Project File

Machine Learning and Artificial Intelligence

Topic-Classification of mushrooms as edible or poisonous based their different features

EAEPCO9

ECAM – 1

Submitted To –

Ms. Kavya Gupta

Submitted by –

Aditya Anand

Roll number – 2020UEA6546

Acknowledgement

In successfully completing this project, many people have I would like to express my special thanks of gratitude to my teacher **Ms Kavya Gupta** and our University **Netaji Subhas University of technology** (NSUT) who gave me the golden opportunity to do this project on the topic MUSHROOM CLASSIFICATION. It helped me in doing a lot of Research and i came to know about a lot of things related to this topic.
Finally, I would also like to thank my parents and friends who helped me a lot in finalising this project within the limited time frame.

Abstract

Predicting if a set of mushrooms is edible or not corresponds to the task of classifying them into two groups—edible or poisonous—on the basis of a classification rule. To support this binary task, I have collected the largest and most comprehensive attribute based data available. In the work, we detail the creation, curation and simulation of a data set for binary classification. We evaluated different machine learning algorithms, namely, naive Bayes, logistic regression,KNN, Random forest, decision tree and stochastic gradient method. The data set contains 23 families and is the largest available. I further provide a fully reproducible workflow and provide the data under the FAIR principles.

Methodology

Libraries Used:

- Sklearn
- Pandas
- Matplotlib
- Seaborn
- Numpy

Classification Models Used:

- Naïve-Bayes
- Logistic regression
- KNN
- Random forest
- Decision tree
- Stochastic gradient method

<u>PROJECT ON THE CLASSIFICATION OF MUSHROOMS INTO EDIBLE OR POISONOUS</u>

IMPORTING NECESSARY LIBRARIES

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sn
from sklearn import preprocessing,tree
from sklearn.metrics import
roc_curve,accuracy_score,confusion_matrix,classification_report,ro
c_auc_score
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.naive_bayes import GaussianNB
from sklearn.neighbors import KNeighborsClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn import feature_selection
%matplotlib inline
```

```
df=pd.read_csv("mushroom_data.csv")
df.shape
```

```
(61069, 21)
```

```
df.head()
```

class	cap-diameter	cap-shape	cap-surface	cap-color	does-bruise-or-bleed	gill-attachment	gill-spacing	gill-color	stem-height	...	stem-root	stem-surface	stem-color	veil-type	veil-color	has-ring	ring-type	spore-print-color
p	15.26	x	g	o	f	e	NaN	w	16.95	...	s	y	w	u	w	t	g	NaN
p	16.60	x	g	o	f	e	NaN	w	17.99	...	s	y	w	u	w	t	g	NaN
p	14.07	x	g	o	f	e	NaN	w	17.80	...	s	y	w	u	w	t	g	NaN
p	14.17	f	h	e	f	e	NaN	w	15.77	...	s	y	w	u	w	t	p	NaN
p	14.64	x	h	o	f	e	NaN	w	16.53	...	s	y	w	u	w	t	p	NaN

Converting String/Char Values to Int

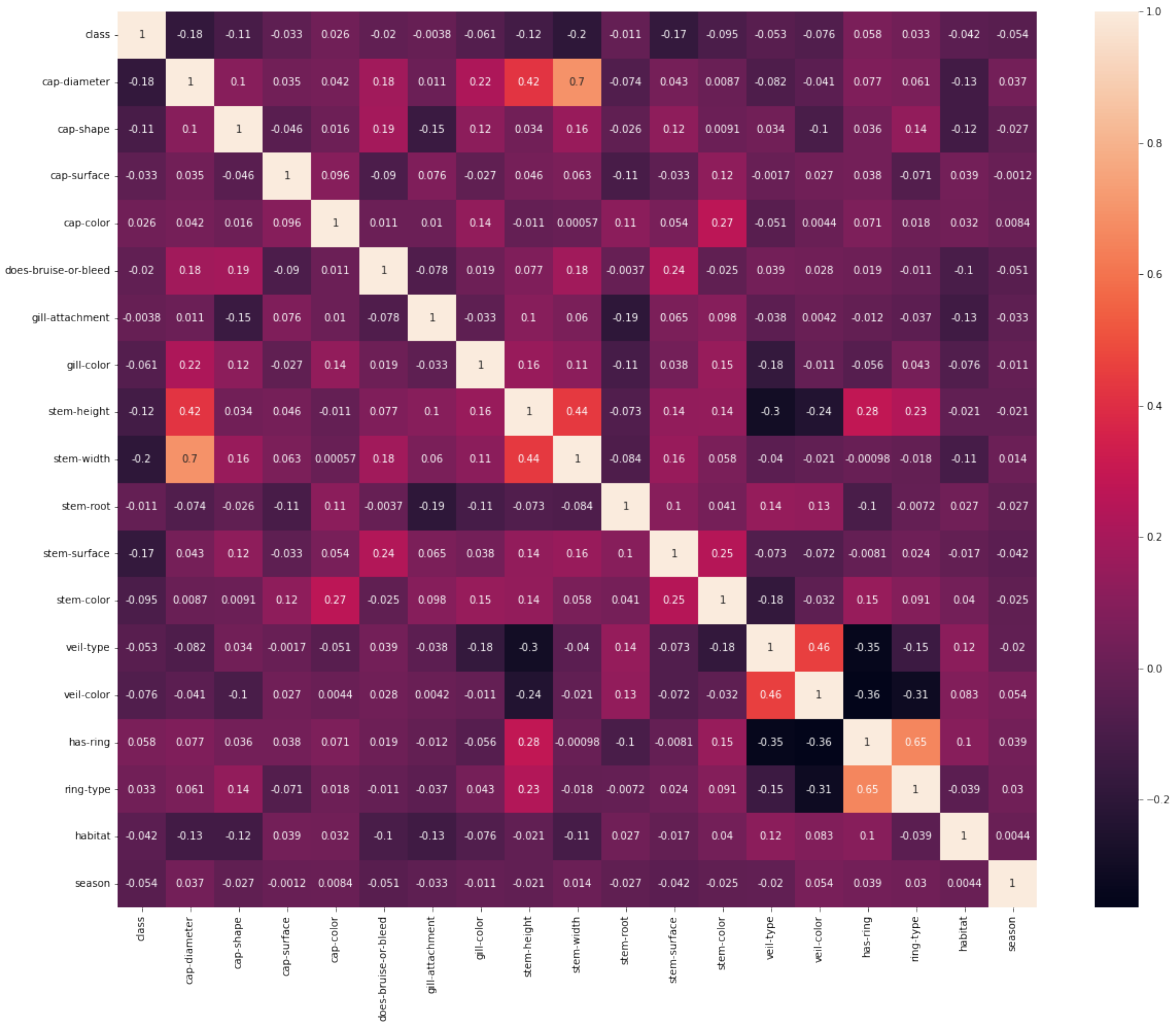
```
le_cap_shape=preprocessing.LabelEncoder()
le_cap_shape.fit(df['cap-shape'])
df['cap-shape']=le_cap_shape.transform(df['cap-shape'])

le_cap_surface=preprocessing.LabelEncoder()
le_cap_surface.fit(df['cap-surface'])
df['cap-surface']=le_cap_surface.transform(df['cap-surface'])

le_cap_color=preprocessing.LabelEncoder()
le_cap_color.fit(df['cap-color'])
df['cap-color']=le_cap_color.transform(df['cap-color'])

le_gill_color=preprocessing.LabelEncoder()
le_gill_color.fit(df['gill-color'])
df['gill-color']=le_gill_color.transform(df['gill-color'])

le_stem_color=preprocessing.LabelEncoder()
le_stem_color.fit(df['stem-color'])
df['stem-color']=le_stem_color.transform(df['stem-color'])

le_has_ring=preprocessing.LabelEncoder()
le_has_ring.fit(df['has-ring'])
df['has-ring']=le_has_ring.transform(df['has-ring'])

le_habitat=preprocessing.LabelEncoder()
le_habitat.fit(df['habitat'])
df['habitat']=le_habitat.transform(df['habitat'])

le_class=preprocessing.LabelEncoder()
le_class.fit(df['class'])
df['class']=le_class.transform(df['class'])

le_does_bruise=preprocessing.LabelEncoder()
le_does_bruise.fit(df['does-bruise-or-bleed'])
df['does-bruise-or-bleed']=le_does_bruise.transform(df['does-
bruise-or-bleed'])

le_gill_attachment=preprocessing.LabelEncoder()
le_gill_attachment.fit(df['gill-attachment'])
df['gill-attachment']=le_gill_attachment.transform(df['gill-
attachment'])

le_stem_root=preprocessing.LabelEncoder()
le_stem_root.fit(df['stem-root'])
df['stem-root']=le_stem_root.transform(df['stem-root'])
```

```
le_stem_surface=preprocessing.LabelEncoder()
le_stem_surface.fit(df['stem-surface'])
df['stem-surface']=le_stem_surface.transform(df['stem-surface'])

le_veil_type=preprocessing.LabelEncoder()
le_veil_type.fit(df['veil-type'])
df['veil-type']=le_veil_type.transform(df['veil-type'])

le_veil_color=preprocessing.LabelEncoder()
le_veil_color.fit(df['veil-color'])
df['veil-color']=le_veil_color.transform(df['veil-color'])

le_ring_type=preprocessing.LabelEncoder()
le_ring_type.fit(df['ring-type'])
df['ring-type']=le_ring_type.transform(df['ring-type'])

le_season=preprocessing.LabelEncoder()
le_season.fit(df['season'])
df['season']=le_season.transform(df['season'])

df=df.drop(['gill-spacing','spore-print-color'],axis=1)
df.head()
```

class	cap-diameter	cap-shape	cap-surface	cap-color	does-bruise-or-bleed	gill-attachment	gill-color	stem-height	stem-width	stem-root	stem-surface	stem-color	veil-type	veil-color	has-ring	ring-type	habitat	season
1	15.26	6	2	6	0	2	10	16.95	17.09	4	7	11	0	4	1	2	0	3
1	16.60	6	2	6	0	2	10	17.99	18.19	4	7	11	0	4	1	2	0	2
1	14.07	6	2	6	0	2	10	17.80	17.74	4	7	11	0	4	1	2	0	3
1	14.17	2	3	1	0	2	10	15.77	15.98	4	7	11	0	4	1	5	0	3
1	14.64	6	3	6	0	2	10	16.53	17.20	4	7	11	0	4	1	5	0	3

Visualizing the correlation between all the features

```
f, ax = plt.subplots(figsize=(20, 16))
corr = df.corr()
sn.heatmap(corr, annot=True)
plt.show()
```



Splitting Dataset

```
xdf=df.copy()
xdf.drop(['class'],axis=1,inplace=True)
```

```
ydf=df['class'].copy()
ydf
```

```
X_train, X_test,Y_train,Y_test =
train_test_split(xdf,ydf,test_size=0.2,random_state=0)
```

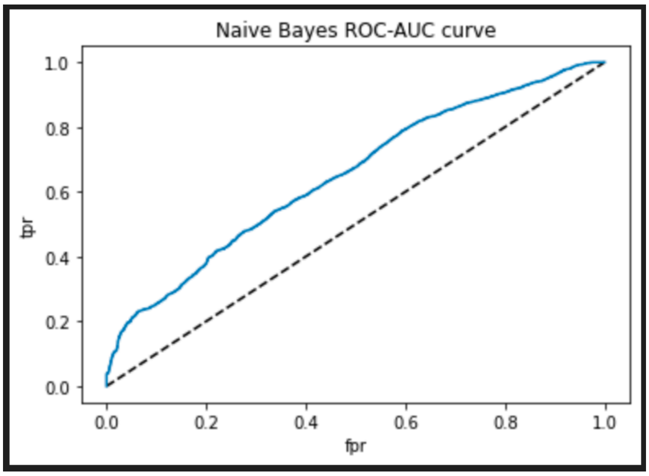
Naive Bayes Model

```
NB_model = GaussianNB()
NB_model.fit(X_train,Y_train)
Y_hat_NB=NB_model.predict(X_test)
print("Accuracy : {}
%".format(accuracy_score(Y_test,Y_hat_NB)*100))
print("Train Accuracy : {}
%".format(accuracy_score(Y_train,NB_model.predict(X_train))*100))
print(classification_report(Y_test,Y_hat_NB))
```

```
Accuracy : 59.56279679056819%
Train Accuracy : 60.04707808822024%
              precision    recall  f1-score   support

           0       0.53      0.59      0.56      5302
           1       0.66      0.60      0.63      6912

    accuracy                           0.60     12214
   macro avg       0.59      0.60      0.59     12214
weighted avg       0.60      0.60      0.60     12214
```

ROC-AUC curve

```
y_pred_proba = NB_model.predict_proba(X_test)[:,1]
fpr, tpr, thresholds = roc_curve(Y_test, y_pred_proba)
plt.plot([0,1],[0,1],'k--')
plt.plot(fpr,tpr)
plt.xlabel('fpr')
plt.ylabel('tpr')
plt.title('Naive Bayes ROC-AUC curve')
plt.show()
```



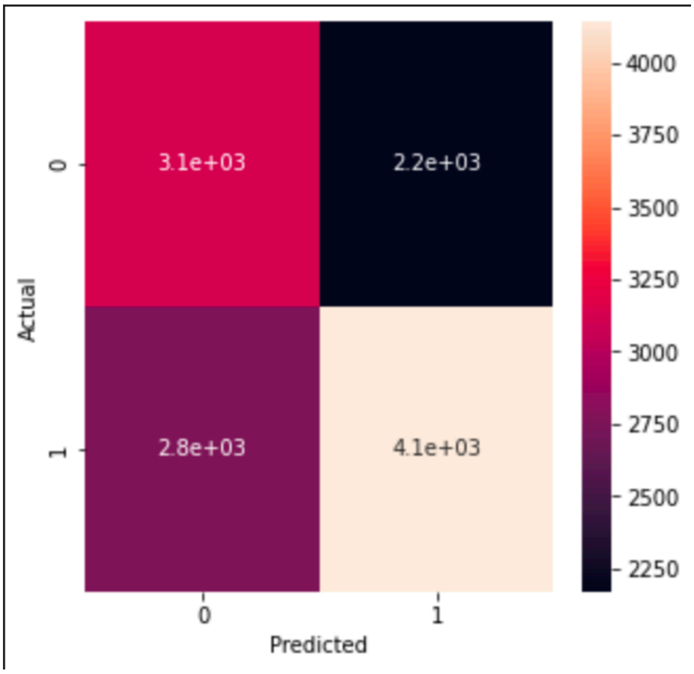
ROC-AUC Score

```
roc_auc_score(Y_test,y_pred_proba)
```

```
0.6492078952124984
```

Confusion matrix

```
cm=confusion_matrix(Y_test,Y_hat_NB)
plt.figure(figsize=(5,5))
sn.heatmap(cm,annot=True)
plt.xlabel('Predicted')
plt.ylabel('Actual')
```



Logistic Regression Model

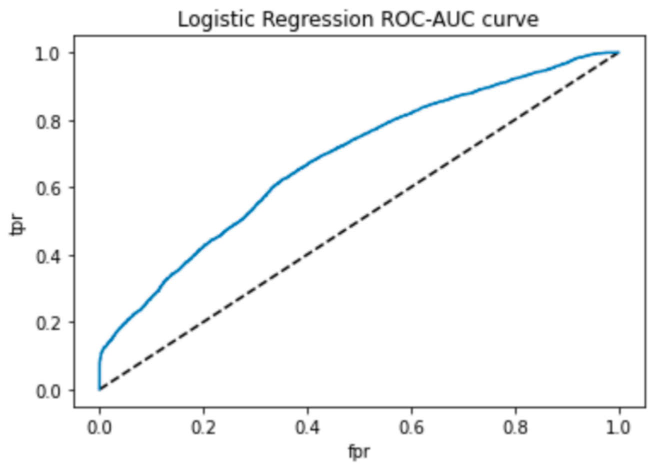
```
LR_model = LogisticRegression(C=0.01,solver='sag')
LR_model.fit(X_train,Y_train)
Y_hat_LR=LR_model.predict(X_test)
print("Accuracy : {}
%".format(accuracy_score(Y_test,Y_hat_LR)*100))
print("Train Accuracy : {}
%".format(accuracy_score(Y_train,LR_model.predict(X_train))*100))
```

```
print(classification_report(Y_test,Y_hat_LR))
```

```
Accuracy : 64.09857540527264%
Train Accuracy : 64.13673114317879%
              precision    recall  f1-score   support

           0       0.61      0.49      0.54      5302
           1       0.66      0.76      0.70      6912

    accuracy                           0.64     12214
   macro avg       0.63      0.62      0.62     12214
weighted avg       0.64      0.64      0.63     12214
```

ROC-AUC curve

```
y_pred_proba = LR_model.predict_proba(X_test)[:,1]
fpr, tpr, thresholds = roc_curve(Y_test, y_pred_proba)
plt.plot([0,1],[0,1],'k--')
plt.plot(fpr,tpr)
plt.xlabel('fpr')
plt.ylabel('tpr')
plt.title('Logistic Regression ROC-AUC curve')
plt.show()
```



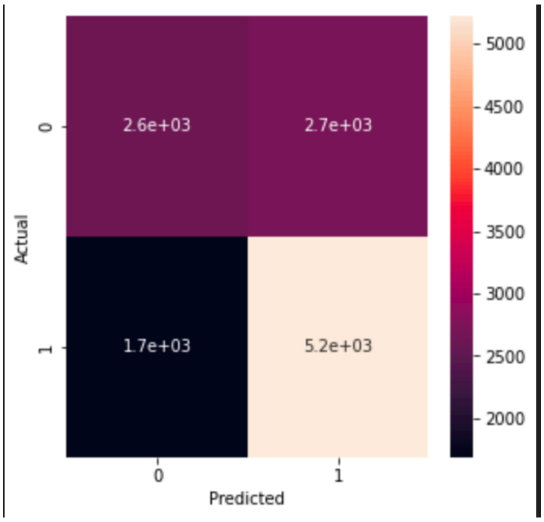
ROC-AUC Score

```
roc_auc_score(Y_test,y_pred_proba)
```

```
0.6815160596280929
```

Confusion matrix

```
cm=confusion_matrix(Y_test,Y_hat_LR)
plt.figure(figsize=(5,5))
sn.heatmap(cm,annot=True)
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.show()
```

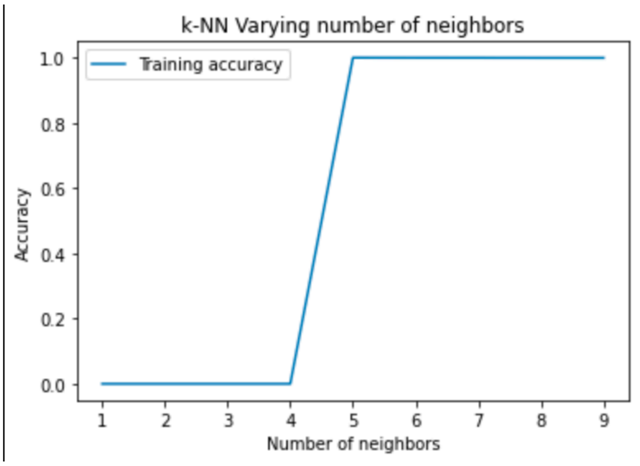


KNN Model

```
Ks=100
train_accuracy =np.zeros((Ks-1))
test_accuracy = np.zeros((Ks-1))
mean_acc=np.zeros((Ks-1))
for n in range(5,Ks):
    knn=KNeighborsClassifier(n_neighbors=n)
    knn.fit(X_train,Y_train)
    Y_hat_knn=knn.predict(X_test)
    mean_acc[n-1]=accuracy_score(Y_test,Y_hat_knn)
    train_accuracy[n-1] = knn.score(X_train, Y_train)
    test_accuracy[n-1] = knn.score(X_test, Y_test)
print("Max accuracy is: ",mean_acc.max(),"with 
k=",mean_acc.argmax()+1)
```

```
Max accuracy is:  0.9988537743572949 with k= 5
```

```
empty= np.arange(1,10)
plt.title('k-NN Varying number of neighbors')
plt.plot(empty, train_accuracy, label='Training accuracy')
plt.legend()
plt.xlabel('Number of neighbors')
plt.ylabel('Accuracy')
plt.show()
```

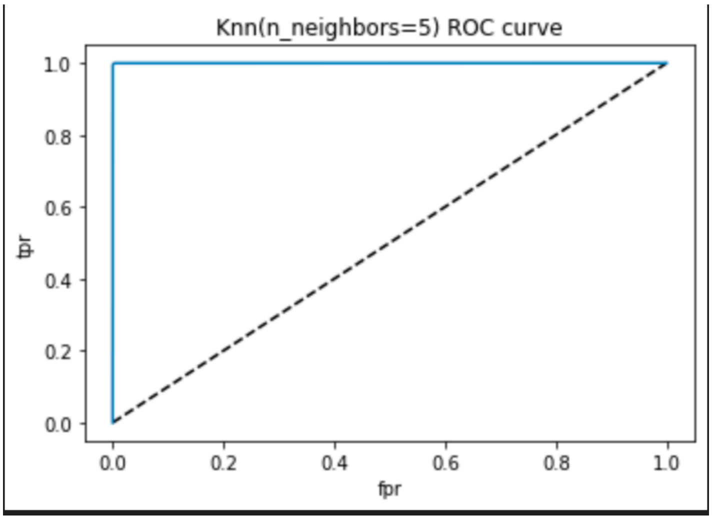



```
KNN_model=KNeighborsClassifier(n_neighbors=mean_acc.argmax()+1)
KNN_model.fit(X_train,Y_train)
Y_hat_KNN=KNN_model.predict(X_test)
print("Accuracy : {}
%".format(accuracy_score(Y_test,Y_hat_KNN)*100))
print("Train Accuracy : {}
%".format(accuracy_score(Y_train,KNN_model.predict(X_train))*100))
print(classification_report(Y_test,Y_hat_KNN))
```

```
Accuracy : 99.8853774357295%
Train Accuracy : 99.94678129157711%
              precision    recall  f1-score   support

           0       1.00      1.00      1.00      5302
           1       1.00      1.00      1.00      6912

    accuracy                           1.00     12214
   macro avg       1.00      1.00      1.00     12214
weighted avg       1.00      1.00      1.00     12214
```

ROC-AUC curve

```
y_pred_proba = knn.predict_proba(X_test)[:,1]
fpr, tpr, thresholds = roc_curve(Y_test, y_pred_proba)
plt.plot([0,1],[0,1],'k--')
plt.plot(fpr,tpr, label='Knn')
plt.xlabel('fpr')
plt.ylabel('tpr')
plt.title('Knn(n_neighbors=5) ROC-AUC curve')
plt.show()
```



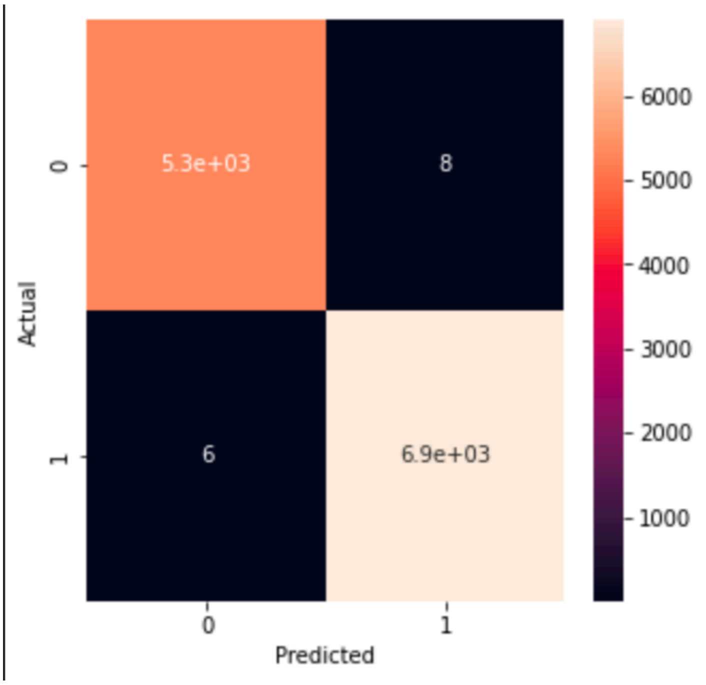
ROC-AUC Score

```
roc_auc_score(Y_test,y_pred_proba)
```

```
0.9998268909705631
```

Confusion matrix

```
cm=confusion_matrix(Y_test,Y_hat_KNN)
plt.figure(figsize=(5,5))
sn.heatmap(cm,annot=True)
plt.xlabel('Predicted')
plt.ylabel('Actual')
```



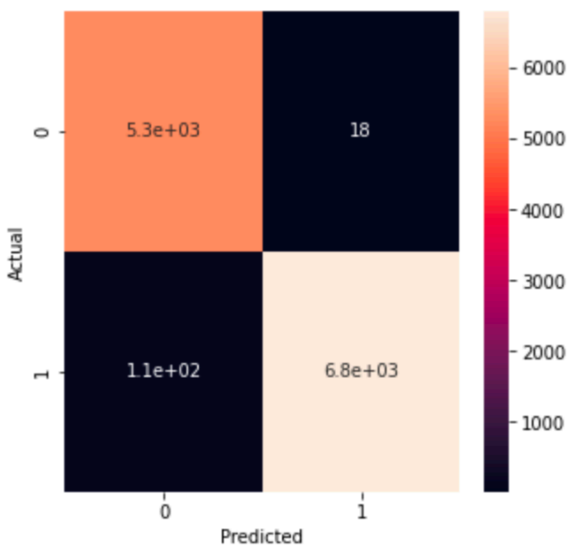
Random Forest Model

```
RF_model = 
RandomForestClassifier(n_estimators=100,max_depth=10,random_state=
0)
RF_model.fit(X_train,Y_train)
Y_hat_RF=RF_model.predict(X_test)
print("Accuracy : {}
%".format(accuracy_score(Y_test,Y_hat_RF)*100))
print("Train Accuracy : {}
%".format(accuracy_score(Y_train,RF_model.predict(X_train))*100))
print(classification_report(Y_test,Y_hat_RF))
```

```
Accuracy : 98.97658424758474%
Train Accuracy : 99.09528195681098%
              precision    recall  f1-score   support

           0       0.98      1.00      0.99      5302
           1       1.00      0.98      0.99      6912

    accuracy                           0.99     12214
   macro avg       0.99      0.99      0.99     12214
weighted avg       0.99      0.99      0.99     12214
```

Confusion matrix

```
cm=confusion_matrix(Y_test,Y_hat_RF)
plt.figure(figsize=(5,5))
sn.heatmap(cm,annot=True)
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.show()
```



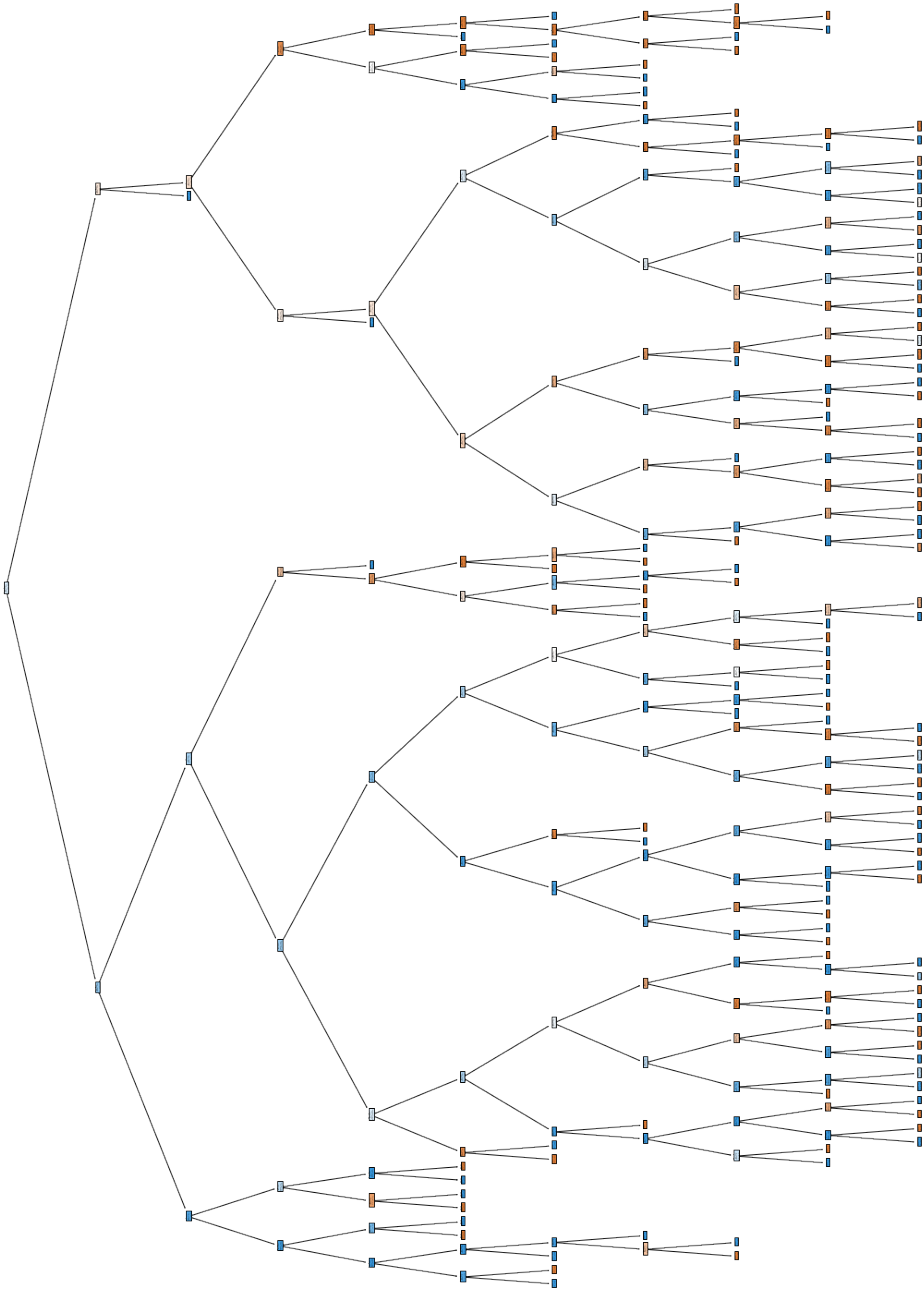
Decision Tree

```
DT_model = DecisionTreeClassifier(max_depth=10,random_state=0)
DT_model.fit(X_train,Y_train)
Y_hat_DT=DT_model.predict(X_test)
print("Accuracy : {}
%".format(accuracy_score(Y_test,Y_hat_DT)*100))
print("Train Accuracy : {}
%".format(accuracy_score(Y_train,DT_model.predict(X_train))*100))
print(classification_report(Y_test,Y_hat_DT))
```

```
Accuracy : 95.82446373014574%
Train Accuracy : 96.0802374373145%
              precision    recall  f1-score   support

           0       0.95      0.95      0.95      5302
           1       0.96      0.96      0.96      6912

    accuracy                           0.96     12214
   macro avg       0.96      0.96      0.96     12214
weighted avg       0.96      0.96      0.96     12214
```

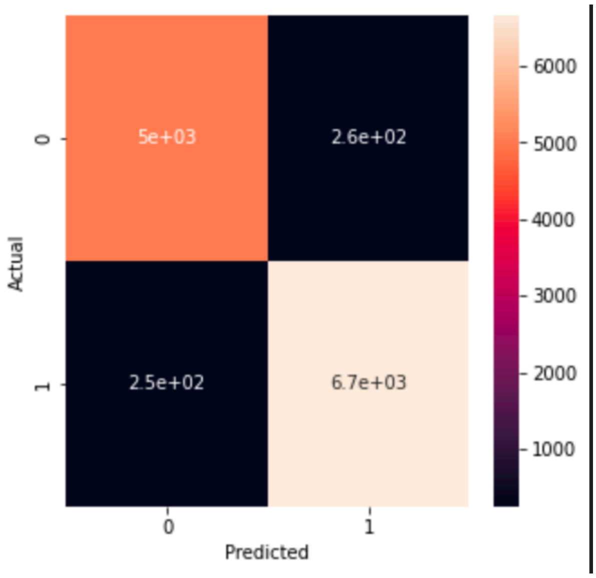
Visualizing the Decision Tree

```
feature_names=df.columns[:-1]
fig=plt.figure(figsize=(25,20))
_=tree.plot_tree(DT_model,feature_names=feature_names,filled=True)
fig.savefig('DT.png')
plt.show()
```



Confusion matrix

```
cm=confusion_matrix(Y_test,Y_hat_DT)
plt.figure(figsize=(5,5))
sn.heatmap(cm,annot=True)
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.show()
```



Stochastic Gradient Descent Model

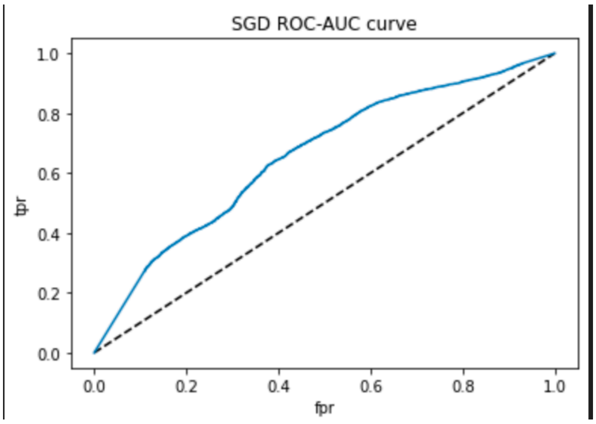
```
from sklearn.linear_model import SGDClassifier
sgd_model= SGDClassifier(loss= 'modified_huber', shuffle=True,
random_state=1)
sgd_model.fit(X_train, Y_train)
Y_hat_SGD=sgd_model.predict(X_test)
```

```
print("Accuracy : {}
%".format(accuracy_score(Y_test,Y_hat_SGD)*100))
print("Train Accuracy : {}
%".format(accuracy_score(Y_train,sgd_model.predict(X_train))*100))
print(classification_report(Y_test,Y_hat_SGD))
```

```
Accuracy : 63.97576551498281%
Train Accuracy : 63.10306007573432%
              precision    recall  f1-score   support

           0       0.64      0.38      0.48      5302
           1       0.64      0.84      0.72      6912

    accuracy                           0.64     12214
   macro avg       0.64      0.61      0.60     12214
weighted avg       0.64      0.64      0.62     12214
```

ROC-AUC curve

```
y_pred_proba = sgd_model.predict_proba(X_test)[:,1]
fpr, tpr, thresholds = roc_curve(Y_test, y_pred_proba)
plt.plot([0,1],[0,1],'k--')
plt.plot(fpr,tpr)
plt.xlabel('fpr')
plt.ylabel('tpr')
plt.title('SGD ROC-AUC curve')
plt.show()
```



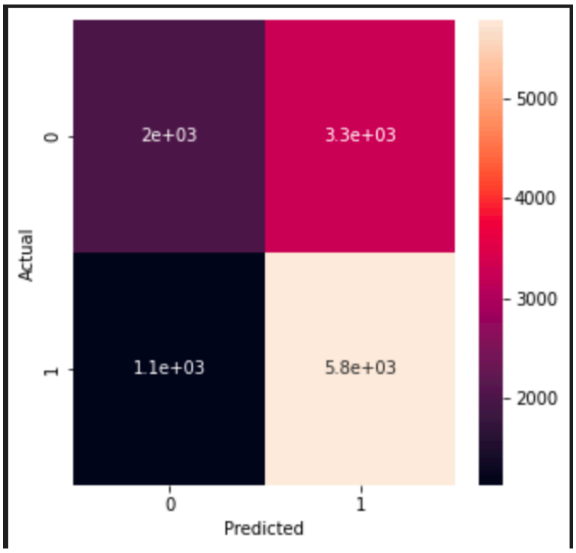
ROC-AUC Score

```
roc_auc_score(Y_test,y_pred_proba)
```

```
0.6576904013771883
```

Confusion matrix

```
cm=confusion_matrix(Y_test,Y_hat_SGD)
plt.figure(figsize=(5,5))
sn.heatmap(cm,annot=True)
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.show()
```



Final Accuracies of All the Models

```
data = [['Naive Bayes',
accuracy_score(Y_train,NB_model.predict(X_train))*100,accuracy_sco
re(Y_test,Y_hat_NB)*100], ['Logistic
Regression',accuracy_score(Y_train,LR_model.predict(X_train))*100,
accuracy_score(Y_test,Y_hat_LR)*100 ],
['KNN',accuracy_score(Y_train,KNN_model.predict(X_train))*100,accu
racy_score(Y_test,Y_hat_KNN)*100 ],['Random
Forest',accuracy_score(Y_train,RF_model.predict(X_train))*100,accu
racy_score(Y_test,Y_hat_RF)*100],['Decision
Tree',accuracy_score(Y_train,DT_model.predict(X_train))*100,accura
cy_score(Y_test,Y_hat_DT)*100],['SGD Method',
accuracy_score(Y_train,sgd_model.predict(X_train))*100,accuracy_sc
ore(Y_test,Y_hat_SGD)*100]]

accuracy_matrix = pd.DataFrame(data, columns = ['Algo Used',
'Train Accuracy','Test Accuracy'],index=['1','2','3','4','5','6'])

accuracy_matrix
```

	Algo Used	Train Accuracy	Test Accuracy
1	Naive Bayes	60.047078	59.562797
2	Logistic Regression	64.136731	64.098575
3	KNN	99.946781	99.885377
4	Random Forest	99.095282	98.976584
5	Decision Tree	96.080237	95.824464
6	SGD Method	63.103060	63.975766

By Looking at the above table it is clear that KNN has the best accuracy for the given data set

Conclusion

To classify the same data, we use different machine learning models and they give different accuracies. Using **Naive Bayes**, we received an accuracy of 60.05% on training data and 59.56% on testing data. Using **Logistic Regression**, we received an accuracy of 64.14% on training data and 64.1% on testing data.

For **Random forest** we received accuracy of 99.1% for training data and 98.98% for testing data. For **KNN** classifier it was 99.95% and 99.88% respectively.

Decision Tree was able to classify the training data with a 96.08% accuracy and the testing data with 95.82% accuracy while for **SGD** it was 63.1% and 63.98% respectively.

Hence, we conclude that the best classifiers for the given dataset and parameters are **<u>Random Forest</u>** and **<u>KNN</u>**.

References

1) https://archive.ics.uci.edu/ml/datasets/Secondary+Mushroom+Dataset - Dataset from UCI

2) https://www.ncbi.nlm.nih.gov/pmc/articles/PMC8046754/ - Mushroom data creation, curation, and simulation to support classification tasks-Reference Paper

3) Product of bachelor thesis at Philipps-UniversitÃ¤t Marburg, Bioinformatics Division, supervised by Dr. G. Hattab -Research Work for collecting and creation of the dataset