****

DEPARTMENTOFARTIFICIALINTELLIGENCE  
ANDMACHINELEARNING

**Machine Learning**

***Lab Manual***

BATCH: 2020 – 2024

YEAR: III

SEMESTER: V

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **MACHINE LEARNING LAB** | | | | | | | | | |
|  | | | | | | | | | |
|  | | | | | | | | | |
| **Course Code** | | | **:** | **20AIL57A** | | |  | **Credits:** | **2** |
| **L:** | **T:** | **P** | **:** | **0:** | **0:** | **2** |  | **CIE Marks:** | **25** |
| **Exam Hours:** | | | **:** | **3** | | |  | **SEE Marks:** | **25** |

|  |  |  |
| --- | --- | --- |
| **Course Outcomes:** | | **At the end of the Course, the Student will be able to** |
| CO# | COURSE OUTCOME | |
| 20AIL57A.1 | |  | | --- | | Understand the implementation of procedures for machine learning algorithms. | | |
| 20AIL57A.2 | |  | | --- | | Design Java/Python programs for various Learning algorithms. | | |
| 20AIL57A.3 | |  | | --- | | Analyze and apply the appropriate data sets for Machine Learning algorithms. | | |
| 20AIL57A.4 | |  | | --- | | Identify and apply Machine Learning algorithms to solve real world problems. | | |

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Mapping of Course Outcomes to Program Outcomes** | | | | | | | | | | | | | | |
|  | PO1 | PO2 | PO3 | PO4 | PO5 | PO6 | PO7 | PO8 | PO9 | PO10 | PO11 | PO12 | PSO1 | PSO2 |
| 20AIL57A.1 | - | - | - | - | - | - | - | - | - | - | - | 3 | 3 | 2 |
| 20AIL57A.2 | 3 | - | - | - | - | - | - | - | - | - | - | 3 | 3 | 2 |
| 20AIL57A.3 | - | 3 | - | - | 3 | - | - | - | - | - | - | 3 | 3 | 2 |
| 20AIL57A.4 | - | - | - | 3 | 3 | - | - | - | - | - | - | 3 | 3 | 2 |
| Correlation levels: 1-Slight(Low) 2-Moderate(Medium) 3-Substantial(High) | | | | | | | | | | | | | | |

|  |  |  |  |
| --- | --- | --- | --- |
| **Ex. No** | **Experiments** | **Hours** | **COs** |
|  | |  | | --- | | Implement and demonstrate the Linear discriminant Analysis (LDA). | | **3** | |  | | --- | | CO1, CO2, CO3, CO4 | |
|  | |  | | --- | | Develop a Support Vector Machine model considering a Sample Dataset and evaluate the model. | | **3** | CO1, CO2, CO3, CO4 |
|  | |  | | --- | | Write a program to demonstrate the working of the decision tree Based CART algorithm. Use an appropriate data set for building the decision tree and apply this knowledge to classify a new Sample. | | **3** | CO1, CO2, CO3, CO4 |
|  | |  | | --- | | Develop a simple regression model for the given dataset and evaluate its performance. | | **3** | CO1, CO2, CO3, CO4 |
|  | |  | | --- | | Apply multivariate regression model using suitable library function to make necessary predictions. | | **3** | CO1, CO2, CO3, CO4 |
|  | |  | | --- | | Implement a program in python to illustrate the Bias Variance Trade-off in a machine learning model | | **3** | CO1, CO2, CO3, CO4 |
|  | |  | | --- | | Apply k-means algorithm to generate clusters for the given dataset and evaluate its performance. | | **3** | CO1, CO2, CO3, CO4 |
|  | |  | | --- | | Implement and demonstrate the Principal Component analysis (PCA) | | **3** | CO1, CO2, CO3, CO4 |
|  | Implement Reinforcement learning with suitable example. | **3** | CO1, CO2, CO3, CO4 |
|  | Implement text classification model using suitable algorithm. | **3** | CO1, CO2, CO3, CO4 |

|  |  |
| --- | --- |
| **Text Books:**  1. Tom Mitchell, “Machine Learning”, McGraw Hill, 1997  2. E. Alpaydin, “Introduction to Machine Learning”, PHI, 2005.  **Reference Books:**  1. AurolienGeron, ”Hands-On Machine Learning with Scikit-Learn and TensorFlow, Shroff/O’Reilly”,2017  2. Andreas Muller and Sarah Guido, ”Introduction to Machine Learning with Python: A Guidefor Data Scientists”, Shroff/O’Reilly, 2016 | |
|  |  |
|  |  |

**CIE- Continuous Internal Evaluation (25 Marks)**

|  |  |
| --- | --- |
| **Bloom’s Category** | **Tests**  **(25 marks)** |
| Remember | **-** |
| Understand | 5 |
| Apply | 15 |
| Analyze | 5 |
| Evaluate | - |
| Create | - |

**SEE- Semester End Examination (25Marks)**

|  |  |
| --- | --- |
| **Bloom’s Category** | **Questions (50 marks)** |
| Remember | **-** |
| Understand | 5 |
| Apply | 15 |
| Analyze | 5 |
| Evaluate | - |
| Create | - |

1. Implement and demonstrate the Linear discriminant Analysis (LDA).

importnumpy as np

import pandas as pd

import matplotlib.pyplot as plt

import sklearn

from sklearn.preprocessing import StandardScaler, LabelEncoder

from sklearn.model\_selection import train\_test\_split

from sklearn.discriminant\_analysis import LinearDiscriminantAnalysis

from sklearn.ensemble import RandomForestClassifier

from sklearn.metrics import accuracy\_score, confusion\_matrix

# read dataset from URL

url = "https://archive.ics.uci.edu/ml/machine-learning-databases/iris/iris.data"

cls = ['sepal-length', 'sepal-width', 'petal-length', 'petal-width', 'Class']

dataset = pd.read\_csv(url, names=cls)

# divide the dataset into class and target variable

X = dataset.iloc[:, 0:4].values

y = dataset.iloc[:, 4].values

# Preprocess the dataset and divide into train and test

sc = StandardScaler()

X = sc.fit\_transform(X)

le = LabelEncoder()

y = le.fit\_transform(y)

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2)

# apply Linear Discriminant Analysis

lda = LinearDiscriminantAnalysis(n\_components=2)

X\_train = lda.fit\_transform(X\_train, y\_train)

X\_test = lda.transform(X\_test)

# plot the scatterplot

plt.scatter(

X\_train[:,0],X\_train[:,1],c=y\_train,cmap='rainbow',

alpha=0.7,edgecolors='b'

)

# classify using random forest classifier

classifier = RandomForestClassifier(max\_depth=2, random\_state=0)

classifier.fit(X\_train, y\_train)

y\_pred = classifier.predict(X\_test)

# print the accuracy and confusion matrix

print('Accuracy : ' + str(accuracy\_score(y\_test, y\_pred)))

conf\_m = confusion\_matrix(y\_test, y\_pred)

print(conf\_m)

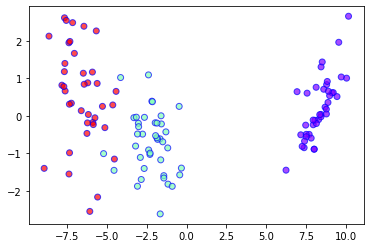
**Output:**

Accuracy : 1.0

[[ 9 0 0]

[ 0 9 0]

[ 0 0 12]]



|  |
| --- |
| 1. Develop a Support Vector Machine model considering a Sample Dataset and evaluate the model. |

import pandas as pd

import matplotlib.pyplot as plt

import seaborn as sns

#Define the col names

colnames=["sepal\_length\_in\_cm", "sepal\_width\_in\_cm","petal\_length\_in\_cm","petal\_width\_in\_cm", "class"]

#Read the dataset

dataset = pd.read\_csv("https://archive.ics.uci.edu/ml/machine-learning-databases/iris/iris.data", header = None, names= colnames )

#Data

dataset.head()

#Encoding the categorical column

dataset = dataset.replace({"class": {"Iris-setosa":1,"Iris-versicolor":2, "Iris-virginica":3}})

#Visualize the new dataset

dataset.head()

plt.figure(1)

sns.heatmap(dataset.corr())

plt.title('Correlation On iris Classes')

X = dataset.iloc[:,:-1]

y = dataset.iloc[:, -1].values

from sklearn.model\_selection import train\_test\_split

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size = 0.25, random\_state = 0)

#Create the SVM model

from sklearn.svm import SVC

classifier = SVC(kernel = 'linear', random\_state = 0)

#Fit the model for the data

classifier.fit(X\_train, y\_train)

#Make the prediction

y\_pred = classifier.predict(X\_test)

from sklearn.metrics import confusion\_matrix,classification\_report

cm = confusion\_matrix(y\_test, y\_pred)

print(cm)

print(classification\_report(y\_test, y\_pred))

from sklearn.model\_selection import cross\_val\_score

accuracies = cross\_val\_score(estimator = classifier, X = X\_train, y = y\_train, cv = 10)

print("Accuracy: {:.2f} %".format(accuracies.mean()\*100))

print("Standard Deviation: {:.2f} %".format(accuracies.std()\*100))

Output:

[[13 0 0]

[ 0 15 1]

[ 0 0 9]]

Accuracy: 98.18 %

Standard Deviation: 3.64 %

1. Write a program to demonstrate the working of the decision tree Based CART algorithm. Use an appropriate data set for building the decision tree and apply this knowledge to classify a new Sample.

import pandas as pd

from sklearn.tree import DecisionTreeClassifier

df=pd.read\_csv(r"C:\Users\HAI\Desktop\Kavya\NHCE\Academic Year 22-23(ODD)\Machine Learning\programs\loan.csv")

df.head(10)

print(df.isnull().sum())

df\_encoded=df.copy()

print(df\_encoded['Gender'].value\_counts())

df\_encoded['Gender'].fillna('Male',inplace=True)

df\_encoded['Married'].fillna('Yes',inplace=True)

df\_encoded['Dependents'].fillna(0,inplace=True)

df\_encoded['Self\_Employed'].fillna('No',inplace=True)

df\_encoded['LoanAmount'].fillna(df\_encoded['LoanAmount'].mean(),inplace=True)

df\_encoded['Loan\_Amount\_Term'].fillna(df\_encoded['Loan\_Amount\_Term'].mean(),inplace=True)

df\_encoded['Credit\_History'].fillna(1.0,inplace=True)

print(df\_encoded.isnull().sum())

from sklearn import preprocessing

le = preprocessing.LabelEncoder()

df\_encoded['Loan\_ID']=le.fit\_transform(df\_encoded['Gender'].values)

df\_encoded['Gender']=le.fit\_transform(df\_encoded['Gender'].values)

df\_encoded['Married']=le.fit\_transform(df\_encoded['Married'].values)

df\_encoded['Dependents']=le.fit\_transform(df\_encoded['Dependents'].values)

df\_encoded['Education']=le.fit\_transform(df\_encoded['Education'].values)

df\_encoded['Self\_Employed']=le.fit\_transform(df\_encoded['Self\_Employed'].values)

df\_encoded['Credit\_History']=le.fit\_transform(df\_encoded['Credit\_History'].values)

df\_encoded['Property\_Area']=le.fit\_transform(df\_encoded['Property\_Area'].values)

df\_encoded['Loan\_Status']=le.fit\_transform(df\_encoded['Loan\_Status'].values)

df\_encoded['ApplicantIncome']=le.fit\_transform(df\_encoded['ApplicantIncome'].values)

df\_encoded['CoapplicantIncome']=le.fit\_transform(df\_encoded['CoapplicantIncome'].values)

df\_encoded['LoanAmount']=le.fit\_transform(df\_encoded['LoanAmount'].values)

df\_encoded['Loan\_Amount\_Term']=le.fit\_transform(df\_encoded['Loan\_Amount\_Term'].values)

print(df\_encoded.head())

from sklearn.model\_selection import train\_test\_split

x = df\_encoded[feature\_cols]

y = df\_encoded['Loan\_Status']

from sklearn.metrics import confusion\_matrix, accuracy\_score, recall\_score, roc\_curve, auc

X\_train,X\_test,y\_train,y\_test=train\_test\_split(x,y,test\_size=0.3,random\_state=1)

treeModel = DecisionTreeClassifier(random\_state=0, max\_depth=2, criterion='gini').fit(X\_train,y\_train)

treeModel.fit(X\_train,y\_train)

PredictedOutput = treeModel.predict(X\_test)

from sklearn import tree

tree.plot\_tree(treeModel);

cm = confusion\_matrix(y\_test, PredictedOutput)

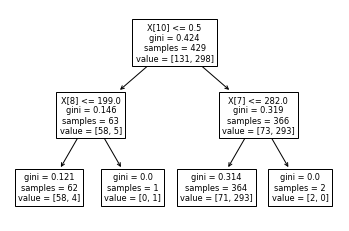
print(cm)

print(classification\_report(y\_test, PredictedOutput))

accuracy = accuracy\_score(y\_test, PredictedOutput)

accuracy

Output:



1. Develop a simple regression model for the given dataset and evaluate its performance.

import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

dataset = pd.read\_csv('Salary.csv')

dataset.head()

X = dataset.iloc[:,:-1].values #independent variable array

y = dataset.iloc[:,1].values #dependent variable vector

from sklearn.model\_selection import train\_test\_split

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X,y,test\_size=1/3,random\_state=0)

from sklearn.linear\_model import LinearRegression

regressor = LinearRegression()

regressor.fit(X\_train,y\_train) #actually produces the linear eqn for the data

y\_pred = regressor.predict(X\_test)

y\_pred

y\_test

#plot for the TRAIN

plt.scatter(X\_train, y\_train, color='red') # plotting the observation line

plt.plot(X\_train, regressor.predict(X\_train), color='blue') # plotting the regression line

plt.title("Salary vs Experience (Training set)") # stating the title of the graph

plt.xlabel("Years of experience") # adding the name of x-axis

plt.ylabel("Salaries") # adding the name of y-axis

plt.show() # specifies end of graph

#plot for the TEST

plt.scatter(X\_test, y\_test, color='red')

plt.plot(X\_train, regressor.predict(X\_train), color='blue') # plotting the regression line

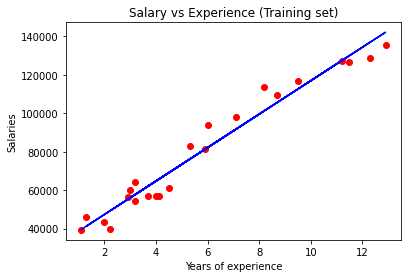
plt.title("Salary vs Experience (Testing set)")

plt.xlabel("Years of experience")

plt.ylabel("Salaries")

plt.show()

Output:



1. Apply multivariate regression model using suitable library function to make necessary predictions.

import pandas as pd

importnumpy as np

importmatplotlib.pyplot as plt

importseaborn as sns

dataset = pd.read\_csv("D:\BackUp\_Rajasree\_02Aug2020\DataSet\Advertising.csv")

dataset.head()

x = dataset[['TV', 'Radio', 'Newspaper']]

y = dataset['Sales']

fromsklearn.model\_selection import train\_test\_split

x\_train, x\_test, y\_train, y\_test = train\_test\_split(x, y, test\_size = 0.3, random\_state = 100)

fromsklearn.linear\_model import LinearRegression

#Fitting the Multiple Linear Regression model

mlr = LinearRegression()

mlr.fit(x\_train, y\_train)

#Intercept and Coefficient

print("Intercept: ", mlr.intercept\_)

print("Coefficients:")

list(zip(x, mlr.coef\_))

#Prediction of test set

y\_pred\_mlr= mlr.predict(x\_test)

#Predicted values

print("Prediction for test set: {}".format(y\_pred\_mlr))

#Actual value and the predicted value

mlr\_diff = pd.DataFrame({'Actual value': y\_test, 'Predicted value': y\_pred\_mlr})

mlr\_diff.head()

#Model Evaluation

fromsklearn import metrics

meanAbErr = metrics.mean\_absolute\_error(y\_test, y\_pred\_mlr)

meanSqErr = metrics.mean\_squared\_error(y\_test, y\_pred\_mlr)

rootMeanSqErr = np.sqrt(metrics.mean\_squared\_error(y\_test, y\_pred\_mlr))

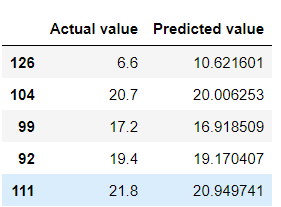
print('R squared: {:.2f}'.format(mlr.score(x,y)\*100))

print('Mean Absolute Error:', meanAbErr)

print('Mean Square Error:', meanSqErr)

print('Root Mean Square Error:', rootMeanSqErr)

Output:



R squared: 89.59

Mean Absolute Error: 1.0638483124072025

Mean Square Error: 1.8506819941636963

Root Mean Square Error: 1.3603977338130553

1. Implement a program in python to illustrate the Bias Variance Trade-off in a machine learning model

# estimate the bias and variance for a regression model

from pandas import read\_csv

fromsklearn.model\_selection import train\_test\_split

fromsklearn.linear\_model import LinearRegression

frommlxtend.evaluate import bias\_variance\_decomp

# load dataset

dataframe = read\_csv("D:\BackUp\_Rajasree\_02Aug2020\DataSet\housing.csv", header=None)

# separate into inputs and outputs

data = dataframe.values

X, y = data[:, :-1], data[:, -1]

# split the data

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.33, random\_state=1)

# define the model

model = LinearRegression()

# estimate bias and variance

mse, bias, var = bias\_variance\_decomp(model, X\_train, y\_train, X\_test, y\_test, loss='mse', num\_rounds=200, random\_seed=1)

# summarize results

print('MSE: %.3f' % mse)

print('Bias: %.3f' % bias)

print('Variance: %.3f' % var)

Output:

MSE: 22.418

Bias: 20.744

Variance: 1.674

1. Apply k-means algorithm to generate clusters for the given dataset and evaluate its performance.

import pandas as pd

importmatplotlib.pyplot as plt

importnumpy as np

importseaborn as sns

df\_cust=pd.read\_csv(r"C:\Users\HAI\Desktop\Kavya\NHCE\Academic Year 22-23(ODD)\Machine Learning\programs\Mall\_Customers.csv")

df\_cust.head()

df\_cust.corr()

df\_cust.columns

df\_cust.drop(["CustomerID"],axis=1,inplace=True)

df\_cust.plot.scatter(x='Age',y='Spending Score (1-100)')

sns.countplot(x='Genre',data=df\_cust)

plt.figure(figsize=(12,10))

sns.countplot(x='Age',data=df\_cust)

#ImportingKMeans from sklearn

fromsklearn.cluster import KMeans

X=df\_cust[["Annual Income (k$)","Spending Score (1-100)"]]

## Elbow Method to find no of clusters

wcss=[]

fori in range(1,11):

km=KMeans(n\_clusters=i)

km.fit(X)

wcss.append(km.inertia\_)

#Taking 5 clusters

km1=KMeans(n\_clusters=5)

#Fitting the input data

km1.fit(X)

#predicting the labels of the input data

y=km1.predict(X)

#adding the labels to a column named label

df\_cust["label"] = y

#The new dataframe with the clustering done

df\_cust.head()

#Scatterplot of the clusters

plt.figure(figsize=(10,6))

sns.scatterplot(x = 'Annual Income (k$)',y = 'Spending Score (1-100)',hue="label",

palette=['green','orange','brown','dodgerblue','red'], legend='full',data = df\_cust ,s = 60 )

plt.xlabel('Annual Income (k$)')

plt.ylabel('Spending Score (1-100)')

plt.title('Spending Score (1-100) vs Annual Income (k$)')

plt.show()

fromsklearn.cluster import AgglomerativeClustering

hc = AgglomerativeClustering(n\_clusters = 5, affinity = 'euclidean', linkage = 'ward')

y\_hc = hc.fit\_predict(X)

y\_hc

#Scatterplot of the clusters

plt.figure(figsize=(10,6))

sns.scatterplot(x = 'Annual Income (k$)',y = 'Spending Score (1-100)',hue="label",

palette=['green','orange','brown','dodgerblue','red'], legend='full',data = df\_cust ,s = 60 )

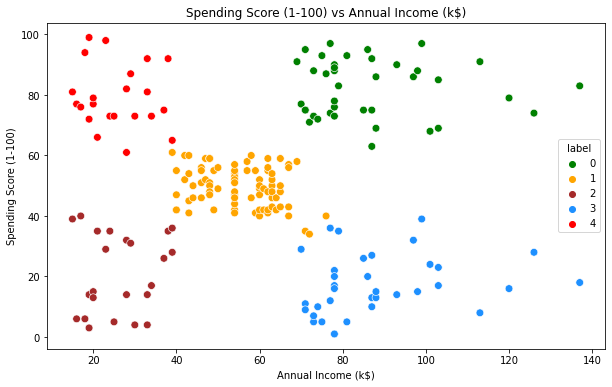
plt.xlabel('Annual Income (k$)')

plt.ylabel('Spending Score (1-100)')

plt.title('Spending Score (1-100) vs Annual Income (k$)')

plt.show()

Output:



|  |
| --- |
| 1. Implement and demonstrate the Principal Component analysis (PCA) |

import pandas as pd

importmatplotlib.pyplot as plt

url = "https://archive.ics.uci.edu/ml/machine-learning-databases/iris/iris.data"

# load dataset into Pandas DataFrame

df = pd.read\_csv(url, names=['sepal length','sepalwidth','petallength','petalwidth','target'])

fromsklearn.preprocessing import StandardScaler

features = ['sepal length', 'sepal width', 'petal length', 'petal width']

# Separating out the features

x = df.loc[:, features].values

# Separating out the target

y = df.loc[:,['target']].values

# Standardizing the features

x = StandardScaler().fit\_transform(x)

fromsklearn.preprocessing import StandardScaler

features = ['sepal length', 'sepal width', 'petal length', 'petal width']

# Separating out the features

x = df.loc[:, features].values

# Separating out the target

y = df.loc[:,['target']].values

# Standardizing the features

x = StandardScaler().fit\_transform(x)

fromsklearn.decomposition import PCA

pca = PCA(n\_components=2)

principalComponents = pca.fit\_transform(x)

principalDf = pd.DataFrame(data = principalComponents

, columns = ['principal component 1', 'principal component 2'])

finalDf = pd.concat([principalDf, df[['target']]], axis = 1)

fig = plt.figure(figsize = (8,8))

ax = fig.add\_subplot(1,1,1)

ax.set\_xlabel('Principal Component 1', fontsize = 15)

ax.set\_ylabel('Principal Component 2', fontsize = 15)

ax.set\_title('2 component PCA', fontsize = 20)

targets = ['Iris-setosa', 'Iris-versicolor', 'Iris-virginica']

colors = ['r', 'g', 'b']

for target, color in zip(targets,colors):

indicesToKeep = finalDf['target'] == target

ax.scatter(finalDf.loc[indicesToKeep, 'principal component 1']

, finalDf.loc[indicesToKeep, 'principal component 2']

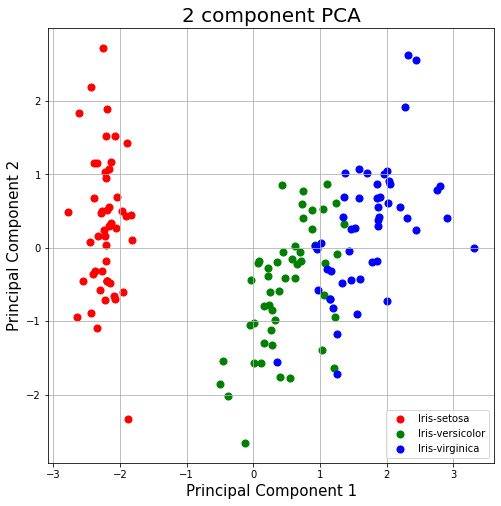
, c = color

, s = 50)

ax.legend(targets)

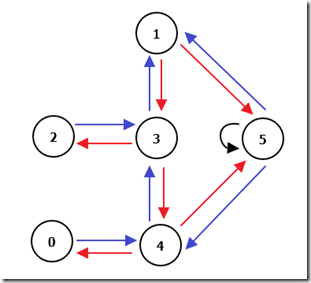
ax.grid()

Output:



1. Implement Reinforcement learning with suitable example.

The example shows a maze through which the agent should go and find its way up to the goal stage (stage 5). Basically, the idea is to train an algorithm to find the optimal path, given an (often random) initial condition, in order to maximize a certain outcome.



importnumpy as np

# R matrix

R = np.matrix([ [-1,-1,-1,-1,0,-1],

[-1,-1,-1,0,-1,100],

[-1,-1,-1,0,-1,-1],

[-1,0,0,-1,0,-1],

[-1,0,0,-1,-1,100],

[-1,0,-1,-1,0,100] ])

# Q matrix

Q = np.matrix(np.zeros([6,6]))

# Gamma (learning parameter).

gamma = 0.8

# Initial state. (Usually to be chosen at random)

initial\_state = 1

# This function returns all available actions in the state given as an argument

defavailable\_actions(state):

current\_state\_row = R[state,]

av\_act = np.where(current\_state\_row>= 0)[1]

returnav\_act

# Get available actions in the current state

available\_act = available\_actions(initial\_state)

# This function chooses at random which action to be performed within the range

# of all the available actions.

defsample\_next\_action(available\_actions\_range):

next\_action = int(np.random.choice(available\_act,1))

returnnext\_action

# Sample next action to be performed

action = sample\_next\_action(available\_act)

# This function updates the Q matrix according to the path selected and the Q

# learning algorithm

def update(current\_state, action, gamma):

max\_index = np.where(Q[action,] == np.max(Q[action,]))[1]

ifmax\_index.shape[0] > 1:

max\_index = int(np.random.choice(max\_index, size = 1))

else:

max\_index = int(max\_index)

max\_value = Q[action, max\_index]

# Q learning formula

Q[current\_state, action] = R[current\_state, action] + gamma \* max\_value

# Update Q matrix

update(initial\_state,action,gamma)

#-------------------------------------------------------------------------------

# Training

# Train over 10 000 iterations. (Re-iterate the process above).

fori in range(10000):

current\_state = np.random.randint(0, int(Q.shape[0]))

available\_act = available\_actions(current\_state)

action = sample\_next\_action(available\_act)

update(current\_state,action,gamma)

# Normalize the "trained" Q matrix

print("Trained Q matrix:")

print(Q/np.max(Q)\*100)

#-------------------------------------------------------------------------------

# Testing

# Goal state = 5

# Best sequence path starting from 2 -> 2, 3, 1, 5

current\_state = 2

steps = [current\_state]

whilecurrent\_state != 5:

next\_step\_index = np.where(Q[current\_state,] == np.max(Q[current\_state,]))[1]

if next\_step\_index.shape[0] > 1:

next\_step\_index = int(np.random.choice(next\_step\_index, size = 1))

else:

next\_step\_index = int(next\_step\_index)

steps.append(next\_step\_index)

current\_state = next\_step\_index

# Print selected sequence of steps

print("Selected path:")

print(steps)

#-------------------------------------------------------------------------------

# OUTPUT

#-------------------------------------------------------------------------------

#

# Trained Q matrix:

#[[ 0. 0. 0. 0. 80. 0. ]

# [ 0. 0. 0. 64. 0. 100. ]

# [ 0. 0. 0. 64. 0. 0. ]

# [ 0. 80. 51.2 0. 80. 0. ]

# [ 0. 80. 51.2 0. 0. 100. ]

# [ 0. 80. 0. 0. 80. 100. ]]

#

# Selected path:

# [2, 3, 1, 5]

#

1. Implement text classification model using suitable algorithm.

fromkeras.preprocessing import text, sequence

fromkeras import layers, models, optimizers

# load the dataset

data = open('data/corpus').read()

labels, texts = [], []

fori, line in enumerate(data.split("\n")):

content = line.split()

labels.append(content[0])

texts.append(" ".join(content[1:]))

# create a dataframe using texts and lables

trainDF = pandas.DataFrame()

trainDF['text'] = texts

trainDF['label'] = labels

# split the dataset into training and validation datasets

train\_x, valid\_x, train\_y, valid\_y = model\_selection.train\_test\_split(trainDF['text'], trainDF['label'])

# label encode the target variable

encoder = preprocessing.LabelEncoder()

train\_y = encoder.fit\_transform(train\_y)

valid\_y = encoder.fit\_transform(valid\_y)

# create a count vectorizer object

count\_vect = CountVectorizer(analyzer='word', token\_pattern=r'\w{1,}')

count\_vect.fit(trainDF['text'])

# transform the training and validation data using count vectorizer object

xtrain\_count= count\_vect.transform(train\_x)

xvalid\_count= count\_vect.transform(valid\_x)

# word level tf-idf

tfidf\_vect = TfidfVectorizer(analyzer='word', token\_pattern=r'\w{1,}', max\_features=5000)

tfidf\_vect.fit(trainDF['text'])

xtrain\_tfidf= tfidf\_vect.transform(train\_x)

xvalid\_tfidf= tfidf\_vect.transform(valid\_x)

# ngram level tf-idf

tfidf\_vect\_ngram = TfidfVectorizer(analyzer='word', token\_pattern=r'\w{1,}', ngram\_range=(2,3), max\_features=5000)

tfidf\_vect\_ngram.fit(trainDF['text'])

xtrain\_tfidf\_ngram= tfidf\_vect\_ngram.transform(train\_x)

xvalid\_tfidf\_ngram= tfidf\_vect\_ngram.transform(valid\_x)

# characters level tf-idf

tfidf\_vect\_ngram\_chars = TfidfVectorizer(analyzer='char', token\_pattern=r'\w{1,}', ngram\_range=(2,3), max\_features=5000)

tfidf\_vect\_ngram\_chars.fit(trainDF['text'])

xtrain\_tfidf\_ngram\_chars= tfidf\_vect\_ngram\_chars.transform(train\_x)

xvalid\_tfidf\_ngram\_chars= tfidf\_vect\_ngram\_chars.transform(valid\_x)

# load the pre-trained word-embedding vectors

embeddings\_index = {}

fori, line in enumerate(open('data/wiki-news-300d-1M.vec')):

values = line.split()

embeddings\_index[values[0]] = numpy.asarray(values[1:], dtype='float32')

# create a tokenizer

token = text.Tokenizer()

token.fit\_on\_texts(trainDF['text'])

word\_index = token.word\_index

# convert text to sequence of tokens and pad them to ensure equal length vectors

train\_seq\_x = sequence.pad\_sequences(token.texts\_to\_sequences(train\_x), maxlen=70)

valid\_seq\_x = sequence.pad\_sequences(token.texts\_to\_sequences(valid\_x), maxlen=70)

# create token-embedding mapping

embedding\_matrix = numpy.zeros((len(word\_index) + 1, 300))

for word, i in word\_index.items():

embedding\_vector = embeddings\_index.get(word)

ifembedding\_vector is not None:

embedding\_matrix[i] = embedding\_vector

trainDF['char\_count'] = trainDF['text'].apply(len)

trainDF['word\_count'] = trainDF['text'].apply(lambda x: len(x.split()))

trainDF['word\_density'] = trainDF['char\_count'] / (trainDF['word\_count']+1)

trainDF['punctuation\_count'] = trainDF['text'].apply(lambda x: len("".join(\_ for \_ in x if \_ in string.punctuation)))

trainDF['title\_word\_count'] = trainDF['text'].apply(lambda x: len([wrd for wrd in x.split() if wrd.istitle()]))

trainDF['upper\_case\_word\_count'] = trainDF['text'].apply(lambda x: len([wrd for wrd in x.split() if wrd.isupper()]))

pos\_family = {

'noun' : ['NN','NNS','NNP','NNPS'],

'pron' : ['PRP','PRP$','WP','WP$'],

'verb' : ['VB','VBD','VBG','VBN','VBP','VBZ'],

'adj' : ['JJ','JJR','JJS'],

'adv' : ['RB','RBR','RBS','WRB']

}

# function to check and get the part of speech tag count of a words in a given sentence

defcheck\_pos\_tag(x, flag):

cnt = 0

try:

wiki = textblob.TextBlob(x)

fortup in wiki.tags:

ppo = list(tup)[1]

ifppo in pos\_family[flag]:

cnt += 1

except:

pass

returncnt

trainDF['noun\_count'] = trainDF['text'].apply(lambda x: check\_pos\_tag(x, 'noun'))

trainDF['verb\_count'] = trainDF['text'].apply(lambda x: check\_pos\_tag(x, 'verb'))

trainDF['adj\_count'] = trainDF['text'].apply(lambda x: check\_pos\_tag(x, 'adj'))

trainDF['adv\_count'] = trainDF['text'].apply(lambda x: check\_pos\_tag(x, 'adv'))

trainDF['pron\_count'] = trainDF['text'].apply(lambda x: check\_pos\_tag(x, 'pron'))

deftrain\_model(classifier, feature\_vector\_train, label, feature\_vector\_valid, is\_neural\_net=False):

# fit the training dataset on the classifier

classifier.fit(feature\_vector\_train, label)

# predict the labels on validation dataset

predictions = classifier.predict(feature\_vector\_valid)

ifis\_neural\_net:

predictions = predictions.argmax(axis=-1)

returnmetrics.accuracy\_score(predictions, valid\_y)

# Naive Bayes on Count Vectors

accuracy = train\_model(naive\_bayes.MultinomialNB(), xtrain\_count, train\_y, xvalid\_count)

print "NB, Count Vectors: ", accuracy

# Naive Bayes on Word Level TF IDF Vectors

accuracy = train\_model(naive\_bayes.MultinomialNB(), xtrain\_tfidf, train\_y, xvalid\_tfidf)

print "NB, WordLevel TF-IDF: ", accuracy

# Naive Bayes on Ngram Level TF IDF Vectors

accuracy = train\_model(naive\_bayes.MultinomialNB(), xtrain\_tfidf\_ngram, train\_y, xvalid\_tfidf\_ngram)

print "NB, N-Gram Vectors: ", accuracy

# Naive Bayes on Character Level TF IDF Vectors

accuracy = train\_model(naive\_bayes.MultinomialNB(), xtrain\_tfidf\_ngram\_chars, train\_y, xvalid\_tfidf\_ngram\_chars)

print "NB, CharLevel Vectors: ", accuracy

Output

NB, Count Vectors: 0.7004

NB, WordLevel TF-IDF: 0.7024

NB, N-Gram Vectors: 0.5344

NB, CharLevel Vectors: 0.6872

https://www.analyticsvidhya.com/blog/2018/04/a-comprehensive-guide-to-understand-and-implement-text-classification-in-python/