

## APPLICATION-BASED PROBLEM

# 1

# Neural Network with a User-Defined Sigmoid Function

Randomly generate 100 linearly-spaced points between -10 and 10. Apply neural networks on it. Define your own sigmoid function.

```
import numpy as np
feature_set = np.array([[0,1,0],[0,0,1],[1,0,0],[1,1,0],[1,1,1]])
labels = np.array([[1,0,0,1,1]])
labels = labels.reshape(5,1)
np.random.seed(42)
weights = np.random.rand(3,1)
bias = np.random.rand(1)
# initialize learning rate to 0.05
lr = 0.05
# use activation function as the sigmoid function
def sigmoid(x):
    return 1/(1+np.exp(-x))
# calculate the derivative of the sigmoid function
def sigmoid_der(x):
    return sigmoid(x)*(1-sigmoid(x))
# epoch is the number of times we want to train the algorithm on our data
for epoch in range(20):
    inputs = feature_set
    # feedforward step1

    XW = np.dot(feature_set, weights) + bias

    #feedforward step2
    z = sigmoid(XW)

    # backpropagation step 1
    error = z - labels
    print(error.sum())

    # backpropagation step 2
    dcost_dpred = error
    dpred_dz = sigmoid_der(z)
```

Copyright © 2023 by McGraw Hill Education (India) Private Limited

```

z_delta = dcost_dpred * dpred_dz

inputs = feature_set.T
weights -= lr * np.dot(inputs, z_delta)

for num in z_delta:
    bias -= lr * num
# find dot product of input and weight vector and add bias to it
XW = np.dot(feature_set, weights) + bias
# pass the dot product through the sigmoid activation function
z = sigmoid(XW)
# z contains predicted output, now find error
error = z - labels
dcost_dpred = error
#d_pred is the sigmoid function and is differentiated with respect to input
dot product z
dpred_dz = sigmoid_der(z)
# calculate the final derivative of the cost function with respect to any
weight
z_delta = dcost_dpred * dpred_dz
inputs = feature_set.T
weights -= lr * np.dot(inputs, z_delta)
single_point = np.array([1,0,0])
result = sigmoid(np.dot(single_point, weights) + bias)
print('Predicting for value [1,0,0] : ',result)
single_point = np.array([0,1,0])
result = sigmoid(np.dot(single_point, weights) + bias)
print('Predicting for value [0,1,0] : ',result)

```

**OUTPUT**

```

1.1484765089981492
1.1370374057501034
1.1255946731475135
1.1141521191251875
1.1027135368122778
1.0912826991625884
1.0798633536714135
1.0684592171946765
1.057073970885539
1.0457112552629333
1.0343746654256973
1.0230677464251738
1.011793988808234
1.0005568243417682
0.9893596219286931
0.97820568372455
0.9670982414627042

```

```
0.9560404529951445  
0.9450353990547999  
0.9340860802442471  
Predicting for value [1,0,0] : [0.65211075]  
Predicting for value [0,1,0] : [0.80332672]
```





Copyright © 2023 by McGraw Hill Education (India) Private Limited

## APPLICATION-BASED PROBLEM

2

# Logistic Regression on Titanic Data Set

```
import numpy as np
import pandas as pd
from sklearn import preprocessing
import matplotlib.pyplot as plt
import seaborn as sns

# Read CSV train data file into DataFrame
train_df = pd.read_csv("titanic_train.csv")

# Read CSV test data file into DataFrame
test_df = pd.read_csv("titanic_test.csv")

# preview train data
print(train_df.head())
print('Number of samples in training data set is {}'.format(train_df.shape[0]))

# preview test data
print(test_df.head())
print('Number of samples in test data set is {}'.format(test_df.shape[0]))

# check missing values in train data
print(train_df.isnull().sum())

# percent of missing "Age"
print('Percent of missing "Age" records is %.2f%%' %((train_df['Age'].isnull().sum()/train_df.shape[0])*100))

# calculate mean age
print('The mean of "Age" is %.2f' %(train_df["Age"].mean(skipna=True)))

# median age
print('The median of "Age" is %.2f' %(train_df["Age"].median(skipna=True)))
```

Copyright © 2023 by McGraw Hill Education (India) Private Limited

```

# percent of missing "Cabin"
print('Percent of missing "Cabin" records is %.2f%%' %((train_df['Cabin'].
isnull().sum()/train_df.shape[0])*100)

# percent of missing "Embarked"
print('Percent of missing "Embarked" records is %.2f%%'
%((train_df['Embarked'].isnull().sum()/train_df.shape[0])*100))
print('Boarded passengers grouped by port of embarkation (C = Cherbourg, Q =
Queenstown, S = Southampton):')
print(train_df['Embarked'].value_counts())
sns.countplot(x='Embarked', data=train_df, palette='Set2')
plt.show()
print('The most common boarding port of embarkation is %s.' %train_
df['Embarked'].value_counts().idxmax())

print(''' For age, we will use median value, for Embarked, we will use S. We
will ignore Cabin as it has many null values ''')

train_data = train_df.copy()
train_data["Age"].fillna(train_df["Age"].median(skipna=True), inplace=True)
train_data["Embarked"].fillna(train_df['Embarked'].value_counts().idxmax(),
inplace=True)
train_data.drop('Cabin', axis=1, inplace=True)

# check missing values in adjusted train data
print(train_data.isnull().sum())

# preview adjusted train data
print(train_data.head())

print("SibSp and Parch relate to traveling with family, so we can combine
these variables into one categorical predictor stating if the passenger was
travelling alone or not. ")

# Creating categorical variable for traveling alone

train_data['TravelAlone'] = np.where((train_data["SibSp"]+train_
data["Parch"])>0, 0, 1)
train_data.drop('SibSp', axis=1, inplace=True)
train_data.drop('Parch', axis=1, inplace=True)

#create categorical variables and drop some variables
training=pd.get_dummies(train_data, columns=["Pclass","Embarked","Sex"])
training.drop('Sex_female', axis=1, inplace=True)
training.drop('PassengerId', axis=1, inplace=True)
training.drop('Name', axis=1, inplace=True)
training.drop('Ticket', axis=1, inplace=True)

```

```

final_train = training
print(final_train.head())

print("Following the same steps on the Training Data Set...")

test_data = test_df.copy()
test_data["Age"].fillna(train_df["Age"].median(skipna=True), inplace=True)
test_data["Fare"].fillna(train_df["Fare"].median(skipna=True), inplace=True)
test_data.drop('Cabin', axis=1, inplace=True)

test_data['TravelAlone']=np.where((test_data["SibSp"]+test_
    data["Parch"])>0, 0, 1)

test_data.drop('SibSp', axis=1, inplace=True)
test_data.drop('Parch', axis=1, inplace=True)

testing = pd.get_dummies(test_data, columns=["Pclass","Embarked","Sex"])
testing.drop('Sex_female', axis=1, inplace=True)
testing.drop('PassengerId', axis=1, inplace=True)
testing.drop('Name', axis=1, inplace=True)
testing.drop('Ticket', axis=1, inplace=True)

final_test = testing
print(final_test.head())

print("Adding Categorical Variable- Minor")

final_train['IsMinor']=np.where(final_train['Age']<=16, 1, 0)

final_test['IsMinor']=np.where(final_test['Age']<=16, 1, 0)

from sklearn.linear_model import LogisticRegression
from sklearn.feature_selection import RFE
from sklearn.model_selection import train_test_split, cross_val_score

cols = ["Age","Fare","TravelAlone","Pclass_1","Pclass_2","Embarked_C","Emba
rked_S","Sex_male","IsMinor"]
X = final_train[cols]
y = final_train['Survived']

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
    random_state=2)
# Build a logreg and compute the feature importances
logreg = LogisticRegression()

logreg.fit(X_train, y_train)
y_pred = logreg.predict(X_test)

```

```

scores_accuracy = cross_val_score(logreg, X, y, cv=10, scoring='accuracy')
scores_log_loss = cross_val_score(logreg, X, y, cv=10, scoring='neg_log_loss')
scores_auc = cross_val_score(logreg, X, y, cv=10, scoring='roc_auc')

print('K-fold cross-validation results:')
print(logreg.__class__.__name__+" average accuracy is %2.3f" % scores_accuracy.mean())
print(logreg.__class__.__name__+" average log_loss is %2.3f" % -scores_log_loss.mean())
print(logreg.__class__.__name__+" average auc is %2.3f" % scores_auc.mean())

```

**OUTPUT**

	PassengerId	Survived	Pclass	...	Fare	Cabin	Embarked
0	1	0	3	...	7.2500	NaN	S
1	2	1	1	...	71.2833	C85	C
2	3	1	3	...	7.9250	NaN	S
3	4	1	1	...	53.1000	C123	S
4	5	0	3	...	8.0500	NaN	S

[5 rows x 12 columns]

Number of samples in training data set is 891.

	PassengerId	Pclass	...	Cabin	Embarked
0	892	3	...	NaN	Q
1	893	3	...	NaN	S
2	894	2	...	NaN	Q
3	895	3	...	NaN	S
4	896	3	...	NaN	S

[5 rows x 11 columns]

Number of samples in test data set is 418.

PassengerId	0
Survived	0
Pclass	0
Name	0
Sex	0
Age	177
SibSp	0
Parch	0
Ticket	0
Fare	0
Cabin	687
Embarked	2

dtype: int64

Percent of missing "Age" records is 19.87%

The mean of "Age" is 29.70



```

The median of "Age" is 28.00
Percent of missing "Cabin" records is 77.10%
Percent of missing "Embarked" records is 0.22%
Boarded passengers grouped by port of embarkation (C = Cherbourg, Q =
Queenstown, S = Southampton):
S      644
C      168
Q       77
Name: Embarked, dtype: int64
The most common boarding port of embarkation is S.
For age, we will use median value and for Embarked, we will use S. We will
ignore Cabin as it has many null values.

```

```

PassengerId      0
Survived          0
Pclass           0
Name             0
Sex              0
Age              0
SibSp            0
Parch            0
Ticket           0
Fare             0
Embarked         0
dtype: int64

```

	PassengerId	Survived	Pclass	...	Ticket	Fare	Embarked
0	1	0	3	...	A/5 21171	7.2500	S
1	2	1	1	...	PC 17599	71.2833	C
2	3	1	3	...	STON/O2. 3101282	7.9250	S
3	4	1	1	...	113803	53.1000	S
4	5	0	3	...	373450	8.0500	S

```
[5 rows x 11 columns]
```

SibSp and Parch relate to traveling with family, so we can combine these variables into one categorical predictor stating if the passenger was travelling alone or not.

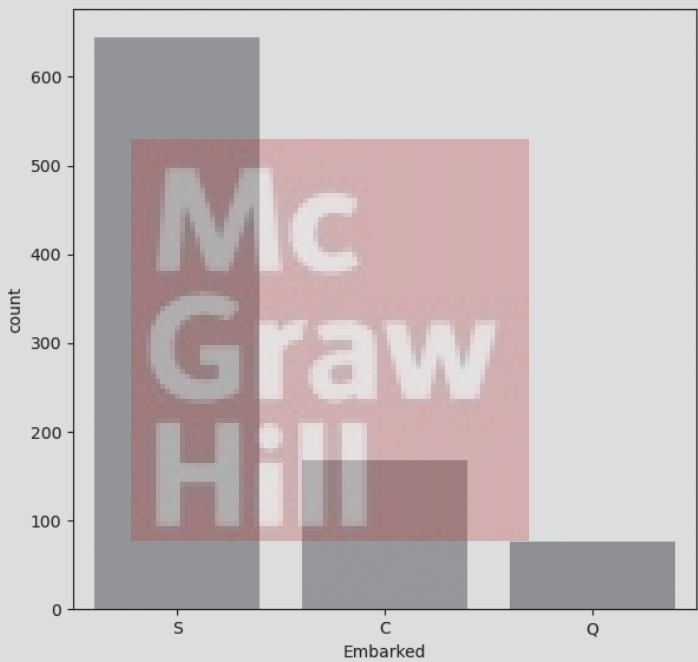
	Survived	Age	Fare	...	Embarked_Q	Embarked_S	Sex_male
0	0	22.0	7.2500	...	0	1	1
1	1	38.0	71.2833	...	0	0	0
2	1	26.0	7.9250	...	0	1	0
3	1	35.0	53.1000	...	0	1	0
4	0	35.0	8.0500	...	0	1	1

```
[5 rows x 11 columns]
```

Following the same steps on the Training Data Set...

	Age	Fare	TravelAlone	...	Embarked_Q	Embarked_S	Sex_male
0	34.5	7.8292	1	...	1	0	1
1	47.0	7.0000	0	...	0	1	0
2	62.0	9.6875	1	...	1	0	1
3	27.0	8.6625	1	...	0	1	1
4	22.0	12.2875	0	...	0	1	0

```
[5 rows x 10 columns]
Adding Categorical Variable- Minor
K-fold cross-validation results:
LogisticRegression average accuracy is 0.799
LogisticRegression average log_loss is 0.455
LogisticRegression average auc is 0.849
```



## APPLICATION-BASED PROBLEM

3

# Spam Detection Technique Using Naïve Bayes Algorithm

Download the SMS dataset from the link <https://archive.ics.uci.edu/ml/machine-learning-databases/00228/> and use Naïve Bayes algorithm to detect spam messages. The dataset has 5,574 messages in English. These messages are already tagged as ham (legitimate) or spam.

```
import pandas as pd
import nltk
from nltk.stem import PorterStemmer
df = pd.read_table('SMS_Spam_Collection', sep = '\t', header = None,
names = ['LABEL', 'SMS'])
print(df)
#convert labels from strings to binary values for classifier
df['LABEL'] = df.LABEL.map({'ham': 0, 'spam': 1})
# convert all characters in the message to lower case:
df['SMS'] = df.SMS.map(lambda x: x.lower())
# remove any punctuation
df['SMS'] = df.SMS.str.replace('[^\w\s]', '')
# tokenize the messages into single words
df['SMS'] = df['SMS'].apply(nltk.word_tokenize)
#perform some word stemming.
stemmer = PorterStemmer()
df['SMS'] = df['SMS'].apply(lambda x: [stemmer.stem(y) for y in x])
#transform the data into occurrences, which will be the features that we
will feed into our model:
from sklearn.feature_extraction.text import CountVectorizer
# This converts the list of words into space-separated strings
df['SMS'] = df['SMS'].apply(lambda x: ' '.join(x))
count_vect = CountVectorizer()
counts = count_vect.fit_transform(df['SMS'])
from sklearn.feature_extraction.text import TfidfTransformer
transformer = TfidfTransformer().fit(counts)
counts = transformer.transform(counts)
```

Copyright © 2023 by McGraw Hill Education (India) Private Limited

```
#split data into training and test sets:
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(counts, df['LABEL'],
test_size=0.1, random_state=30)
#initialize the Naive Bayes Classifier and fit the data
from sklearn.naive_bayes import MultinomialNB
model = MultinomialNB().fit(X_train, y_train)
# Evaluate the Model
import numpy as np
predicted = model.predict(X_test)
print(np.mean(predicted == y_test))
#print the confusion matrix
from sklearn.metrics import confusion_matrix
print(confusion_matrix(y_test, predicted))
```

**OUTPUT**

```
      LABEL      SMS
0      ham  Go until jurong point, crazy.. Available only ...
1      ham      Ok lar... Joking wif u oni...
2  spam  Free entry in 2 a wkly comp to win FA Cup fina...
3      ham  U dun say so early hor... U c already then say...
4      ham  Nah I don't think he goes to usf, he lives aro...
...      ...      ...
5567 spam  This is the 2nd time we have tried 2 contact u...
5568 ham      Will ü b going to esplanade fr home?
5569 ham  Pity, * was in mood for that. So...any other s...
5570 ham  The guy did some bitching but I acted like i'd...
5571 ham      Rofl. Its true to its name
[5572 rows x 2 columns]
0.9354838709677419
[[486   0]
 [ 36 36]]
```

# Data Analysis Using Pandas

Download phone\_data.csv from Kaggle.com and analyze the dataset as directed below.

```
import pandas as pd
import dateutil
# Load data from csv file
data = pd.read_csv('phone_data.csv')
# Convert date from string to date times
data['date'] = data['date'].apply(dateutil.parser.parse, dayfirst=True)
# Number of rows in the dataset
print('How many rows the dataset: ', data['item'].count() )
# Longest phone call / data entry
print('What was the longest phone call: ', data['duration'].max() )
# Number of seconds of phone calls recorded in total
print('How many seconds of phone calls are recorded in total: ', data['duration']
[data['item'] == 'call'].sum() )
# Number of non-null unique network entries
print('Number of non-null unique network entries: ', data['network'].nunique() )
# Number of entries for each month
print('How many entries are there for each month: ')
print( data['month'].value_counts() )
# Get the first entry for each month
print( data.groupby('month').first() )
# Sum of the durations per month
print( data.groupby('month')['duration'].sum() )
# Number of dates / entries in each month
print( data.groupby('month')['date'].count() )
# Sum of durations, for calls only, to each network
print( data[data['item'] == 'call'].groupby('network')['duration'].sum() )
# Number of calls, sms, and data entries in each month
print( data.groupby(['month', 'item'])['date'].count() )
# Number of calls, texts, and data sent per month, split by network_type
print( data.groupby(['month', 'network_type'])['date'].count() )
```

**OUTPUT**

How many rows the dataset: 830

What was the longest phone call: 10528.0

How many seconds of phone calls are recorded in total: 92321.0

Number of non-null unique network entries: 9

How many entries are there for each month:

2014-11 230

2015-01 205

2014-12 157

2015-02 137

2015-03 101

Name: month, dtype: int64

	index	date	duration	item	network	network_type
month						
2014-11	0	2014-10-15 06:58:00	34.429	data	data	data
2014-12	228	2014-11-13 06:58:00	34.429	data	data	data
2015-01	381	2014-12-13 06:58:00	34.429	data	data	data
2015-02	577	2015-01-13 06:58:00	34.429	data	data	data
2015-03	729	2015-02-12 20:15:00	69.000	call	landline	landline

month

2014-11 26639.441

2014-12 14641.870

2015-01 18223.299

2015-02 15522.299

2015-03 22750.441

Name: duration, dtype: float64

month

2014-11 230

2014-12 157

2015-01 205

2015-02 137

2015-03 101

Name: date, dtype: int64

network

Meteor 7200.0

Tesco 13828.0

Three 36464.0

Vodafone 14621.0

landline 18433.0

voicemail 1775.0

Name: duration, dtype: float64

month item

2014-11 call 107

data 29

sms 94

```

2014-12  call      79
         data      30
         sms       48
2015-01  call      88
         data      31
         sms       86
2015-02  call      67
         data      31
         sms       39
2015-03  call      47
         data      29
         sms       25

```

```
Name: date, dtype: int64
```

```

month    network_type
2014-11  data          29
         landline       5
         mobile       189
         special        1
         voicemail      6
2014-12  data          30
         landline       7
         mobile       108
         voicemail      8
         world          4
2015-01  data          31
         landline      11
         mobile       160
         voicemail      3
2015-02  data          31
         landline       8
         mobile       90
         special        2
         voicemail      6
2015-03  data          29
         landline      11
         mobile       54
         voicemail      4
         world          3

```

```
Name: date, dtype: int64
```



Copyright © 2023 by McGraw Hill Education (India) Private Limited



# Support Vector Machine Algorithm (SVM)

Write the code to visualize the SVM algorithm using the `meshgrid()` and contour plots.

The `numpy.meshgrid` function creates a rectangular grid using two one-dimensional arrays representing the Cartesian indexing or Matrix indexing. The function returns two 2D arrays representing the X and Y coordinates of all the points.

Contour plots also known as Level Plots are used to display a 3D surface on a 2D plane. For this, it plots two predictor variables X Y on the y-axis and a response variable Z as contours. Contour plots help users to visualize how the value of Z changes with respect to two inputs X and Y. That is,  $Z = f(X,Y)$ . A contour line of the function of two variables is drawn. This line is basically a curve along which the function has a constant value.

The **`contourf()` function** defined in the `pyplot` module of `matplotlib` library is used to plot contours. It accepts the following parameters:

**X, Y** that represent the coordinates of the values in Z.

**Z** is the height values over which the contour is drawn.

**levels** is used to determine the numbers and positions of the contour lines/regions.

```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
dataset = pd.read_csv('Social_Network_Ads.csv')
X = dataset.iloc[:, [2, 3]].values
y = dataset.iloc[:, 4].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)
from sklearn.preprocessing import StandardScaler
sc = StandardScaler()
X_train = sc.fit_transform(X_train)
X_test = sc.transform(X_test)
from sklearn.svm import SVC
classifier = SVC(kernel = 'rbf', random_state = 0)
```

```

classifier.fit(X_train, y_train)
y_pred = classifier.predict(X_test)
from sklearn.metrics import confusion_matrix, accuracy_score
cm = confusion_matrix(y_test, y_pred)
print(cm)
accuracy_score(y_test, y_pred)
from matplotlib.colors import ListedColormap
X_set, y_set = X_test, y_test
X1, X2 = np.meshgrid(np.arange(start = X_set[:, 0].min() - 1, stop = X_set[:, 0].max() + 1, step = 0.01),
                     np.arange(start = X_set[:, 1].min() - 1, stop = X_set[:, 1].max() + 1, step = 0.01))
plt.contourf(X1, X2, classifier.predict(np.array([X1.ravel(), X2.ravel()]).T).reshape(X1.shape),
             alpha = 0.75, cmap = ListedColormap(('red', 'green')))
plt.xlim(X1.min(), X1.max())
plt.ylim(X2.min(), X2.max())
for i, j in enumerate(np.unique(y_set)):
    plt.scatter(X_set[y_set == j, 0], X_set[y_set == j, 1],
                c = ListedColormap(('red', 'green'))(i), label = j)
plt.title('SVM (Test set)')
plt.xlabel('Age')
plt.ylabel('Estimated Salary')
plt.legend()
plt.show()

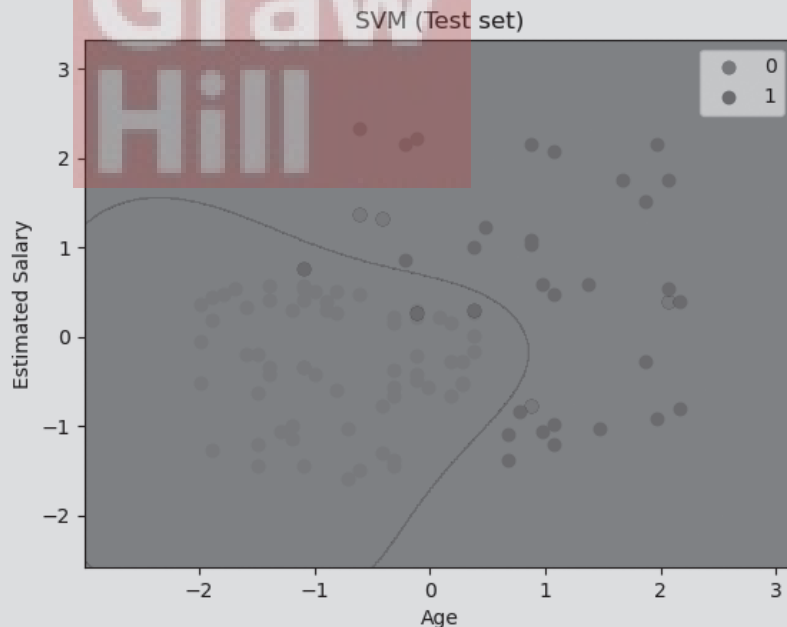
```

**OUTPUT**

```

[[64  4]
 [ 3 29]]

```



From the above graph, it is evident that there are in total 7 incorrect predictions, three green (Yes) predictions were predicted as Red (No) and four Red (No) predictions were predicted as yes in green color. The overall accuracy was 93%.

Write a code that predicts Diabetes in a patient using SVM algorithm. Print the accuracy. Also compare the accuracies obtained using at least three other algorithms.

```
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score
import matplotlib.pyplot as plt

df = pd.read_csv('diabetes.csv')
print(df.head())
x = df.drop('Outcome',axis=1)
y = df['Outcome']
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.25,
random_state=42)

SupportVectorClassModel = SVC()
SupportVectorClassModel.fit(x_train,y_train)
y_pred = SupportVectorClassModel.predict(x_test)
accuracy = accuracy_score(y_test,y_pred)*100
print(accuracy)
```

#### OUTPUT

Pregnancies	Glucose	BloodPressure	...	DiabetesPedigreeFunction	Age	Outcome
0	6	148	72	... 0.627	50	1
1	1	85	66	... 0.351	31	0
2	8	183	64	... 0.672	32	1
3	1	89	66	... 0.167	21	0
4	0	137	40	... 2.288	33	1

[5 rows x 9 columns]  
72.91666666666666

#### Comparing accuracy using other algorithms

```
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score
import matplotlib.pyplot as plt
df = pd.read_csv('diabetes.csv')
print(df.head())
x = df.drop('Outcome',axis=1)
y = df['Outcome']
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.25,
random_state=42)

import seaborn as sns
sns.countplot(df['Outcome'],label="Count")
plt.show()
```

```

from sklearn.neighbors import KNeighborsClassifier
training_accuracy = []
test_accuracy = []
# try n_neighbors from 1 to 10
neighbors_settings = range(1, 11)
for n_neighbors in neighbors_settings:
    # build the model
    knn = KNeighborsClassifier(n_neighbors=n_neighbors)
    knn.fit(x_train, y_train)
    # record training set accuracy
    training_accuracy.append(knn.score(x_train, y_train))
    # record test set accuracy
    test_accuracy.append(knn.score(x_test, y_test))
plt.plot(neighbors_settings, training_accuracy, label="training accuracy")
plt.plot(neighbors_settings, test_accuracy, label="test accuracy")
plt.ylabel("Accuracy")
plt.xlabel("n_neighbors")
plt.legend()
plt.show()
knn = KNeighborsClassifier(n_neighbors=9)
knn.fit(x_train, y_train)
print('Accuracy of K-NN classifier on training set: {:.2f}'.format(knn.score
(x_train, y_train)))
print('Accuracy of K-NN classifier on test set: {:.2f}'.format(knn.score(x_test,
y_test)))

from sklearn.linear_model import LogisticRegression
logreg = LogisticRegression().fit(x_train, y_train)
print("Logistic Regression Accuracy on Training set score: {:.3f}".format(logreg.
score(x_train, y_train)))
print("Logistic Regression Accuracy on Test set score: {:.3f}".format(logreg.
score(x_test, y_test)))

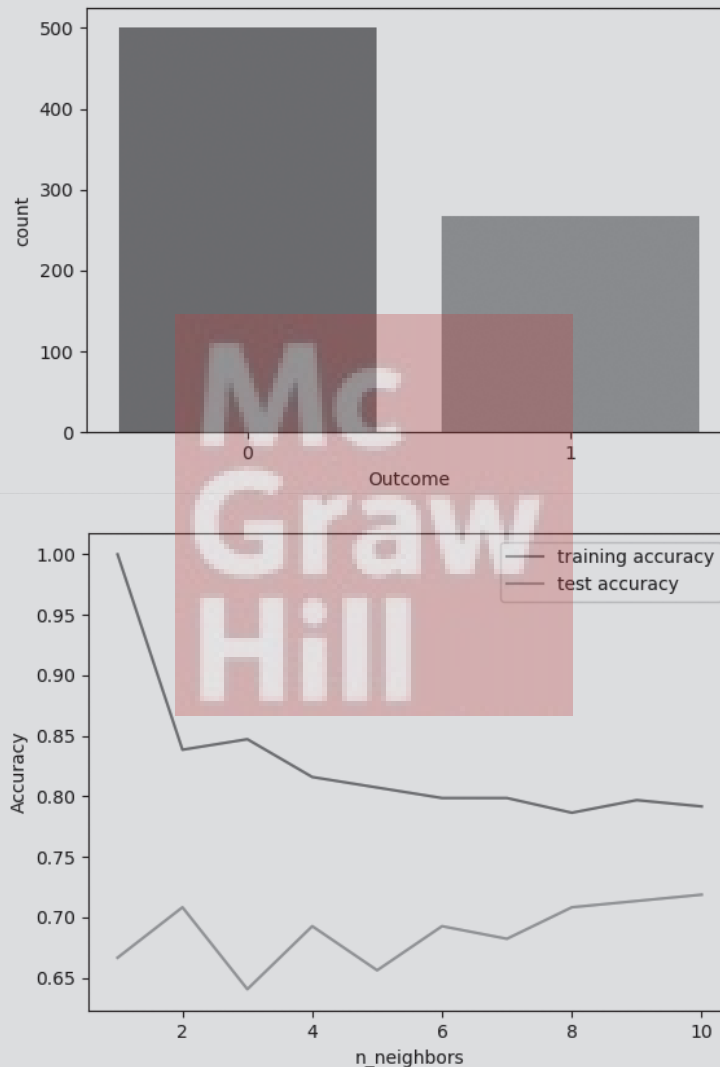
from sklearn.tree import DecisionTreeClassifier
tree = DecisionTreeClassifier(random_state=0)
tree.fit(x_train, y_train)
print("Decision Tree Accuracy on training set: {:.3f}".format(tree.score(x_train,
y_train)))
print("Decision Tree Accuracy on test set: {:.3f}".format(tree.score(x_test,
y_test)))
from sklearn.ensemble import RandomForestClassifier
rf = RandomForestClassifier(n_estimators=100, random_state=0)
rf.fit(x_train, y_train)
print("Random Forest Accuracy on training set: {:.3f}".format(rf.score(x_train,
y_train)))
print("Random Forest Accuracy on test set: {:.3f}".format(rf.score(x_test,
y_test)))

```

**OUTPUT**

Pregnancies	Glucose	BloodPressure	...	DiabetesPedigreeFunction	Age	Outcome	
0	6	148	72	...	0.627	50	1
1	1	85	66	...	0.351	31	0
2	8	183	64	...	0.672	32	1
3	1	89	66	...	0.167	21	0
4	0	137	40	...	2.288	33	1

[5 rows x 9 columns]

**Feature importance in decision trees**

Feature importance rates how important each feature is for the decision tree. Its value varies from 0 to 1 for each feature, where 0 means that feature is not at all used and 1 means that it perfectly predicts the value of the target variable. Feature importance of all the variables always sum to 1. We can display the feature importance of all variables by writing,

Copyright © 2023 by McGraw Hill Education (India) Private Limited

```
print("Feature importances:\n{}".format(tree.feature_importances_))
```

**Write a code to extract important features of the diabetes.csv dataset.**

```
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score
import matplotlib.pyplot as plt

df = pd.read_csv('diabetes.csv')
print(df.head())

x = df.drop('Outcome',axis=1)
y = df['Outcome']

diabetes_features = x.columns
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.25,
random_state=42)

from sklearn.tree import DecisionTreeClassifier
tree = DecisionTreeClassifier(random_state=0)
tree.fit(x_train, y_train)
print("Decision Tree Accuracy on training set: {:.3f}".format(tree.score(x_train,
y_train)))
print("Decision Tree Accuracy on test set: {:.3f}".format(tree.score(x_test,
y_test)))

def plot_feature_importances_diabetes(model):
    plt.figure(figsize=(8,6))
    n_features = 8
    plt.barh(range(n_features), model.feature_importances_, align='center')
    plt.yticks(np.arange(n_features), diabetes_features)
    plt.xlabel("Feature importance")
    plt.ylabel("Feature")
    plt.ylim(-1, n_features)
    plt.show()
plot_feature_importances_diabetes(tree)
```

#### OUTPUT

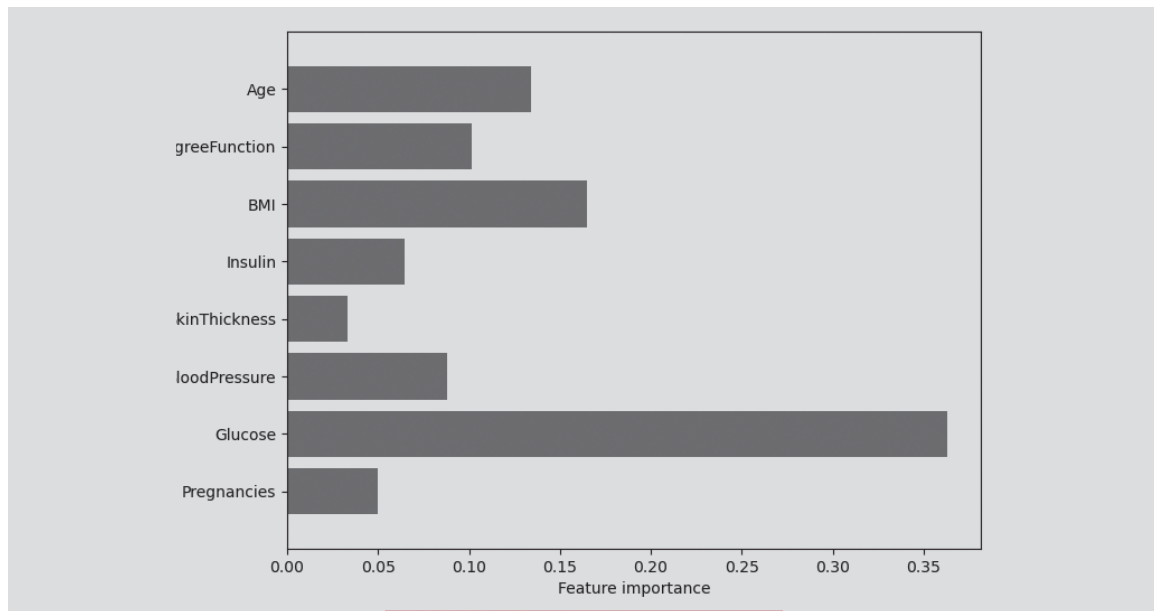
Pregnancies	Glucose	BloodPressure	...	DiabetesPedigreeFunction	Age	Outcome
0	6	148	72 ...	0.627	50	1
1	1	85	66 ...	0.351	31	0
2	8	183	64 ...	0.672	32	1
3	1	89	66 ...	0.167	21	0
4	0	137	40 ...	2.288	33	1

```
[5 rows x 9 columns]
```

```
Decision Tree Accuracy on training set: 1.000
```

```
Decision Tree Accuracy on test set: 0.688
```

Copyright © 2023 by McGraw Hill Education (India) Private Limited



Mc  
Graw  
Hill



Copyright © 2023 by McGraw Hill Education (India) Private Limited