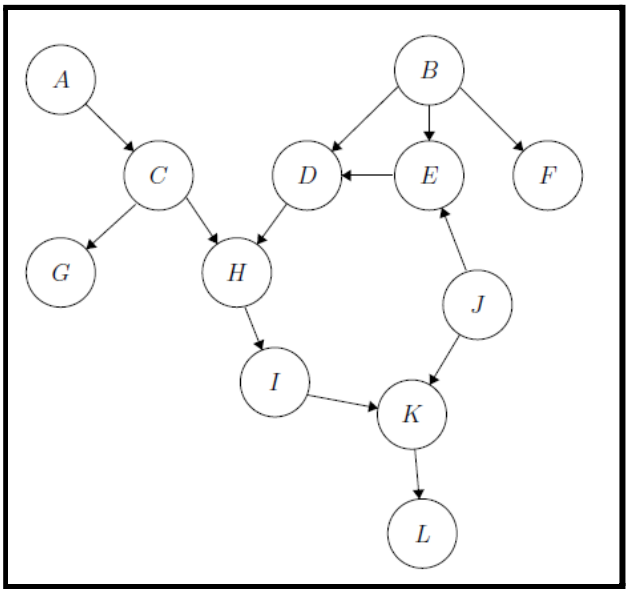
Substituting these values in the above equation,
(0.42 + 0.16) / P(B) = 1
P(B) = 0.58
P(A|B) = (P(B|A) . P(A)) / P(B) = (0.7*0.6)/0.58 = 0.72413
P(A|B) = 0.7241 (approximated)

(c) (6 points) Compute P(~B | D). Show your work.

P(B|D) + P(~B|D) = 1
P(D|B)=0.25 (Given)
P(B) = 0.58 (Given)
P(D|~B) = 0.51 (Given)
P(~B) = 1-0.58 = 0.42
[(P(D|B) * P(B)) / P(D)] + [(P(D|~B)*P(~B)) / P(D)] = 1

[(0.25*0.58) + (0.51*0.42)] / P(D) = 1
P(D) = 0.145 + 0.2142
= 0.3592
P(D)=0.3592
P(~B|D) = (P(D|~B) * P(~B)) / P(D)
P(D|~B) = 0.51 (Given)
P(D)=0.3592
P(~B) = 0.42
Substituting the values, we get:
= (0.51*0.42)/0.3592
= 0.59632
P(~B|D) = 0.59632

2. (20 points) [D-Separation] [Designed and Graded by Ge Gao] Conditional independence is a key concept in Bayesian Belief Network (BBN). Please answer the following conditional independence and d-separation questions using the graphs below.



(a) (5 points) In Figure 2, is A d-separated by ∅ from B? If so, how? Justify your answer.

X1 = B
X2= ∅
X3 = A

B->E->D->H<-C<-A
E: Type = Serial, Not in Evidence, Not blocked
D: Type = Serial, Not in Evidence, Not blocked
H: Type = Converging, Not in Evidence, Blocked

Path is blocked by H

B->D->H<-C<-A
D: Type = Serial, Not in Evidence, Not blocked
H: Type = Converging, Not in Evidence, Blocked

Path is blocked by H

B->E<-J->K<-I<-H<-C<-A

E: Type = Converging, Not in evidence, Blocked
Path is blocked by E

Therefore, A is d-seperated from B by ∅.
D-SEPERATED

(b) (5 points) In Figure 2, is {A;C} d-separated by H from K? If so, how? Justify your answer.

X1=K
X2=H
X3={A,C}

Consider {A,C} to be represented as W
K<-I<-H<-W
H: Type = Serial, In evidence, Blocked
Path is blocked by H.

K<-J->E->D->H<-W
H: Type = Converging, In evidence, Not blocked
D: Type = Serial, Not in evidence, Not blocked
E: Type = Diverging, Not in evidence, Not blocked
J: Type = Diverging, Not in evidence, Not blocked
Path is not blocked.

Path: K<-J->E<-B->D->H<-W
J: Type = Diverging, Not in evidence, Not blocked
E: Type = Converging, Not in evidence, Blocked
Path is blocked

We have a path that isn't blocked, K<-J->E->D->H<-W
Therefore, we conclude that {A,C} isn't d-separated by H from K.
NOT D-SEPERATED

(c) (5 points) In Figure 2, is D d-separated by {B; J} from E? If so, how? Justify your answer.

X1=D
X2={B,J}
X3=E

When path taken is **Path: D->H->I->K<-J->E**
H: Type = Serial, and is not in evidence, therefore not blocked.
I: Type = Serial, and is not in evidence, therefore not blocked.

K: Type = Converging and is not in evidence, therefore blocked.

Path is Blocked by K.

When path taken is **Path: D<-B->E**
B: type = Diverging and is in evidence, Therefore, blocked.

Path is blocked by B.

Path: D<-E
In this it is always active as they are connected by an edge. They are thereby dependent.

Due to the existence of one active edge, we reasonably conclude that they aren't d-separated.
NOT D-SEPERATED

(d) (5 points) In Figure 2, is {A;C;G} d-separated by {H;K} from {B;E; F}? If so, how? Justify your answer.

X1={B,E.F} represented as Q
X2={H,K}
X3={A,C,G} represented as T

Path: Q -> D ->H <- T
D: Type = Serial and is not in evidence, therefore not blocked.
H: Type = Converging and is in evidence, therefore not blocked.
Therefore, path is blocked.

Path: Q <-J->K<-I<-H<-T
J: Type = Diverging and is not in evidence. Therefore, not blocked.
K: Type = Converging and is in evidence. Therefore, not blocked.
I: Type = Serial, and is not in evidence. Therefore, not blocked.
H: Type = Serial, and is in evidence. Therefore, blocked.
Path is blocked by H.

We have a path that isn't blocked, therefore{A,C,G} isn't d-separated by {H,K} from {B,E,F}.
NOT D-SEPERATED

3. (20 points) [General Linear Regression] [Designed and Graded by Angela Zhang]

(a) (5 points) Given the following three data points of (x_1, x_2, y): (0, 3, -3), (1, 1, 4), (3, 1, 5), try to use a linear regression $y = \beta_1.x_1^2 + \beta_2 x_1 x_2 + \beta_0$ to predict y. Determine the values of β_1, β_2, and β_0 by manual calculation and show each step of your work.

Solution:

As per the given data points, the output for linear regression is:

X_1	X_2	Y	Y-hat
0	3	-3	β_0
1	1	4	$\beta_1 + \beta_2 + \beta_0$
3	1	5	$9\beta_1 + 3\beta_2 + \beta_0$

So the error would be calculated as:

$\mathbf{J} = (-3-\beta_0)^2 + (4 - (\beta_1 + \beta_2 + \beta_0))^2 + (5 - (9\beta_1 + 3\beta_2 + \beta_0))^2$

$= (3 + \beta_0)^2 + (4 - \beta_1 - \beta_2 - \beta_0)^2 + (5 - 9\beta_1 - 3\beta_2 - \beta_0))^2$

Derivative of J is calculated as

$\mathbf{dJ / d\beta_1} = 0 + 2(4 - \beta_1 - \beta_2 - \beta_0)(-1) + 2(5 - 9\beta_1 - 3\beta_2 - \beta_0)(-9)$

$= (-2)(4 - \beta_1 - \beta_2 - \beta_0) + (-18)(5 - 9\beta_1 - 3\beta_2 - \beta_0)$

$= -98 + 164\beta_1 + 56\beta_2 + 20\beta_0$

But $dJ / d\beta_1 = 0$, so we get an equation:

$\mathbf{164\beta_1 + 56\beta_2 + 20\beta_0 = 98}$ _____(1)

$\mathbf{dJ / d\beta_2} = 0 + 2(4 - \beta_1 - \beta_2 - \beta_0)(-1) + 2(5 - 9\beta_1 - 3\beta_2 - \beta_0)(-3)$

$= (-2)(4 - \beta_1 - \beta_2 - \beta_0) + (-6)(5 - 9\beta_1 - 3\beta_2 - \beta_0)$

$= -38 + 56\beta_1 + 20\beta_2 + 8\beta_0$

But $\partial J / \partial \beta_2 = 0$, so we get an equation:

$56\beta_1 + 20\beta_2 + 8\beta_0 = 38$ _____(2)

$dJ / d\beta_0$ $= 2(-3-\beta_0)(-1) + 2(4 - \beta_1 - \beta_2 - \beta_0)(-1) + 2(5 - 9\beta_1 - 3\beta_2 - \beta_0)(-1)$

$= 2(3+\beta_0) + (-2)(4 - \beta_1 - \beta_2 - \beta_0) + (-2)(5 - 9\beta_1 - 3\beta_2 - \beta_0)$

$= -12 + 20\beta_1 + 8\beta_2 + 6\beta_0$

But $\partial J / \partial \beta_0 = 0$, so we get an equation:

$20\beta_1 + 8\beta_2 + 6\beta_0 = 12$ _____(3)

Together we have 3 equations:

- $164\beta_1 + 56\beta_2 + 20\beta_0 = 98$
- $56\beta_1 + 20\beta_2 + 8\beta_0 = 38$
- $20\beta_1 + 8\beta_2 + 6\beta_0 = 12$

Solving these equations would give us the values of coefficients:

$\beta 1 = -13/6 =$ **-2.1667**

$\beta_2 = 55/6 =$ **9.1667**

$\beta_0 =$ **-3**

(b) (15 points) [Programming Task] Apply the following three linear regressions:

(1) $y = \alpha_1 x_1 + \alpha_2 x_2 + \alpha_3 x_3 + \alpha_0$
(2) $y = \beta_1 x_1^2 + \beta_2 x_2^2 + \beta_3 x_3^2 + \beta_0$
(3) $y = \gamma_1 x_1^3 + \gamma_2 x_2^3 + \gamma_3 x_3^3 + \gamma_0$

to the provided data file “generator temperature.csv”.

The objective of the linear regression model is to predict the house price of unit area. Write code in Python to perform following tasks. Please report your output and relevant code in the document file and also include your code (ends with .py) in the .zip file.

i. (10 points) Load the data. Fit the whole dataset to the three linear regression models, respectively. Report the coefficients (αs, βs, γs) of the three models.

Refer 3.py in - engr-ALDA-fall2021-H12/HW4/

The coefficients for α_1, α_2, α_3, and α_0 for Model (1), coefficients for β_1, β_2, β_3, and β_0 for Model (2) and coefficients for γ_1, γ_2, γ_3, and γ_0 for Model (3) are:

```
Coefficients are:

For Model (1) -  [3.4486292  0.06728927 0.9896317  0.01559074]
For Model (2) -  [2.87978802e+01 1.39932134e-04 1.57693190e-02 6.27798319e-06]
For Model (3) -  [4.23239372e+01 2.87274252e-07 3.20011338e-04 1.91742496e-09]
```

ii. (5 points) 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?

```
RMSE Values are:

Leave-one-out RMSE for Model  (1) -  13.948128263247039
Leave-one-out RMSE for Model  (2) -  14.197352235522946
Leave-one-out RMSE for Model  (3) -  15.099194160871548
```

The model with the least Root mean square error - **Model 1** would be the best model to fit the given data.

4. (25 points) [ANN] [Designed and Graded by John Wesley Hostetter] Train, validate, and test a neural network model using the dataset in “ann 2021.zip",which contains training data, validation data, and test data. The goal here is to train the artificial neural network to recognize handwritten digits. Use the Python neural networks package, Keras (i.e. import tensorflow.keras), for this problem. (All the output should be included in your report. Otherwise, your points are deducted.)

(a) (3 points) 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 code in a `README.md' file.

REFER HW4/ann.py file in the GitHub repository for the source code.

Language - Python V3.7.12
Frameworks used - TensorFlow API V2.0, Matplotlib V3.2.2, numpy V1.19.5 and Pandas V1.1.5
We've utilised Tensorflow's API to build, train and test our neural network.
We've used python's package manager 'pip' V21.1.3 to install the dependencies mentioned above.

Steps followed (Assuming you have python and pip installed in your system):
1.Change to the directory you wish to work in.
2.Run `pip install tensorflow numpy pandas matplotlib` in the command line.
3.Store the datasets in the same directory and create a python file with the name of your choice.
4.Run the python file using the command `python3 (insert-filename().py` and record the output.

Also refer the README.md file provided in the GitHub repository.

(b) (2 points) Please apply a fixed random seed of 2021 to generate the same result every time. To do this, you will need to override the `PYTHONHASHSEED' environment variable accessible via the import os. Then, write the following: os.environ[`PYTHONHASHSEED']=`2021'. You should also override the random seed generators found in the `random', `numpy', and `tensorflow' libraries with the value of 2021.

```
import os
os.environ['PYTHONHASHSEED'] = '2021'
random.seed(2021)
np.random.seed(2021)
tf.random.set_seed(2021)
```

(c) (8 points) Create a body of code that iterates over a range of possible number of hidden neurons X = (4; 16; 32; 64) that will be used to define a neural network. For each number of hidden neurons, do the following:

i. Define a neural network with its parameters as follows: activation function for hidden layer = `relu', activation for output layer = `sigmoid', loss function = `binary crossentropy', optimizer = `adam', metrics = `accuracy', epochs=5, batch size=10. The model should have a single hidden layer, with the current number of

hidden neurons in the loop, x, being assigned to define the size of the single hidden layer.
ii. Fit the neural network with the provided training data (this is where you will need the epochs and batch size parameters provided in the previous step).
iii. Validate the neural network that has a hidden layer of size x using the given validation data. The validation accuracy is used to determine how many number of hidden neurons are optimal for this problem. You will want to save this value for later use.
iv. Provide the essential code for the “neural network learning" and include descriptive comments in your report.

```
X = [4,16,32,64]
models = []
model_history = []
# We store the size of the hidden neurons in a list X, the correponding model for each neuron size in models and its metric details in model_history

for x in X:
    print("Number of hidden neurons is ",x)
    # Here we define a Sequential model with `relu` activation function for the hidden layer. `signmoid` function for the output layer
    model = Sequential([
        Dense(units=x, activation='relu'),
        Dense(units=1, activation='sigmoid')
    ])
    # We then compile the model with the following parameters and fit the model with the validation data set included.
    model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])
    history = model.fit(x=train.loc[:, train.columns != 'Class'], y=train['Class'],validation_data=val_dataset, batch_size=10, epochs=5, verbose=2)
    # We append the accuracy metrics to the lists model_history and the model to models.
    models.append(model)
    model_history.append(history)

model_metrics = []
val_metrics = []

for i in model_history:
    model_metrics.append(i.history['accuracy'][-1])
    val_metrics.append(i.history['val_accuracy'][-1])
```

Training the neural network

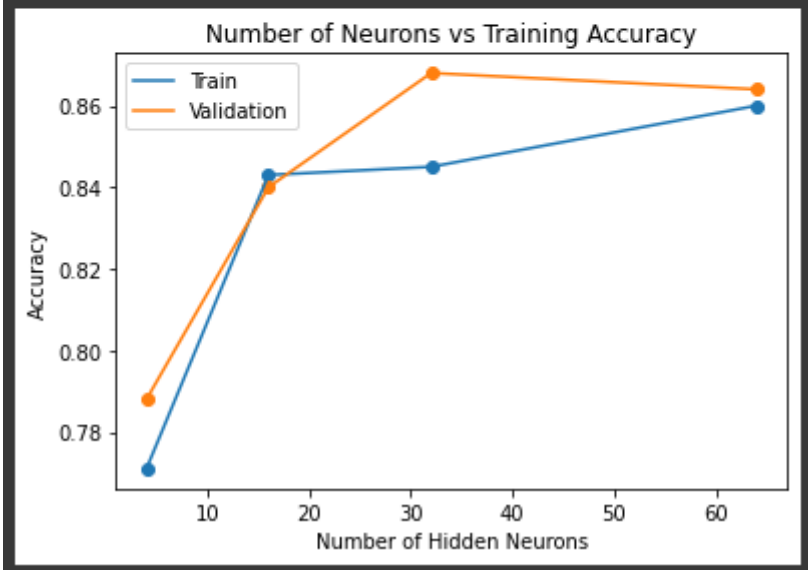
```
Number of hidden neurons is  4
Epoch 1/5
100/100 - 1s - loss: 0.7059 - accuracy: 0.4610 - val_loss: 0.7010 - val_accuracy: 0.4720
Epoch 2/5
100/100 - 0s - loss: 0.6828 - accuracy: 0.5560 - val_loss: 0.6796 - val_accuracy: 0.6200
Epoch 3/5
100/100 - 0s - loss: 0.6593 - accuracy: 0.6440 - val_loss: 0.6493 - val_accuracy: 0.7000
Epoch 4/5
100/100 - 0s - loss: 0.6214 - accuracy: 0.7140 - val_loss: 0.6005 - val_accuracy: 0.7640
Epoch 5/5
100/100 - 0s - loss: 0.5705 - accuracy: 0.7710 - val_loss: 0.5457 - val_accuracy: 0.7880
Number of hidden neurons is  16
Epoch 1/5
100/100 - 1s - loss: 0.6665 - accuracy: 0.6000 - val_loss: 0.6224 - val_accuracy: 0.7280
Epoch 2/5
100/100 - 0s - loss: 0.6037 - accuracy: 0.7400 - val_loss: 0.5588 - val_accuracy: 0.7960
Epoch 3/5
100/100 - 0s - loss: 0.5329 - accuracy: 0.7920 - val_loss: 0.4880 - val_accuracy: 0.8120
Epoch 4/5
100/100 - 0s - loss: 0.4640 - accuracy: 0.8210 - val_loss: 0.4277 - val_accuracy: 0.8200
Epoch 5/5
100/100 - 0s - loss: 0.4117 - accuracy: 0.8430 - val_loss: 0.3879 - val_accuracy: 0.8400
Number of hidden neurons is  32
Epoch 1/5
100/100 - 1s - loss: 0.6728 - accuracy: 0.5770 - val_loss: 0.6196 - val_accuracy: 0.7680
Epoch 2/5
100/100 - 0s - loss: 0.5745 - accuracy: 0.7850 - val_loss: 0.5155 - val_accuracy: 0.8360
Epoch 3/5
100/100 - 0s - loss: 0.4717 - accuracy: 0.8350 - val_loss: 0.4275 - val_accuracy: 0.8320
Epoch 4/5
100/100 - 0s - loss: 0.4003 - accuracy: 0.8390 - val_loss: 0.3792 - val_accuracy: 0.8560
Epoch 5/5
100/100 - 0s - loss: 0.3631 - accuracy: 0.8450 - val_loss: 0.3584 - val_accuracy: 0.8680
Number of hidden neurons is  64
Epoch 1/5
100/100 - 1s - loss: 0.6450 - accuracy: 0.6540 - val_loss: 0.5598 - val_accuracy: 0.7960
Epoch 2/5
100/100 - 0s - loss: 0.5087 - accuracy: 0.8140 - val_loss: 0.4430 - val_accuracy: 0.8280
Epoch 3/5
100/100 - 0s - loss: 0.4130 - accuracy: 0.8470 - val_loss: 0.3785 - val_accuracy: 0.8400
Epoch 4/5
100/100 - 0s - loss: 0.3636 - accuracy: 0.8510 - val_loss: 0.3506 - val_accuracy: 0.8520
Epoch 5/5
100/100 - 0s - loss: 0.3382 - accuracy: 0.8600 - val_loss: 0.3403 - val_accuracy: 0.8640
```

Output for the trained neural network

(d) (5 points) 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 analyse the trend.)

```
import matplotlib.pyplot as plt


plt.plot(X,model_metrics)
plt.scatter(X,model_metrics)
plt.plot(X,val_metrics)
plt.scatter(X,val_metrics)
plt.title('Number of Neurons vs Training Accuracy')
plt.ylabel('Accuracy')
plt.xlabel('Number of Hidden Neurons')
plt.legend(['Train','Validation'])
plt.show()
```



(e) (5 points) Provide a simple analysis about your results and choose the optimal number of hidden neurons from the analysis.

Although the training accuracy rises by a small amount when n = 64 compared to n = 32, the validation accuracy decreases, which can be a result of **overfitting**. But for **n = 32**, we see a sharp rise in validation accuracy with almost similar training accuracy. Thus, we've chosen 32 number of hidden neurons for our artificial neural network since we obtain the **highest training and validation accuracy** for the given value.

(f) (2 points) Report the test accuracy using the given test data on the neural network with the optimal number of hidden neurons.

```
test_acc = []
for model in models:
  test_acc.append(model.evaluate(test.loc[:, test.columns != 'Class'], test['Class']))
```

```
8/8 [==============================] - 0s 2ms/step - loss: 0.5353 - accuracy: 0.8040
8/8 [==============================] - 0s 2ms/step - loss: 0.4015 - accuracy: 0.8360
8/8 [==============================] - 0s 3ms/step - loss: 0.3738 - accuracy: 0.8360
8/8 [==============================] - 0s 2ms/step - loss: 0.3514 - accuracy: 0.8560
```

```
print("Optimal number of neurons in the network is", X[2], "with accuracy", test_acc[2][1])
```

```
Optimal number of neurons in the network is 32 with accuracy 0.8360000252723694
```

5. (20 points) [SVM Programming] [Ge Gao (Designed) & Chengyuan Liu (Graded)] In this question, you will employ SVM to solve a classification problem for the provided "svm_data_2021.csv". This data consists of 15 features and a label for each row that could be 0 or 1. Write code in Python to perform the following tasks. Please report your output and relevant code in the document file and include your code (ends with .py) in the .zip file.

REFER HW4/svm.py file in the GitHub repository for the source code.

(a) (1 point) Load data. Report the size of the positive (class 1) and negative (class 0) samples in dataset.

```
df['Class'].value_counts()
```

```
1.0    125
0.0    125
Name: Class, dtype: int64
```

(b) (2 points) Use stratified random sampling to divide the dataset into training data(80%) and testing data (20%). Report the number of positive and negative samples in both training and testing data.

```
from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(df.loc[:, df.columns != 'Class'],df['Class'],stratify = df['Class'],train_size = 0.8)
```

```
print("Number of Class 0 and Class 1 samples in training data")
print(y_train.value_counts())

print("Number of Class 0 and Class 1 samples in testing data")
print(y_test.value_counts())
```

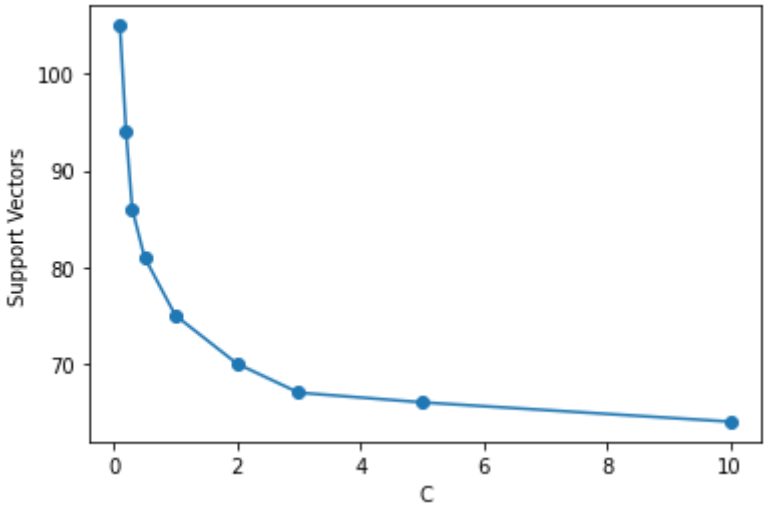
```
Number of Class 0 and Class 1 samples in training data
1.0    100
0.0    100
Name: Class, dtype: int64
Number of Class 0 and Class 1 samples in testing data
1.0    25
0.0    25
Name: Class, dtype: int64
```

(c) (7 points) Using the data given in “svm 2021.zip" in that “train data 2021.csv" as training data, “test data 2021.csv" as testing data, take SVM with linear kernel as classifier (third-party packages are allowed to be used) and set the regularization parameter C as: [0.1, 0.2, 0.3, 0.5, 1, 2, 3, 5, 10], 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 on the plot by explaining: 1) as C increases, how the number of SVs changes, and 2) why.

```
from sklearn.svm import SVC
C = [0.1, 0.2, 0.3, 0.5, 1, 2, 3, 5, 10]
models = []

for c in C:
  model = SVC(C = c,kernel = 'linear')
  models.append(model.fit(train_svm.loc[:,df.columns != 'Class'],train_svm['Class']))
```

```
sv_count = []
for m in models:
  sv_count.append(m.n_support_[0] + m.n_support_[1])
import matplotlib.pyplot as plt
plt.plot(C,sv_count)
plt.xlabel("C")
plt.ylabel("Support Vectors")
plt.scatter(C,sv_count)
plt.show()
```



As C increases, the number of support vectors decreases since the classifier uses hyperplanes that misclassifies lesser points at the expense of a using a smaller margin. When C decreases, the number of support vectors increases since the classifier uses hyperplanes that has a larger margin at the expense of misclassifying more points. In short, it is a trade-off between margin-size and number of misclassified points.

(d) (10 points) Compare the performance of four different kernel functions: linear, polynomial, radial basic function (Gaussian kernel), and Sigmoid. For each type of kernel functions, train your SVM classifiers using the training data and evaluate the resulted SVM classifier using testing data by making a table to record accuracy, precision, recall and f-measure of the corresponding classification results.

For different kernel functions, try to tune its parameters such that:
• for the linear kernel, try to tune C.
• for the polynomial kernel, try to tune C, degree, and coef0.
• for the RBF kernel, try to tune C and
• for the Sigmoid kernel, try to tune C, coef0, and gamma

More specifically, you should explore each of parameters within these ranges:
C = [0.1, 0.2, 0.3, 1, 5, 10, 20, 100, 200, 1000] degree = [1, 2, 3, 4, 5]. coef0 = [0.0001, 0.001, 0.002, 0.01, 0.02, 0.1, 0.2, 0.3, 1, 2, 5, 10]
Gamma = [0.0001, 0.001, 0.002, 0.01, 0.02, 0.03, 0.1, 0.2, 1, 2, 3]
You are strongly encouraged to use sklearn's GridSearchCV() function. If you decide to use this library feature, please set the cross-validation parameter to be 5 (argument of \None" is 5-fold cross validation by default). For each of the four types of kernel functions, please report the best result (using F-measure to break ties) and its corresponding optimal parameters. For this dataset, which kernel function will you choose?

```
from sklearn.model_selection import GridSearchCV

C = [0.1, 0.2, 0.3, 1, 5, 10, 20, 100, 200, 1000]
degree = [1, 2, 3, 4, 5]
coef0 = [0.0001, 0.001, 0.002, 0.01, 0.02, 0.1, 0.2, 0.3, 1, 2, 5, 10]
gamma = [0.0001, 0.001, 0.002, 0.01, 0.02, 0.03, 0.1, 0.2, 1, 2, 3]
kernel = ['linear', 'rbf', 'poly','sigmoid']

svc = SVC()

parameter_linear = {'kernel': [kernel[0]],'C': C}
clf = GridSearchCV(svc,parameter_linear, cv = 5, refit = 'accuracy')
clf.fit(train_svm.loc[:, train_svm.columns != 'Class'], train_svm['Class'])
y_linear = clf.predict(test_svm.loc[:,train_svm.columns != 'Class'])

parameter_poly = {'kernel' : [kernel[2]], 'C': C,'degree': degree, 'coef0': coef0}
clf2 = GridSearchCV(svc,parameter_poly, cv = 5, refit = 'acuracy')
clf2.fit(train_svm.loc[:, train_svm.columns != 'Class'], train_svm['Class'])
y_poly = clf2.predict(test_svm.loc[:,test_svm.columns != 'Class'])

parameter_rbf = {'kernel' : [kernel[1]], 'C': C,'gamma':gamma}
clf3 = GridSearchCV(svc,parameter_rbf, cv = 5, refit = 'accuracy')
clf3.fit(train_svm.loc[:, train_svm.columns != 'Class'], train_svm['Class'])
y_rbf = clf3.predict(test_svm.loc[:,test_svm.columns != 'Class'])

parameter_sigmoid = {'kernel' : [kernel[3]], 'C': C,'gamma':gamma, 'coef0': coef0}
clf4 = GridSearchCV(svc,parameter_sigmoid, cv = 5, refit = 'accuracy')
clf4.fit(train_svm.loc[:, train_svm.columns != 'Class'], train_svm['Class'])
y_sigmoid = clf4.predict(test_svm.loc[:,test_svm.columns != 'Class'])
```

\

```
from sklearn import metrics

print("\tAcc  f1\t\t\tPrecision\t   Recall\t      C deg c0 gamma")
print("-------------------------------------------------------------------
-----------------------------")
print( "Linear ",metrics.accuracy_score(test_svm['Class'],y_linear), me
trics.f1_score(test_svm['Class'], y_linear), metrics.precision_score(te
st_svm['Class'], y_linear),metrics.recall_score(test_svm['Class'], y_li
near),clf.best_estimator_.C, clf.best_estimator_.degree, clf.best_estim
ator_.coef0, "-")
print( "Poly   ", metrics.accuracy_score(test_svm['Class'], y_poly), me
trics.f1_score(test_svm['Class'], y_poly),"", metrics.precision_score(t
est_svm['Class'], y_poly),"",metrics.recall_score(test_svm['Class'], y_
poly),clf2.best_estimator_.C, clf2.best_estimator_.degree, clf2.best_es
timator_.coef0, "-")
print( "RBF    ", metrics.accuracy_score(test_svm['Class'],y_rbf), metr
ics.f1_score(test_svm['Class'],y_rbf), metrics.precision_score(test_svm
['Class'],y_rbf),metrics.recall_score(test_svm['Class'],y_rbf),clf3.bes
t_estimator_.C,'', clf3.best_estimator_.degree, clf3.best_estimator_.co
ef0, clf3.best_estimator_.gamma )
print( "Sigmoid",metrics.accuracy_score(test_svm['Class'],y_sigmoid), m
etrics.f1_score(test_svm['Class'],y_sigmoid), metrics.precision_score(t
est_svm['Class'],y_sigmoid),metrics.recall_score(test_svm['Class'],y_si
gmoid),clf4.best_estimator_.C,'', clf4.best_estimator_.degree, clf4.bes
t_estimator_.coef0, clf4.best_estimator_.gamma)
```

```
        Acc  f1                 Precision          Recall             C deg c0 gamma
-------------------------------------------------------------------------------------------
Linear  0.87 0.8785046728971961 0.8392857142857143 0.9215686274509803 0.1 3 0.0 -
Poly    0.82 0.847457627118644  0.746268656716418  0.9803921568627451 0.1 2 0.3 -
RBF     0.86 0.8679245283018867 0.8363636363636363 0.9019607843137255 20  3 0.0 0.002
Sigmoid 0.79 0.7999999999999999 0.7777777777777778 0.8235294117647058 10  3 0.002 2
```

We choose the Linear kernel since it has overall, the highest values of accuracy, f1 and recall. It has equal precision value with RBF, but since its recall value is higher, we choose the Linear kernel (Precision recall trade-off). Even if the RBF kernel had equal values, we would choose the Linear kernel since it is the simplest model. (Occam's Razor)

6. [30 Bonus Points] [SVM Theory] [Angela Zhang (Designed) & Tyrone Wu (Graded)]

Given 3-dimensional data points$\langle X^i, Y\rangle$, i $\in$ [1, 2, 3, 4] as shown in Table 1. In this question, you will employ the kernel function for SVM trained with these four data points. Let α_1; α_2; α_3 and α_4 be the Lagrangian multipliers associated with them as: α_i is associated with
$\langle X^i, y_i\rangle$.

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

Table 1: Q5(b)

(a) (4 points) Suppose the kernel function is: $K(X^i, X^j) = (2*(X^i .X^j) + 4*(X^i .X^j)^2)$, where X^i and X^j are two data points i and j. This kernel is equal to an inner product $\Phi(Xi).\Phi(Xj)$ with a certain function of Φ. What is the function of Φ?

Give that for the Kernel function, $K(X^i, X^j) = (2*(X^i .X^j) + 4*(X^i .X^j)^2)$

And $(X^i. X^j)$ is the dot product of (X_1^i, X_2^i) and (X_1^j, X_2^j), which is

$(X_1^i, X_2^i).(X_1^j, X_2^j) = (X_1^i. X_1^j) + (X_2^i,.X_2^j)$

If we apply this to the Kernel function,

$K(X^i, X^j)$ $= (2*(X^i .X^j) + 4*(X^i .X^j)^2) = 2((X_1^i. X_1^j) + (X_2^i. X_2^j)) + 4((X_1^i. X_1^j) + (X_2^i. X_2^j))^2 =$

$2(X_1^i. X_1^j) + 2(X_2^i. X_2^j) + 4(X_1^i. X_1^j)^2 + 4(X_2^i. X_2^j)^2 + 8(X_1^i. X_1^j.X_2^i. X_2^j)$

But also $(X^i. X^j) = \Phi(X^i).\Phi(X^j)$, where we can split the equation above symmetrically in :

$\Phi(X^i) = \sqrt{2}(X_1^i) + \sqrt{2}(X_2^i) + 2(X_1^i)^2 + 2(X_2^i)^2 + 2\sqrt{2}(X_1^i.X_2^i)$ & similarly we get

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

The $\Phi(X)$ function can be generalized as:

$\Phi(X) = \sqrt{2}(X_1) + \sqrt{2}(X_2) + 2(X_1)^2 + 2(X_2)^2 + 2\sqrt{2}(X_1.X_2)$.

(b) (3 points) Transform the four given data points X^i, i $\in$ [1, 2, 3, 4] to the higher dimensional space via the function get from (i). Report your results.

Data Points	X_1	X_2	y	$\sqrt{2}(X_1)$	$\sqrt{2}(X_2)$	$2(X_1)^2$	$2(X_2)^2$	$2\sqrt{2}(X_1.X_2)$
X^1	-1	1	1	-1.41	1.41	2	2	-2.82
X^2	-1	-1	1	-1.41	-1.41	2	2	2.82
X^3	1	1	-1	1.41	1.41	2	2	2.82
X^4	1	-1	-1	1.41	-1.41	2	2	-2.82

(c) (10 points) Using the kernel function in (a), what (dual) optimization problem needs to be solved in terms of the is in order to determine their values?

For the Data given, the Dot product would be:

Dot-product $(X^i.X^j)$ i/j	1	2	3	4
1	2	0	0	-2
2	0	2	-2	0
3	0	-2	2	0
4	-2	0	0	2

And the Kernel values would be wrt to the equation - $\mathbf{K(X^i, X^j) = (2*(X^i .X^j) + 4*(X^i .X^j)^2)}$

$K(X^i, X^j)$ i/j	1	2	3	4
1	20	0	0	12
2	0	20	12	0
3	0	12	20	0
4	12	0	0	20

We use the formula: $\sum \boldsymbol{\alpha_i - ½ \sum_i \sum_j \alpha_i.\alpha_j.K(x_i x_j).y_i y_j}$

$J = \alpha_1 + \alpha_2 + \alpha_3 + \alpha_4 - ½*(\alpha_1^2(20)(1) + \alpha_1\alpha_4(12)(-1) + \alpha_2^2(20)(1) + \alpha_2\alpha_3(12)(-1) + \alpha_2\alpha_3(12)(-1) + \alpha_3^2(20)(1) + \alpha_1\alpha_4(12)(-1) + \alpha_4^2(20)(1)) =$

$\boldsymbol{J = \alpha_1 + \alpha_2 + \alpha_3 + \alpha_4 + 12(\alpha_1\alpha_4 + \alpha_2\alpha_3) - 10(\alpha_1^2 + \alpha_2^2 + \alpha_3^2 + \alpha_4^2)}$

And also we consider a constraint: $\boldsymbol{\sum_i \alpha_i y_i >= 0}$

$\boldsymbol{\alpha_1 + \alpha_2 - \alpha_3 - \alpha_4 = 0}$ and $\boldsymbol{\alpha > 0.}$

(d) (10 points) Assume that the solution to the optimization problem shows that $\alpha_1 = \alpha_2 = \alpha_3 = \alpha_4 = 1/8$. Apply Lagrange multipliers to determine the maximum margin linear decision boundary in the transformed higher dimensional space. Note that the maximum margin linear decision boundary is defined as w(x)+b, where the transformation is already defined by the kernel above and b is constant. (Show your work).

Given that $\alpha_1 = \alpha_2 = \alpha_3 = \alpha_4 = ⅛$, the Maximum Margin Linear Decision Boundry:

$W.\Phi(X) + b$ and taking the equation for X^1 -

$W.\Phi(X) + b = y^1$

$\sum \alpha_i y_i \Phi(X^i).\Phi(X^1) + b = y^1$

$\alpha_1 y_1 \Phi(X^1).\Phi(X^1) + \alpha_2 y_2 \Phi(X^2).\Phi(X^1) + \alpha_3 y_3 \Phi(X^3).\Phi(X^1) + \alpha_4 y_4 \Phi(X^4).\Phi(X^1) + b = y^1$

$\alpha_1 y_1 K(X^1, X^1) + \alpha_2 y_2 K(X^2, X^1) + \alpha_3 y_3 K(X^3, X^1) + \alpha_4 y_4 K(X^4, X^1) + b = y^1$

Substituting the values from above:

$(⅛)(1)(20) + 0 + 0 + (⅛)(-1)(12) + b = 1$

$1 + b = 1$

$b = 0.$

The hyperplane equation would be : $⅛ \Phi(X^1) + ⅛ \Phi(X^2) - ⅛ \Phi(X^3) - ⅛ \Phi(X^4)$.

The Maximum margin Linear decision boundary would be =

$\alpha_1 y_1\ \Phi(X^1).\Phi(X^1) + \alpha_2 y_2\ \Phi(X^2).\Phi(X^1) + \alpha_3 y_3\ \Phi(X^3).\Phi(X^1) + \alpha_4 y_4\ \Phi(X^4).\Phi(X^1)$ =

$\alpha_1 y_1\ K(X^1, X^1) + \alpha_2 y_2\ K(X^2, X^1) + \alpha_3 y_3\ K(X^3, X^1) + \alpha_4 y_4\ K(X^4, X^1)$

Substituting values from the table in 6(c):

$(\frac{1}{8})(1)(20) + 0 + 0 + (\frac{1}{8})(-1)(12) = 8/8 = 1.$

Maximum margin Linear decision = 1.

(e) (3 points) Use the maximum margin linear decision boundary from (d) to classify point (-2,-2).

Dot-product ($X^i.X^j$) i/j	**1**	**2**	**3**	**4**
X^5	0	4	-4	0

$K(X^i, X^j)$ i/j	**1**	**2**	**3**	**4**
X^5	0	72	56	0

With the new data point the Maximum Linear Decision Boundary equation is:

$W.\Phi((-2,-2)) = W.\Phi(X^n) = (\frac{1}{8}\ \Phi(X^1) + \frac{1}{8}\ \Phi(X^2) - \frac{1}{8}\ \Phi(X^3) - \frac{1}{8}\ \Phi(X^4)) * \Phi(X^n)$

$= \frac{1}{8}\ K(X^1, X^n) + \frac{1}{8}\ K(X^2, X^n) + \frac{1}{8}\ K(X^3, X^n) + \frac{1}{8}\ K(X^4, X^n)$

$= \frac{1}{8}(0) + \frac{1}{8}(72) - \frac{1}{8}(56) - \frac{1}{8}(0) = 16/8 = 2.$

So the new data point - X^n is classified as **positive**.