In [1]:

```
import pandas as pd
```

In [3]:

```
housing = pd.read_csv("/Users/moizzah/Desktop/housing/housing.csv")
```

The usual missing value imputation routine please

In [4]:

```
housing_df = housing[['housing_median_age', 'total_rooms',
       'total_bedrooms', 'population', 'households', 'median_income',
       'median_house_value','ocean_proximity']].copy()
median = housing_df['total_bedrooms'].median()
housing_df['total_bedrooms'].fillna(median, inplace = True)
```

Label Encoding

instead of mapping the categorical variable
use label encoder from sklearn

In [5]:

```
from sklearn import preprocessing
```

In [6]:

```
lab_enc = preprocessing.LabelEncoder()
```

In [7]:

```
lab_enc.fit(housing['ocean_proximity'].unique())
```

Out[7]:

```
LabelEncoder()
```

In [23]:

```
housing_df['ocean_proximity'].unique()
```

Out[23]:

```
array(['NEAR BAY', '<1H OCEAN', 'INLAND', 'NEAR OCEAN', 'ISLAND'],
      dtype=object)
```

In [9]:

```
list(lab_enc.classes_)
```

Out[9]:

```
['<1H OCEAN', 'INLAND', 'ISLAND', 'NEAR BAY', 'NEAR OCEAN']
```

In [24]:

```
housing_df['ocean_proximity'].head()
```

Out[24]:

```
0    NEAR BAY
1    NEAR BAY
2    NEAR BAY
3    NEAR BAY
4    NEAR BAY
Name: ocean_proximity, dtype: object
```

In [25]:

```
housing_df['ocean_proximity'].tail()
```

Out[25]:

```
20635    INLAND
20636    INLAND
20637    INLAND
20638    INLAND
20639    INLAND
Name: ocean_proximity, dtype: object
```

In [27]:

```
housing_df['ocean_proximity'] = lab_enc.transform(housing_df['ocean_proximity'])
```

In [13]:

```
housing['ocean_proximity'].head()
```

Out[13]:

```
0    3
1    3
2    3
3    3
4    3
Name: ocean_proximity, dtype: int64
```

In [28]:

```
housing_df['ocean_proximity'].tail()
```

Out[28]:

```
20635    1
20636    1
20637    1
20638    1
20639    1
Name: ocean_proximity, dtype: int64
```

Now the usual scaling routine please

In [29]:

```
from sklearn.preprocessing import StandardScaler as ss
temp = housing_df[['housing_median_age', 'total_rooms',
       'total_bedrooms', 'population', 'households', 'median_income',
       'median_house_value']].copy()
temp = ss().fit_transform(temp)
housing_df[['housing_median_age', 'total_rooms',
       'total_bedrooms', 'population', 'households', 'median_income',
       'median_house_value']] = temp
housing_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 20640 entries, 0 to 20639
Data columns (total 8 columns):
housing_median_age    20640 non-null float64
total_rooms           20640 non-null float64
total_bedrooms        20640 non-null float64
population            20640 non-null float64
households            20640 non-null float64
median_income         20640 non-null float64
median_house_value    20640 non-null float64
ocean_proximity       20640 non-null int64
dtypes: float64(7), int64(1)
memory usage: 1.3 MB
```

Create the feature set
that is: train and test splits

note: Do not create separate dataframes from predictor variables and target variables
just provide the indices in train_Test_split method

In [16]:

```
from sklearn.model_selection import train_test_split
```

In [31]:

```
x_housing_train, x_housing_test, y_housing_train, y_housing_test = train_test_sp
lit(housing_df.iloc[:, [0,1,2,3,4,5,7]],

housing_df.iloc[:,[6]],

test_size = 0.3, random_state = 123)
```

now repeat for titanic dataset
remember to use label encoder

Also, try using pandas method drop to drop columns

In [43]:

```
titanic = pd.read_csv('/Users/moizzah/Desktop/titanic/titanic.csv')
titanic.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 891 entries, 0 to 890
Data columns (total 12 columns):
PassengerId    891 non-null int64
Survived       891 non-null int64
Pclass         891 non-null int64
Name           891 non-null object
Sex            891 non-null object
Age            714 non-null float64
SibSp          891 non-null int64
Parch          891 non-null int64
Ticket         891 non-null object
Fare           891 non-null float64
Cabin          204 non-null object
Embarked       889 non-null object
dtypes: float64(2), int64(5), object(5)
memory usage: 83.7+ KB
```

In [44]:

```
titanic.dropna(subset = ['Embarked'], inplace = True)
titanic = titanic.drop(['Cabin'], axis=1)
titanic.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 889 entries, 0 to 890
Data columns (total 11 columns):
PassengerId    889 non-null int64
Survived       889 non-null int64
Pclass         889 non-null int64
Name           889 non-null object
Sex            889 non-null object
Age            712 non-null float64
SibSp          889 non-null int64
Parch          889 non-null int64
Ticket         889 non-null object
Fare           889 non-null float64
Embarked       889 non-null object
dtypes: float64(2), int64(5), object(4)
memory usage: 83.3+ KB
```

In [45]:

```
mean_age = titanic['Age'].mean()
titanic['Age'].fillna(mean_age, inplace = True)
titanic.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 889 entries, 0 to 890
Data columns (total 11 columns):
PassengerId    889 non-null int64
Survived       889 non-null int64
Pclass         889 non-null int64
Name           889 non-null object
Sex            889 non-null object
Age            889 non-null float64
SibSp          889 non-null int64
Parch          889 non-null int64
Ticket         889 non-null object
Fare           889 non-null float64
Embarked       889 non-null object
dtypes: float64(2), int64(5), object(4)
memory usage: 83.3+ KB
```

In [48]:

```
lab_enc.fit(titanic['Sex'].unique())
```

Out[48]:

```
LabelEncoder()
```

In [49]:

```
list(lab_enc.classes_)
```

Out[49]:

```
['female', 'male']
```

In [51]:

```
titanic['Sex'] = lab_enc.transform(titanic['Sex'])
titanic.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 889 entries, 0 to 890
Data columns (total 11 columns):
PassengerId    889 non-null int64
Survived       889 non-null int64
Pclass         889 non-null int64
Name           889 non-null object
Sex            889 non-null int64
Age            889 non-null float64
SibSp          889 non-null int64
Parch          889 non-null int64
Ticket         889 non-null object
Fare           889 non-null float64
Embarked       889 non-null object
dtypes: float64(2), int64(6), object(3)
memory usage: 83.3+ KB
```

In [52]:

```
lab_enc.fit(titanic['Embarked'].unique())
```

Out[52]:

```
LabelEncoder()
```

In [53]:

```
list(lab_enc.classes_)
```

Out[53]:

```
['C', 'Q', 'S']
```

In [54]:

```
titanic['Embarked'] = lab_enc.transform(titanic['Embarked'])
titanic.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 889 entries, 0 to 890
Data columns (total 11 columns):
PassengerId    889 non-null int64
Survived       889 non-null int64
Pclass         889 non-null int64
Name           889 non-null object
Sex            889 non-null int64
Age            889 non-null float64
SibSp          889 non-null int64
Parch          889 non-null int64
Ticket         889 non-null object
Fare           889 non-null float64
Embarked       889 non-null int64
dtypes: float64(2), int64(7), object(2)
memory usage: 83.3+ KB
```

In [55]:

```
x_titanic_train, x_titanic_test, y_titanic_train, y_titanic_test = train_test_sp
lit(titanic.iloc[:, [2,4,5,6,7,10]],

titanic.iloc[:,[1]],

test_size = 0.3, random_state = 123)
```

Naive Bayes Classifiers

In [20]:

```
from sklearn.naive_bayes import MultinomialNB, GaussianNB, BernoulliNB
```

In [58]:

```
#x_titanic_train.info()
G_nb = GaussianNB().fit(x_titanic_train, y_titanic_train)
```

```
/Library/Frameworks/Python.framework/Versions/3.8/lib/python3.8/site
-packages/sklearn/utils/validation.py:724: DataConversionWarning: A
column-vector y was passed when a 1d array was expected. Please chan
ge the shape of y to (n_samples, ), for example using ravel().
  y = column_or_1d(y, warn=True)
```

In [59]:

```
G_nb.class_prior_
```

Out[59]:

```
array([0.62379421, 0.37620579])
```

In [60]:

```
G_nb.class_count_
```

Out[60]:

```
array([388., 234.])
```

In [61]:

```
G_nb.classes_
```

Out[61]:

```
array([0, 1])
```

In [63]:

```
G_nb_pred = G_nb.predict(x_titanic_test)
G_nb_pred
```

Out[63]:

```
array([0, 0, 1, 0, 0, 0, 0, 1, 0, 1, 0, 0, 1, 0, 0, 1, 0, 1, 0, 1,
1, 0,
       0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0,
1, 1,
       1, 1, 0, 0, 1, 0, 1, 0, 0, 0, 1, 1, 0, 0, 1, 0, 1, 1, 1, 1,
1, 1,
       0, 0, 0, 0, 1, 1, 0, 1, 0, 0, 1, 0, 1, 1, 1, 0, 0, 0, 1, 1,
0, 0,
       0, 0, 1, 1, 1, 0, 1, 0, 1, 0, 0, 1, 0, 1, 1, 0, 0, 1, 0, 0,
0, 1,
       1, 0, 1, 0, 0, 1, 1, 0, 1, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 0,
0, 0,
       0, 0, 1, 0, 1, 1, 0, 1, 0, 1, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0,
1, 0,
       0, 0, 1, 1, 1, 1, 1, 0, 0, 0, 0, 1, 1, 0, 1, 0, 1, 0, 1, 0,
1, 0,
       1, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 1, 1, 0, 1, 0,
1, 0,
       0, 0, 1, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0,
0, 0,
       0, 1, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 1,
0, 0,
       1, 0, 0, 0, 1, 0, 1, 0, 0, 1, 0, 1, 0, 1, 1, 0, 0, 0, 0, 1,
0, 1,
       0, 0, 1])
```

In [64]:

```
from sklearn.metrics import confusion_matrix, accuracy_score, precision_score, r
ecall_score, f1_score
```

In [65]:

```
confusion_matrix(y_titanic_test, G_nb_pred)
```

Out[65]:

```
array([[135,  26],
       [ 31,  75]])
```

In [66]:

```
accuracy_score(y_titanic_test, G_nb_pred)
```

Out[66]:

```
0.7865168539325843
```

In [67]:

```
precision_score(y_titanic_test, G_nb_pred)
```

Out[67]:

```
0.7425742574257426
```

In [68]:

```
recall_score(y_titanic_test, G_nb_pred)
```

Out[68]:

```
0.7075471698113207
```

In [69]:

```
f1_score(y_titanic_test, G_nb_pred)
```

Out[69]:

```
0.7246376811594202
```

Now, from the titanic dataset, pick only those predictor variables,
which can be used to train multinomial Naive Bayes
use the same train test split, make copies, add a suffix _mnb wherever appropriate
compare this mnb classifier with G_nb classifier

Bagging

First Apply on Decision Tree classifier

In [72]:

```
from sklearn.ensemble import BaggingClassifier
from sklearn.tree import DecisionTreeClassifier
```

base_estimator
The base estimator to fit on random subsets of the dataset. If None, then the base estimator is a decision tree.

In [73]:

```
tree_restricted = DecisionTreeClassifier(criterion = 'entropy', random_state = 1
23, max_depth = 4)
```

In [74]:

```
bagging = BaggingClassifier(tree_restricted, n_estimators=100, max_samples=0.8,
                        random_state=198)
```

In [75]:

```
bagging.fit(x_titanic_train, y_titanic_train)
```

```
/Library/Frameworks/Python.framework/Versions/3.8/lib/python3.8/site
-packages/sklearn/ensemble/bagging.py:623: DataConversionWarning: A
column-vector y was passed when a 1d array was expected. Please chan
ge the shape of y to (n_samples, ), for example using ravel().
  y = column_or_1d(y, warn=True)
```

Out[75]:

```
BaggingClassifier(base_estimator=DecisionTreeClassifier(class_weight
=None,
                                                        criterion='e
ntropy',
                                                        max_depth=4,
                                                        max_features
=None,
                                                        max_leaf_nod
es=None,
                                                        min_impurity
_decrease=0.0,
                                                        min_impurity
_split=None,
                                                        min_samples_
leaf=1,
                                                        min_samples_
split=2,
                                                        min_weight_f
raction_leaf=0.0,
                                                        presort=Fals
e,
                                                        random_state
=123,
                                                        splitter='be
st'),
                  bootstrap=True, bootstrap_features=False, max_feat
ures=1.0,
                  max_samples=0.8, n_estimators=100, n_jobs=None,
                  oob_score=False, random_state=198, verbose=0,
                  warm_start=False)
```

In [76]:

```
bagging.base_estimator_
```

Out[76]:

```
DecisionTreeClassifier(class_weight=None, criterion='entropy', max_d
epth=4,
                       max_features=None, max_leaf_nodes=None,
                       min_impurity_decrease=0.0, min_impurity_split
=None,
                       min_samples_leaf=1, min_samples_split=2,
                       min_weight_fraction_leaf=0.0, presort=False,
                       random_state=123, splitter='best')
```

In [77]:

```
bagging.n_features_
```

Out[77]:

```
6
```

In [78]:

```
bagging.estimators_
```

Out[78]:

```
[DecisionTreeClassifier(class_weight=None, criterion='entropy', max_
depth=4,
                        max_features=None, max_leaf_nodes=None,
                        min_impurity_decrease=0.0, min_impurity_spli
t=None,
                        min_samples_leaf=1, min_samples_split=2,
                        min_weight_fraction_leaf=0.0, presort=False,
                        random_state=1511094012, splitter='best'),
 DecisionTreeClassifier(class_weight=None, criterion='entropy', max_
depth=4,
                        max_features=None, max_leaf_nodes=None,
                        min_impurity_decrease=0.0, min_impurity_spli
t=None,
                        min_samples_leaf=1, min_samples_split=2,
                        min_weight_fraction_leaf=0.0, presort=False,
                        random_state=1492618134, splitter='best'),
 DecisionTreeClassifier(class_weight=None, criterion='entropy', max_
depth=4,
                        max_features=None, max_leaf_nodes=None,
                        min_impurity_decrease=0.0, min_impurity_spli
t=None,
                        min_samples_leaf=1, min_samples_split=2,
                        min_weight_fraction_leaf=0.0, presort=False,
                        random_state=1489790253, splitter='best'),
 DecisionTreeClassifier(class_weight=None, criterion='entropy', max_
depth=4,
                        max_features=None, max_leaf_nodes=None,
                        min_impurity_decrease=0.0, min_impurity_spli
t=None,
                        min_samples_leaf=1, min_samples_split=2,
                        min_weight_fraction_leaf=0.0, presort=False,
                        random_state=215994538, splitter='best'),
 DecisionTreeClassifier(class_weight=None, criterion='entropy', max_
depth=4,
                        max_features=None, max_leaf_nodes=None,
                        min_impurity_decrease=0.0, min_impurity_spli
t=None,
                        min_samples_leaf=1, min_samples_split=2,
                        min_weight_fraction_leaf=0.0, presort=False,
                        random_state=1049357520, splitter='best'),
 DecisionTreeClassifier(class_weight=None, criterion='entropy', max_
depth=4,
                        max_features=None, max_leaf_nodes=None,
                        min_impurity_decrease=0.0, min_impurity_spli
t=None,
                        min_samples_leaf=1, min_samples_split=2,
                        min_weight_fraction_leaf=0.0, presort=False,
                        random_state=1086753958, splitter='best'),
 DecisionTreeClassifier(class_weight=None, criterion='entropy', max_
depth=4,
                        max_features=None, max_leaf_nodes=None,
                        min_impurity_decrease=0.0, min_impurity_spli
t=None,
                        min_samples_leaf=1, min_samples_split=2,
                        min_weight_fraction_leaf=0.0, presort=False,
                        random_state=631037180, splitter='best'),
 DecisionTreeClassifier(class_weight=None, criterion='entropy', max_
depth=4,
                        max_features=None, max_leaf_nodes=None,
                        min_impurity_decrease=0.0, min_impurity_spli
t=None,
                        min_samples_leaf=1, min_samples_split=2,
                        min_weight_fraction_leaf=0.0, presort=False,
                        random_state=1137172324, splitter='best'),
 DecisionTreeClassifier(class_weight=None, criterion='entropy', max_
depth=4,
                        max_features=None, max_leaf_nodes=None,
                        min_impurity_decrease=0.0, min_impurity_spli
t=None,
                        min_samples_leaf=1, min_samples_split=2,
                        min_weight_fraction_leaf=0.0, presort=False,
                        random_state=1491004505, splitter='best'),
 DecisionTreeClassifier(class_weight=None, criterion='entropy', max_
depth=4,
                        max_features=None, max_leaf_nodes=None,
                        min_impurity_decrease=0.0, min_impurity_spli
t=None,
                        min_samples_leaf=1, min_samples_split=2,
                        min_weight_fraction_leaf=0.0, presort=False,
                        random_state=231221994, splitter='best'),
 DecisionTreeClassifier(class_weight=None, criterion='entropy', max_
depth=4,
                        max_features=None, max_leaf_nodes=None,
                        min_impurity_decrease=0.0, min_impurity_spli
t=None,
                        min_samples_leaf=1, min_samples_split=2,
                        min_weight_fraction_leaf=0.0, presort=False,
                        random_state=1152921957, splitter='best'),
 DecisionTreeClassifier(class_weight=None, criterion='entropy', max_
depth=4,
                        max_features=None, max_leaf_nodes=None,
                        min_impurity_decrease=0.0, min_impurity_spli
t=None,
                        min_samples_leaf=1, min_samples_split=2,
                        min_weight_fraction_leaf=0.0, presort=False,
                        random_state=1550519296, splitter='best'),
 DecisionTreeClassifier(class_weight=None, criterion='entropy', max_
depth=4,
                        max_features=None, max_leaf_nodes=None,
                        min_impurity_decrease=0.0, min_impurity_spli
t=None,
                        min_samples_leaf=1, min_samples_split=2,
                        min_weight_fraction_leaf=0.0, presort=False,
                        random_state=1413460195, splitter='best'),
 DecisionTreeClassifier(class_weight=None, criterion='entropy', max_
depth=4,
                        max_features=None, max_leaf_nodes=None,
                        min_impurity_decrease=0.0, min_impurity_spli
t=None,
                        min_samples_leaf=1, min_samples_split=2,
                        min_weight_fraction_leaf=0.0, presort=False,
                        random_state=554749366, splitter='best'),
 DecisionTreeClassifier(class_weight=None, criterion='entropy', max_
depth=4,
                        max_features=None, max_leaf_nodes=None,
                        min_impurity_decrease=0.0, min_impurity_spli
t=None,
                        min_samples_leaf=1, min_samples_split=2,
                        min_weight_fraction_leaf=0.0, presort=False,
                        random_state=1882573037, splitter='best'),
 DecisionTreeClassifier(class_weight=None, criterion='entropy', max_
depth=4,
                       max_features=None, max_leaf_nodes=None,
                       min_impurity_decrease=0.0, min_impurity_spli
t=None,
                       min_samples_leaf=1, min_samples_split=2,
                       min_weight_fraction_leaf=0.0, presort=False,
                       random_state=1470386578, splitter='best'),
 DecisionTreeClassifier(class_weight=None, criterion='entropy', max_
depth=4,
                       max_features=None, max_leaf_nodes=None,
                       min_impurity_decrease=0.0, min_impurity_spli
t=None,
                       min_samples_leaf=1, min_samples_split=2,
                       min_weight_fraction_leaf=0.0, presort=False,
                       random_state=1361166231, splitter='best'),
 DecisionTreeClassifier(class_weight=None, criterion='entropy', max_
depth=4,
                       max_features=None, max_leaf_nodes=None,
                       min_impurity_decrease=0.0, min_impurity_spli
t=None,
                       min_samples_leaf=1, min_samples_split=2,
                       min_weight_fraction_leaf=0.0, presort=False,
                       random_state=484371728, splitter='best'),
 DecisionTreeClassifier(class_weight=None, criterion='entropy', max_
depth=4,
                       max_features=None, max_leaf_nodes=None,
                       min_impurity_decrease=0.0, min_impurity_spli
t=None,
                       min_samples_leaf=1, min_samples_split=2,
                       min_weight_fraction_leaf=0.0, presort=False,
                       random_state=874145041, splitter='best'),
 DecisionTreeClassifier(class_weight=None, criterion='entropy', max_
depth=4,
                       max_features=None, max_leaf_nodes=None,
                       min_impurity_decrease=0.0, min_impurity_spli
t=None,
                       min_samples_leaf=1, min_samples_split=2,
                       min_weight_fraction_leaf=0.0, presort=False,
                       random_state=1458501657, splitter='best'),
 DecisionTreeClassifier(class_weight=None, criterion='entropy', max_
depth=4,
                       max_features=None, max_leaf_nodes=None,
                       min_impurity_decrease=0.0, min_impurity_spli
t=None,
                       min_samples_leaf=1, min_samples_split=2,
                       min_weight_fraction_leaf=0.0, presort=False,
                       random_state=2033873965, splitter='best'),
 DecisionTreeClassifier(class_weight=None, criterion='entropy', max_
depth=4,
                       max_features=None, max_leaf_nodes=None,
                       min_impurity_decrease=0.0, min_impurity_spli
t=None,
                       min_samples_leaf=1, min_samples_split=2,
                       min_weight_fraction_leaf=0.0, presort=False,
                       random_state=1894172727, splitter='best'),
 DecisionTreeClassifier(class_weight=None, criterion='entropy', max_
depth=4,
                       max_features=None, max_leaf_nodes=None,
                       min_impurity_decrease=0.0, min_impurity_spli
t=None,
                       min_samples_leaf=1, min_samples_split=2,
                       min_weight_fraction_leaf=0.0, presort=False,
                       random_state=91268423, splitter='best'),
 DecisionTreeClassifier(class_weight=None, criterion='entropy', max_
depth=4,
                       max_features=None, max_leaf_nodes=None,
                       min_impurity_decrease=0.0, min_impurity_spli
t=None,
                       min_samples_leaf=1, min_samples_split=2,
                       min_weight_fraction_leaf=0.0, presort=False,
                       random_state=62078030, splitter='best'),
 DecisionTreeClassifier(class_weight=None, criterion='entropy', max_
depth=4,
                       max_features=None, max_leaf_nodes=None,
                       min_impurity_decrease=0.0, min_impurity_spli
t=None,
                       min_samples_leaf=1, min_samples_split=2,
                       min_weight_fraction_leaf=0.0, presort=False,
                       random_state=1702124106, splitter='best'),
 DecisionTreeClassifier(class_weight=None, criterion='entropy', max_
depth=4,
                       max_features=None, max_leaf_nodes=None,
                       min_impurity_decrease=0.0, min_impurity_spli
t=None,
                       min_samples_leaf=1, min_samples_split=2,
                       min_weight_fraction_leaf=0.0, presort=False,
                       random_state=963804990, splitter='best'),
 DecisionTreeClassifier(class_weight=None, criterion='entropy', max_
depth=4,
                       max_features=None, max_leaf_nodes=None,
                       min_impurity_decrease=0.0, min_impurity_spli
t=None,
                       min_samples_leaf=1, min_samples_split=2,
                       min_weight_fraction_leaf=0.0, presort=False,
                       random_state=889293054, splitter='best'),
 DecisionTreeClassifier(class_weight=None, criterion='entropy', max_
depth=4,
                       max_features=None, max_leaf_nodes=None,
                       min_impurity_decrease=0.0, min_impurity_spli
t=None,
                       min_samples_leaf=1, min_samples_split=2,
                       min_weight_fraction_leaf=0.0, presort=False,
                       random_state=1939125160, splitter='best'),
 DecisionTreeClassifier(class_weight=None, criterion='entropy', max_
depth=4,
                       max_features=None, max_leaf_nodes=None,
                       min_impurity_decrease=0.0, min_impurity_spli
t=None,
                       min_samples_leaf=1, min_samples_split=2,
                       min_weight_fraction_leaf=0.0, presort=False,
                       random_state=766829522, splitter='best'),
 DecisionTreeClassifier(class_weight=None, criterion='entropy', max_
depth=4,
                       max_features=None, max_leaf_nodes=None,
                       min_impurity_decrease=0.0, min_impurity_spli
t=None,
                       min_samples_leaf=1, min_samples_split=2,
                       min_weight_fraction_leaf=0.0, presort=False,
                       random_state=1330890377, splitter='best'),
 DecisionTreeClassifier(class_weight=None, criterion='entropy', max_
depth=4,
                       max_features=None, max_leaf_nodes=None,
                       min_impurity_decrease=0.0, min_impurity_spli
t=None,
                       min_samples_leaf=1, min_samples_split=2,
                       min_weight_fraction_leaf=0.0, presort=False,
                       random_state=1923321578, splitter='best'),
 DecisionTreeClassifier(class_weight=None, criterion='entropy', max_
depth=4,
                       max_features=None, max_leaf_nodes=None,
                       min_impurity_decrease=0.0, min_impurity_spli
t=None,
                       min_samples_leaf=1, min_samples_split=2,
                       min_weight_fraction_leaf=0.0, presort=False,
                       random_state=573446295, splitter='best'),
 DecisionTreeClassifier(class_weight=None, criterion='entropy', max_
depth=4,
                       max_features=None, max_leaf_nodes=None,
                       min_impurity_decrease=0.0, min_impurity_spli
t=None,
                       min_samples_leaf=1, min_samples_split=2,
                       min_weight_fraction_leaf=0.0, presort=False,
                       random_state=316863034, splitter='best'),
 DecisionTreeClassifier(class_weight=None, criterion='entropy', max_
depth=4,
                       max_features=None, max_leaf_nodes=None,
                       min_impurity_decrease=0.0, min_impurity_spli
t=None,
                       min_samples_leaf=1, min_samples_split=2,
                       min_weight_fraction_leaf=0.0, presort=False,
                       random_state=844066409, splitter='best'),
 DecisionTreeClassifier(class_weight=None, criterion='entropy', max_
depth=4,
                       max_features=None, max_leaf_nodes=None,
                       min_impurity_decrease=0.0, min_impurity_spli
t=None,
                       min_samples_leaf=1, min_samples_split=2,
                       min_weight_fraction_leaf=0.0, presort=False,
                       random_state=1491491739, splitter='best'),
 DecisionTreeClassifier(class_weight=None, criterion='entropy', max_
depth=4,
                       max_features=None, max_leaf_nodes=None,
                       min_impurity_decrease=0.0, min_impurity_spli
t=None,
                       min_samples_leaf=1, min_samples_split=2,
                       min_weight_fraction_leaf=0.0, presort=False,
                       random_state=48606468, splitter='best'),
 DecisionTreeClassifier(class_weight=None, criterion='entropy', max_
depth=4,
                       max_features=None, max_leaf_nodes=None,
                       min_impurity_decrease=0.0, min_impurity_spli
t=None,
                       min_samples_leaf=1, min_samples_split=2,
                       min_weight_fraction_leaf=0.0, presort=False,
                       random_state=1206131705, splitter='best'),
 DecisionTreeClassifier(class_weight=None, criterion='entropy', max_
depth=4,
                       max_features=None, max_leaf_nodes=None,
                       min_impurity_decrease=0.0, min_impurity_spli
t=None,
                       min_samples_leaf=1, min_samples_split=2,
                       min_weight_fraction_leaf=0.0, presort=False,
                       random_state=1731208420, splitter='best'),
 DecisionTreeClassifier(class_weight=None, criterion='entropy', max_
depth=4,
                       max_features=None, max_leaf_nodes=None,
                       min_impurity_decrease=0.0, min_impurity_spli
t=None,
                       min_samples_leaf=1, min_samples_split=2,
                       min_weight_fraction_leaf=0.0, presort=False,
                       random_state=2109098374, splitter='best'),
 DecisionTreeClassifier(class_weight=None, criterion='entropy', max_
depth=4,
                       max_features=None, max_leaf_nodes=None,
                       min_impurity_decrease=0.0, min_impurity_spli
t=None,
                       min_samples_leaf=1, min_samples_split=2,
                       min_weight_fraction_leaf=0.0, presort=False,
                       random_state=342189113, splitter='best'),
 DecisionTreeClassifier(class_weight=None, criterion='entropy', max_
depth=4,
                       max_features=None, max_leaf_nodes=None,
                       min_impurity_decrease=0.0, min_impurity_spli
t=None,
                       min_samples_leaf=1, min_samples_split=2,
                       min_weight_fraction_leaf=0.0, presort=False,
                       random_state=1315463242, splitter='best'),
 DecisionTreeClassifier(class_weight=None, criterion='entropy', max_
depth=4,
                       max_features=None, max_leaf_nodes=None,
                       min_impurity_decrease=0.0, min_impurity_spli
t=None,
                       min_samples_leaf=1, min_samples_split=2,
                       min_weight_fraction_leaf=0.0, presort=False,
                       random_state=687553301, splitter='best'),
 DecisionTreeClassifier(class_weight=None, criterion='entropy', max_
depth=4,
                       max_features=None, max_leaf_nodes=None,
                       min_impurity_decrease=0.0, min_impurity_spli
t=None,
                       min_samples_leaf=1, min_samples_split=2,
                       min_weight_fraction_leaf=0.0, presort=False,
                       random_state=1341167774, splitter='best'),
 DecisionTreeClassifier(class_weight=None, criterion='entropy', max_
depth=4,
                       max_features=None, max_leaf_nodes=None,
                       min_impurity_decrease=0.0, min_impurity_spli
t=None,
                       min_samples_leaf=1, min_samples_split=2,
                       min_weight_fraction_leaf=0.0, presort=False,
                       random_state=555194453, splitter='best'),
 DecisionTreeClassifier(class_weight=None, criterion='entropy', max_
depth=4,
                       max_features=None, max_leaf_nodes=None,
                       min_impurity_decrease=0.0, min_impurity_spli
t=None,
                       min_samples_leaf=1, min_samples_split=2,
                       min_weight_fraction_leaf=0.0, presort=False,
                       random_state=296091748, splitter='best'),
 DecisionTreeClassifier(class_weight=None, criterion='entropy', max_
depth=4,
                       max_features=None, max_leaf_nodes=None,
                       min_impurity_decrease=0.0, min_impurity_spli
```

```
t=None,
                        min_samples_leaf=1, min_samples_split=2,
                        min_weight_fraction_leaf=0.0, presort=False,
                        random_state=1147976689, splitter='best'),
 DecisionTreeClassifier(class_weight=None, criterion='entropy', max_
depth=4,
                        max_features=None, max_leaf_nodes=None,
                        min_impurity_decrease=0.0, min_impurity_spli
t=None,
                        min_samples_leaf=1, min_samples_split=2,
                        min_weight_fraction_leaf=0.0, presort=False,
                        random_state=953544143, splitter='best'),
 DecisionTreeClassifier(class_weight=None, criterion='entropy', max_
depth=4,
                        max_features=None, max_leaf_nodes=None,
                        min_impurity_decrease=0.0, min_impurity_spli
t=None,
                        min_samples_leaf=1, min_samples_split=2,
                        min_weight_fraction_leaf=0.0, presort=False,
                        random_state=766640980, splitter='best'),
 DecisionTreeClassifier(class_weight=None, criterion='entropy', max_
depth=4,
                        max_features=None, max_leaf_nodes=None,
                        min_impurity_decrease=0.0, min_impurity_spli
t=None,
                        min_samples_leaf=1, min_samples_split=2,
                        min_weight_fraction_leaf=0.0, presort=False,
                        random_state=455490427, splitter='best'),
 DecisionTreeClassifier(class_weight=None, criterion='entropy', max_
depth=4,
                        max_features=None, max_leaf_nodes=None,
                        min_impurity_decrease=0.0, min_impurity_spli
t=None,
                        min_samples_leaf=1, min_samples_split=2,
                        min_weight_fraction_leaf=0.0, presort=False,
                        random_state=160556819, splitter='best'),
 DecisionTreeClassifier(class_weight=None, criterion='entropy', max_
depth=4,
                        max_features=None, max_leaf_nodes=None,
                        min_impurity_decrease=0.0, min_impurity_spli
t=None,
                        min_samples_leaf=1, min_samples_split=2,
                        min_weight_fraction_leaf=0.0, presort=False,
                        random_state=749100668, splitter='best'),
 DecisionTreeClassifier(class_weight=None, criterion='entropy', max_
depth=4,
                        max_features=None, max_leaf_nodes=None,
                        min_impurity_decrease=0.0, min_impurity_spli
t=None,
                        min_samples_leaf=1, min_samples_split=2,
                        min_weight_fraction_leaf=0.0, presort=False,
                        random_state=776093812, splitter='best'),
 DecisionTreeClassifier(class_weight=None, criterion='entropy', max_
depth=4,
                        max_features=None, max_leaf_nodes=None,
                        min_impurity_decrease=0.0, min_impurity_spli
t=None,
                        min_samples_leaf=1, min_samples_split=2,
                        min_weight_fraction_leaf=0.0, presort=False,
                        random_state=598294353, splitter='best'),
 DecisionTreeClassifier(class_weight=None, criterion='entropy', max_
depth=4,
                        max_features=None, max_leaf_nodes=None,
                        min_impurity_decrease=0.0, min_impurity_spli
t=None,
                        min_samples_leaf=1, min_samples_split=2,
                        min_weight_fraction_leaf=0.0, presort=False,
                        random_state=338475798, splitter='best'),
 DecisionTreeClassifier(class_weight=None, criterion='entropy', max_
depth=4,
                        max_features=None, max_leaf_nodes=None,
                        min_impurity_decrease=0.0, min_impurity_spli
t=None,
                        min_samples_leaf=1, min_samples_split=2,
                        min_weight_fraction_leaf=0.0, presort=False,
                        random_state=1480375130, splitter='best'),
 DecisionTreeClassifier(class_weight=None, criterion='entropy', max_
depth=4,
                        max_features=None, max_leaf_nodes=None,
                        min_impurity_decrease=0.0, min_impurity_spli
t=None,
                        min_samples_leaf=1, min_samples_split=2,
                        min_weight_fraction_leaf=0.0, presort=False,
                        random_state=1869373923, splitter='best'),
 DecisionTreeClassifier(class_weight=None, criterion='entropy', max_
depth=4,
                        max_features=None, max_leaf_nodes=None,
                        min_impurity_decrease=0.0, min_impurity_spli
t=None,
                        min_samples_leaf=1, min_samples_split=2,
                        min_weight_fraction_leaf=0.0, presort=False,
                        random_state=1491563726, splitter='best'),
 DecisionTreeClassifier(class_weight=None, criterion='entropy', max_
depth=4,
                        max_features=None, max_leaf_nodes=None,
                        min_impurity_decrease=0.0, min_impurity_spli
t=None,
                        min_samples_leaf=1, min_samples_split=2,
                        min_weight_fraction_leaf=0.0, presort=False,
                        random_state=1939244847, splitter='best'),
 DecisionTreeClassifier(class_weight=None, criterion='entropy', max_
depth=4,
                        max_features=None, max_leaf_nodes=None,
                        min_impurity_decrease=0.0, min_impurity_spli
t=None,
                        min_samples_leaf=1, min_samples_split=2,
                        min_weight_fraction_leaf=0.0, presort=False,
                        random_state=1892945645, splitter='best'),
 DecisionTreeClassifier(class_weight=None, criterion='entropy', max_
depth=4,
                        max_features=None, max_leaf_nodes=None,
                        min_impurity_decrease=0.0, min_impurity_spli
t=None,
                        min_samples_leaf=1, min_samples_split=2,
                        min_weight_fraction_leaf=0.0, presort=False,
                        random_state=1019133162, splitter='best'),
 DecisionTreeClassifier(class_weight=None, criterion='entropy', max_
depth=4,
                        max_features=None, max_leaf_nodes=None,
                        min_impurity_decrease=0.0, min_impurity_spli
t=None,
                        min_samples_leaf=1, min_samples_split=2,
                        min_weight_fraction_leaf=0.0, presort=False,
                        random_state=1430229607, splitter='best'),
 DecisionTreeClassifier(class_weight=None, criterion='entropy', max_
depth=4,
                        max_features=None, max_leaf_nodes=None,
                        min_impurity_decrease=0.0, min_impurity_spli
t=None,
                        min_samples_leaf=1, min_samples_split=2,
                        min_weight_fraction_leaf=0.0, presort=False,
                        random_state=89896859, splitter='best'),
 DecisionTreeClassifier(class_weight=None, criterion='entropy', max_
depth=4,
                        max_features=None, max_leaf_nodes=None,
                        min_impurity_decrease=0.0, min_impurity_spli
t=None,
                        min_samples_leaf=1, min_samples_split=2,
                        min_weight_fraction_leaf=0.0, presort=False,
                        random_state=1768440018, splitter='best'),
 DecisionTreeClassifier(class_weight=None, criterion='entropy', max_
depth=4,
                        max_features=None, max_leaf_nodes=None,
                        min_impurity_decrease=0.0, min_impurity_spli
t=None,
                        min_samples_leaf=1, min_samples_split=2,
                        min_weight_fraction_leaf=0.0, presort=False,
                        random_state=1323641480, splitter='best'),
 DecisionTreeClassifier(class_weight=None, criterion='entropy', max_
depth=4,
                        max_features=None, max_leaf_nodes=None,
                        min_impurity_decrease=0.0, min_impurity_spli
t=None,
                        min_samples_leaf=1, min_samples_split=2,
                        min_weight_fraction_leaf=0.0, presort=False,
                        random_state=1948649296, splitter='best'),
 DecisionTreeClassifier(class_weight=None, criterion='entropy', max_
depth=4,
                        max_features=None, max_leaf_nodes=None,
                        min_impurity_decrease=0.0, min_impurity_spli
t=None,
                        min_samples_leaf=1, min_samples_split=2,
                        min_weight_fraction_leaf=0.0, presort=False,
                        random_state=168190458, splitter='best'),
 DecisionTreeClassifier(class_weight=None, criterion='entropy', max_
depth=4,
                        max_features=None, max_leaf_nodes=None,
                        min_impurity_decrease=0.0, min_impurity_spli
t=None,
                        min_samples_leaf=1, min_samples_split=2,
                        min_weight_fraction_leaf=0.0, presort=False,
                        random_state=1903993590, splitter='best'),
 DecisionTreeClassifier(class_weight=None, criterion='entropy', max_
depth=4,
                        max_features=None, max_leaf_nodes=None,
                        min_impurity_decrease=0.0, min_impurity_spli
t=None,
                        min_samples_leaf=1, min_samples_split=2,
                        min_weight_fraction_leaf=0.0, presort=False,
                        random_state=36638185, splitter='best'),
 DecisionTreeClassifier(class_weight=None, criterion='entropy', max_
depth=4,
                        max_features=None, max_leaf_nodes=None,
                        min_impurity_decrease=0.0, min_impurity_spli
t=None,
                        min_samples_leaf=1, min_samples_split=2,
                        min_weight_fraction_leaf=0.0, presort=False,
                        random_state=1330893745, splitter='best'),
 DecisionTreeClassifier(class_weight=None, criterion='entropy', max_
depth=4,
                        max_features=None, max_leaf_nodes=None,
                        min_impurity_decrease=0.0, min_impurity_spli
t=None,
                        min_samples_leaf=1, min_samples_split=2,
                        min_weight_fraction_leaf=0.0, presort=False,
                        random_state=1442805223, splitter='best'),
 DecisionTreeClassifier(class_weight=None, criterion='entropy', max_
depth=4,
                        max_features=None, max_leaf_nodes=None,
                        min_impurity_decrease=0.0, min_impurity_spli
t=None,
                        min_samples_leaf=1, min_samples_split=2,
                        min_weight_fraction_leaf=0.0, presort=False,
                        random_state=2034281815, splitter='best'),
 DecisionTreeClassifier(class_weight=None, criterion='entropy', max_
depth=4,
                        max_features=None, max_leaf_nodes=None,
                        min_impurity_decrease=0.0, min_impurity_spli
t=None,
                        min_samples_leaf=1, min_samples_split=2,
                        min_weight_fraction_leaf=0.0, presort=False,
                        random_state=2123062096, splitter='best'),
 DecisionTreeClassifier(class_weight=None, criterion='entropy', max_
depth=4,
                        max_features=None, max_leaf_nodes=None,
                        min_impurity_decrease=0.0, min_impurity_spli
t=None,
                        min_samples_leaf=1, min_samples_split=2,
                        min_weight_fraction_leaf=0.0, presort=False,
                        random_state=1331199993, splitter='best'),
 DecisionTreeClassifier(class_weight=None, criterion='entropy', max_
depth=4,
                        max_features=None, max_leaf_nodes=None,
                        min_impurity_decrease=0.0, min_impurity_spli
t=None,
                        min_samples_leaf=1, min_samples_split=2,
                        min_weight_fraction_leaf=0.0, presort=False,
                        random_state=1370116973, splitter='best'),
 DecisionTreeClassifier(class_weight=None, criterion='entropy', max_
depth=4,
                        max_features=None, max_leaf_nodes=None,
                        min_impurity_decrease=0.0, min_impurity_spli
t=None,
                        min_samples_leaf=1, min_samples_split=2,
                        min_weight_fraction_leaf=0.0, presort=False,
                        random_state=1428536671, splitter='best'),
 DecisionTreeClassifier(class_weight=None, criterion='entropy', max_
depth=4,
                        max_features=None, max_leaf_nodes=None,
                        min_impurity_decrease=0.0, min_impurity_spli
t=None,
                        min_samples_leaf=1, min_samples_split=2,
                        min_weight_fraction_leaf=0.0, presort=False,
                        random_state=1961456663, splitter='best'),
 DecisionTreeClassifier(class_weight=None, criterion='entropy', max_
depth=4,
                        max_features=None, max_leaf_nodes=None,
                        min_impurity_decrease=0.0, min_impurity_spli
t=None,
                        min_samples_leaf=1, min_samples_split=2,
                        min_weight_fraction_leaf=0.0, presort=False,
                        random_state=1722692559, splitter='best'),
 DecisionTreeClassifier(class_weight=None, criterion='entropy', max_
depth=4,
                        max_features=None, max_leaf_nodes=None,
                        min_impurity_decrease=0.0, min_impurity_spli
t=None,
                        min_samples_leaf=1, min_samples_split=2,
                        min_weight_fraction_leaf=0.0, presort=False,
                        random_state=1328959080, splitter='best'),
 DecisionTreeClassifier(class_weight=None, criterion='entropy', max_
depth=4,
                        max_features=None, max_leaf_nodes=None,
                        min_impurity_decrease=0.0, min_impurity_spli
t=None,
                        min_samples_leaf=1, min_samples_split=2,
                        min_weight_fraction_leaf=0.0, presort=False,
                        random_state=2145262861, splitter='best'),
 DecisionTreeClassifier(class_weight=None, criterion='entropy', max_
depth=4,
                        max_features=None, max_leaf_nodes=None,
                        min_impurity_decrease=0.0, min_impurity_spli
t=None,
                        min_samples_leaf=1, min_samples_split=2,
                        min_weight_fraction_leaf=0.0, presort=False,
                        random_state=1389233840, splitter='best'),
 DecisionTreeClassifier(class_weight=None, criterion='entropy', max_
depth=4,
                        max_features=None, max_leaf_nodes=None,
                        min_impurity_decrease=0.0, min_impurity_spli
t=None,
                        min_samples_leaf=1, min_samples_split=2,
                        min_weight_fraction_leaf=0.0, presort=False,
                        random_state=1023603906, splitter='best'),
 DecisionTreeClassifier(class_weight=None, criterion='entropy', max_
depth=4,
                        max_features=None, max_leaf_nodes=None,
                        min_impurity_decrease=0.0, min_impurity_spli
t=None,
                        min_samples_leaf=1, min_samples_split=2,
                        min_weight_fraction_leaf=0.0, presort=False,
                        random_state=606217668, splitter='best'),
 DecisionTreeClassifier(class_weight=None, criterion='entropy', max_
depth=4,
                        max_features=None, max_leaf_nodes=None,
                        min_impurity_decrease=0.0, min_impurity_spli
t=None,
                        min_samples_leaf=1, min_samples_split=2,
                        min_weight_fraction_leaf=0.0, presort=False,
                        random_state=445026325, splitter='best'),
 DecisionTreeClassifier(class_weight=None, criterion='entropy', max_
depth=4,
                        max_features=None, max_leaf_nodes=None,
                        min_impurity_decrease=0.0, min_impurity_spli
t=None,
                        min_samples_leaf=1, min_samples_split=2,
                        min_weight_fraction_leaf=0.0, presort=False,
                        random_state=998528415, splitter='best'),
 DecisionTreeClassifier(class_weight=None, criterion='entropy', max_
depth=4,
                        max_features=None, max_leaf_nodes=None,
                        min_impurity_decrease=0.0, min_impurity_spli
t=None,
                        min_samples_leaf=1, min_samples_split=2,
                        min_weight_fraction_leaf=0.0, presort=False,
                        random_state=520769204, splitter='best'),
 DecisionTreeClassifier(class_weight=None, criterion='entropy', max_
depth=4,
                        max_features=None, max_leaf_nodes=None,
                        min_impurity_decrease=0.0, min_impurity_spli
t=None,
                        min_samples_leaf=1, min_samples_split=2,
                        min_weight_fraction_leaf=0.0, presort=False,
                        random_state=1935512569, splitter='best'),
 DecisionTreeClassifier(class_weight=None, criterion='entropy', max_
depth=4,
                        max_features=None, max_leaf_nodes=None,
                        min_impurity_decrease=0.0, min_impurity_spli
t=None,
                        min_samples_leaf=1, min_samples_split=2,
                        min_weight_fraction_leaf=0.0, presort=False,
                        random_state=531795499, splitter='best'),
 DecisionTreeClassifier(class_weight=None, criterion='entropy', max_
depth=4,
                        max_features=None, max_leaf_nodes=None,
                        min_impurity_decrease=0.0, min_impurity_spli
t=None,
                        min_samples_leaf=1, min_samples_split=2,
                        min_weight_fraction_leaf=0.0, presort=False,
                        random_state=2073176537, splitter='best'),
 DecisionTreeClassifier(class_weight=None, criterion='entropy', max_
depth=4,
                        max_features=None, max_leaf_nodes=None,
                        min_impurity_decrease=0.0, min_impurity_spli
t=None,
                        min_samples_leaf=1, min_samples_split=2,
                        min_weight_fraction_leaf=0.0, presort=False,
                        random_state=632471792, splitter='best'),
 DecisionTreeClassifier(class_weight=None, criterion='entropy', max_
depth=4,
                        max_features=None, max_leaf_nodes=None,
                        min_impurity_decrease=0.0, min_impurity_spli
t=None,
                        min_samples_leaf=1, min_samples_split=2,
                        min_weight_fraction_leaf=0.0, presort=False,
                        random_state=1637909726, splitter='best'),
 DecisionTreeClassifier(class_weight=None, criterion='entropy', max_
depth=4,
                        max_features=None, max_leaf_nodes=None,
                        min_impurity_decrease=0.0, min_impurity_spli
t=None,
                        min_samples_leaf=1, min_samples_split=2,
                        min_weight_fraction_leaf=0.0, presort=False,
                        random_state=1017279060, splitter='best'),
 DecisionTreeClassifier(class_weight=None, criterion='entropy', max_
depth=4,
                        max_features=None, max_leaf_nodes=None,
                        min_impurity_decrease=0.0, min_impurity_spli
t=None,
                        min_samples_leaf=1, min_samples_split=2,
                        min_weight_fraction_leaf=0.0, presort=False,
                        random_state=1599810166, splitter='best'),
 DecisionTreeClassifier(class_weight=None, criterion='entropy', max_
depth=4,
                        max_features=None, max_leaf_nodes=None,
                        min_impurity_decrease=0.0, min_impurity_spli
t=None,
                        min_samples_leaf=1, min_samples_split=2,
                        min_weight_fraction_leaf=0.0, presort=False,
                        random_state=1917937328, splitter='best'),
 DecisionTreeClassifier(class_weight=None, criterion='entropy', max_
depth=4,
                        max_features=None, max_leaf_nodes=None,
                        min_impurity_decrease=0.0, min_impurity_spli
t=None,
                        min_samples_leaf=1, min_samples_split=2,
                        min_weight_fraction_leaf=0.0, presort=False,
                        random_state=2055237436, splitter='best'),
 DecisionTreeClassifier(class_weight=None, criterion='entropy', max_
depth=4,
                        max_features=None, max_leaf_nodes=None,
                        min_impurity_decrease=0.0, min_impurity_spli
t=None,
                        min_samples_leaf=1, min_samples_split=2,
                        min_weight_fraction_leaf=0.0, presort=False,
                        random_state=279452207, splitter='best'),
 DecisionTreeClassifier(class_weight=None, criterion='entropy', max_
depth=4,
                        max_features=None, max_leaf_nodes=None,
                        min_impurity_decrease=0.0, min_impurity_spli
t=None,
                        min_samples_leaf=1, min_samples_split=2,
                        min_weight_fraction_leaf=0.0, presort=False,
                        random_state=1216725771, splitter='best'),
 DecisionTreeClassifier(class_weight=None, criterion='entropy', max_
depth=4,
                        max_features=None, max_leaf_nodes=None,
                        min_impurity_decrease=0.0, min_impurity_spli
t=None,
                        min_samples_leaf=1, min_samples_split=2,
                        min_weight_fraction_leaf=0.0, presort=False,
                        random_state=703936783, splitter='best'),
 DecisionTreeClassifier(class_weight=None, criterion='entropy', max_
depth=4,
                        max_features=None, max_leaf_nodes=None,
                        min_impurity_decrease=0.0, min_impurity_spli
t=None,
                        min_samples_leaf=1, min_samples_split=2,
                        min_weight_fraction_leaf=0.0, presort=False,
                        random_state=1258939861, splitter='best'),
 DecisionTreeClassifier(class_weight=None, criterion='entropy', max_
depth=4,
                        max_features=None, max_leaf_nodes=None,
                        min_impurity_decrease=0.0, min_impurity_spli
t=None,
                        min_samples_leaf=1, min_samples_split=2,
                        min_weight_fraction_leaf=0.0, presort=False,
                        random_state=810911689, splitter='best'),
 DecisionTreeClassifier(class_weight=None, criterion='entropy', max_
depth=4,
                        max_features=None, max_leaf_nodes=None,
                        min_impurity_decrease=0.0, min_impurity_spli
t=None,
                        min_samples_leaf=1, min_samples_split=2,
                        min_weight_fraction_leaf=0.0, presort=False,
                        random_state=284863514, splitter='best')]
```

In [79]:

```
bagging.estimators_features_
```

Out[79]:

```
[array([0, 1, 2, 3, 4, 5]),
 array([0, 1, 2, 3, 4, 5]),
 array([0, 1, 2, 3, 4, 5]),
 array([0, 1, 2, 3, 4, 5]),
 array([0, 1, 2, 3, 4, 5]),
 array([0, 1, 2, 3, 4, 5]),
 array([0, 1, 2, 3, 4, 5]),
 array([0, 1, 2, 3, 4, 5]),
 array([0, 1, 2, 3, 4, 5]),
 array([0, 1, 2, 3, 4, 5]),
 array([0, 1, 2, 3, 4, 5]),
 array([0, 1, 2, 3, 4, 5]),
 array([0, 1, 2, 3, 4, 5]),
 array([0, 1, 2, 3, 4, 5]),
 array([0, 1, 2, 3, 4, 5]),
 array([0, 1, 2, 3, 4, 5]),
 array([0, 1, 2, 3, 4, 5]),
 array([0, 1, 2, 3, 4, 5]),
 array([0, 1, 2, 3, 4, 5]),
 array([0, 1, 2, 3, 4, 5]),
 array([0, 1, 2, 3, 4, 5]),
 array([0, 1, 2, 3, 4, 5]),
 array([0, 1, 2, 3, 4, 5]),
 array([0, 1, 2, 3, 4, 5]),
 array([0, 1, 2, 3, 4, 5]),
 array([0, 1, 2, 3, 4, 5]),
 array([0, 1, 2, 3, 4, 5]),
 array([0, 1, 2, 3, 4, 5]),
 array([0, 1, 2, 3, 4, 5]),
 array([0, 1, 2, 3, 4, 5]),
 array([0, 1, 2, 3, 4, 5]),
 array([0, 1, 2, 3, 4, 5]),
 array([0, 1, 2, 3, 4, 5]),
 array([0, 1, 2, 3, 4, 5]),
 array([0, 1, 2, 3, 4, 5]),
 array([0, 1, 2, 3, 4, 5]),
 array([0, 1, 2, 3, 4, 5]),
 array([0, 1, 2, 3, 4, 5]),
 array([0, 1, 2, 3, 4, 5]),
 array([0, 1, 2, 3, 4, 5]),
 array([0, 1, 2, 3, 4, 5]),
 array([0, 1, 2, 3, 4, 5]),
 array([0, 1, 2, 3, 4, 5]),
 array([0, 1, 2, 3, 4, 5]),
 array([0, 1, 2, 3, 4, 5]),
 array([0, 1, 2, 3, 4, 5]),
 array([0, 1, 2, 3, 4, 5]),
 array([0, 1, 2, 3, 4, 5]),
 array([0, 1, 2, 3, 4, 5]),
 array([0, 1, 2, 3, 4, 5]),
 array([0, 1, 2, 3, 4, 5]),
 array([0, 1, 2, 3, 4, 5]),
 array([0, 1, 2, 3, 4, 5]),
 array([0, 1, 2, 3, 4, 5]),
 array([0, 1, 2, 3, 4, 5]),
 array([0, 1, 2, 3, 4, 5]),
 array([0, 1, 2, 3, 4, 5]),
 array([0, 1, 2, 3, 4, 5]),
 array([0, 1, 2, 3, 4, 5]),
 array([0, 1, 2, 3, 4, 5]),
 array([0, 1, 2, 3, 4, 5]),
 array([0, 1, 2, 3, 4, 5]),
 array([0, 1, 2, 3, 4, 5]),
 array([0, 1, 2, 3, 4, 5]),
 array([0, 1, 2, 3, 4, 5]),
 array([0, 1, 2, 3, 4, 5]),
 array([0, 1, 2, 3, 4, 5]),
 array([0, 1, 2, 3, 4, 5]),
 array([0, 1, 2, 3, 4, 5]),
 array([0, 1, 2, 3, 4, 5]),
 array([0, 1, 2, 3, 4, 5]),
 array([0, 1, 2, 3, 4, 5]),
 array([0, 1, 2, 3, 4, 5]),
 array([0, 1, 2, 3, 4, 5]),
 array([0, 1, 2, 3, 4, 5]),
 array([0, 1, 2, 3, 4, 5]),
 array([0, 1, 2, 3, 4, 5]),
 array([0, 1, 2, 3, 4, 5]),
 array([0, 1, 2, 3, 4, 5]),
 array([0, 1, 2, 3, 4, 5]),
 array([0, 1, 2, 3, 4, 5]),
 array([0, 1, 2, 3, 4, 5]),
 array([0, 1, 2, 3, 4, 5]),
 array([0, 1, 2, 3, 4, 5]),
 array([0, 1, 2, 3, 4, 5]),
 array([0, 1, 2, 3, 4, 5]),
 array([0, 1, 2, 3, 4, 5]),
 array([0, 1, 2, 3, 4, 5]),
 array([0, 1, 2, 3, 4, 5]),
 array([0, 1, 2, 3, 4, 5]),
 array([0, 1, 2, 3, 4, 5]),
 array([0, 1, 2, 3, 4, 5]),
 array([0, 1, 2, 3, 4, 5]),
 array([0, 1, 2, 3, 4, 5]),
 array([0, 1, 2, 3, 4, 5]),
 array([0, 1, 2, 3, 4, 5]),
 array([0, 1, 2, 3, 4, 5]),
 array([0, 1, 2, 3, 4, 5]),
 array([0, 1, 2, 3, 4, 5]),
 array([0, 1, 2, 3, 4, 5])]
```

In [80]:

```
bagging.score(x_titanic_train, y_titanic_train)
```

Out[80]:

```
0.8488745980707395
```

In [82]:

```
bagging.score(x_titanic_test, y_titanic_test)
```

Out[82]:

```
0.8089887640449438
```

extract feature importance
then plot a bar graph

In [86]:

```
import numpy as np
feature_importances = np.mean([
    tree.feature_importances_ for tree in bagging.estimators_
], axis=0)
feature_importances
```

Out[86]:

```
array([0.21931641, 0.46901014, 0.19808952, 0.06888841, 0.01381123,
       0.03088428])
```

In [88]:

```
import matplotlib.pyplot as plt
%matplotlib inline
```

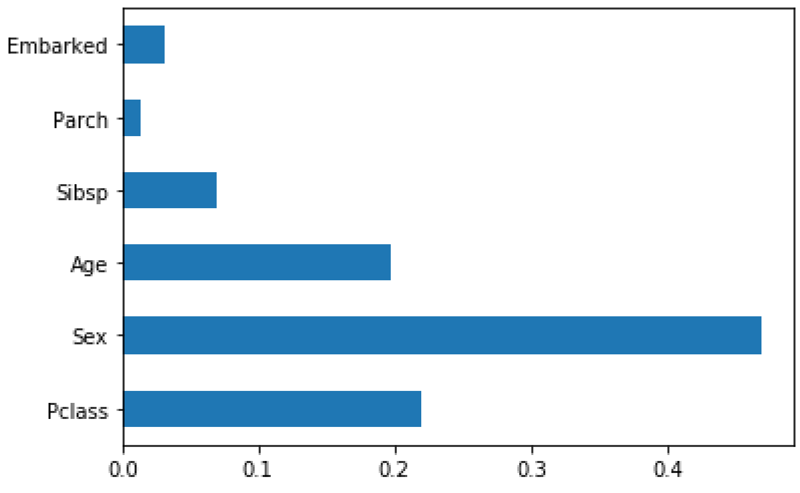
In [92]:

```
feature_importances_series = pd.Series(feature_importances,
                                       index = ['Pclass', 'Sex','Age','Sibsp','P
arch', 'Embarked'])
feature_importances_series.plot(kind = 'barh')

# plt.figure()
# plt.title("Mean Feature importances")
# plt.barh(y = feature_importances, width =
#         color="r", align="center")
# plt.show()
```

Out[92]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x7fcda045db80>
```



Random Forest

In [93]:

```
from sklearn.ensemble import RandomForestClassifier
```

In [94]:

```
rf = RandomForestClassifier(random_state = 198, verbose = 1, )
```

In [96]:

```
rf.fit(x_titanic_train, y_titanic_train)
```

```
/Library/Frameworks/Python.framework/Versions/3.8/lib/python3.8/site
-packages/sklearn/ensemble/forest.py:244: FutureWarning: The default
value of n_estimators will change from 10 in version 0.20 to 100 in
0.22.
  warn("The default value of n_estimators will change from "
<ipython-input-96-163ff23db28a>:1: DataConversionWarning: A column-v
ector y was passed when a 1d array was expected. Please change the s
hape of y to (n_samples,), for example using ravel().
  rf.fit(x_titanic_train, y_titanic_train)
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurr
ent workers.
[Parallel(n_jobs=1)]: Done  10 out of  10 | elapsed:    0.0s finishe
d
```

Out[96]:

```
RandomForestClassifier(bootstrap=True, class_weight=None, criterion
='gini',
                       max_depth=None, max_features='auto', max_leaf
_nodes=None,
                       min_impurity_decrease=0.0, min_impurity_split
=None,
                       min_samples_leaf=1, min_samples_split=2,
                       min_weight_fraction_leaf=0.0, n_estimators=1
0,
                       n_jobs=None, oob_score=False, random_state=19
8,
                       verbose=1, warm_start=False)
```

rf.n_features_

In [99]:

```
rf.feature_importances_
```

Out[99]:

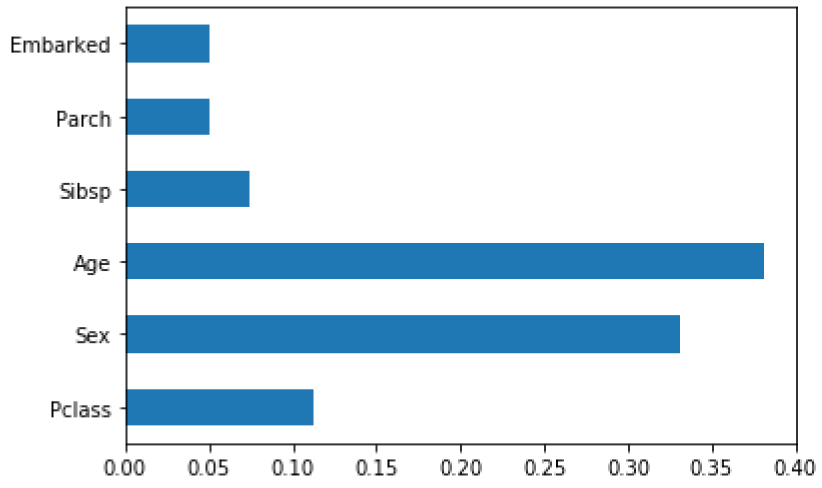
```
array([0.11203241, 0.3309911 , 0.3810256 , 0.07413389, 0.0508831 ,
       0.0509339 ])
```

In [103]:

```
rf_feature_importances = pd.Series(rf.feature_importances_,
                                   index = ['Pclass', 'Sex','Age','Sibsp','Parc
h', 'Embarked'])
rf_feature_importances.plot(kind = 'barh')
```

Out[103]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x7fcd807b31f0>
```



In [104]:

```
rf.score(x_titanic_train, y_titanic_train)
```

```
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurr
ent workers.
[Parallel(n_jobs=1)]: Done  10 out of  10 | elapsed:    0.0s finishe
d
```

Out[104]:

```
0.927652733118971
```

In [105]:

```
rf.score(x_titanic_test, y_titanic_test)
```

```
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurr
ent workers.
[Parallel(n_jobs=1)]: Done  10 out of  10 | elapsed:    0.0s finishe
d
```

Out[105]:

```
0.8014981273408239
```

now change the values for following parameters of random forest and create model rf2
criterion to entropy
max_features to 4
n_estimators to 50
max_depth to 4

Support Vector Machines

In [113]:

```
import nltk
sms = pd.read_csv("/Users/moizzah/Desktop/spam.csv", encoding = 'latin')
sms.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 5572 entries, 0 to 5571
Data columns (total 5 columns):
v1            5572 non-null object
v2            5572 non-null object
Unnamed: 2    50 non-null object
Unnamed: 3    12 non-null object
Unnamed: 4    6 non-null object
dtypes: object(5)
memory usage: 217.8+ KB
```

In [114]:

```
sms.head()
```

Out[114]:

	v1	v2	Unnamed: 2	Unnamed: 3	Unnamed: 4
0	ham	Go until jurong point, crazy.. Available only ...	NaN	NaN	NaN
1	ham	Ok lar... Joking wif u oni...	NaN	NaN	NaN
2	spam	Free entry in 2 a wkly comp to win FA Cup fina...	NaN	NaN	NaN
3	ham	U dun say so early hor... U c already then say...	NaN	NaN	NaN
4	ham	Nah I don't think he goes to usf, he lives aro...	NaN	NaN	NaN

In [115]:

```
sms = sms.loc[:, 'v1':'v2']
sms.head()
```

Out[115]:

	v1	v2
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...

In [117]:

```
sms.columns = ['cat', 'text']
sms.columns
```

Out[117]:

```
Index(['cat', 'text'], dtype='object')
```

In [122]:

```
sms['cat'].value_counts()
```

Out[122]:

```
ham     4825
spam     747
Name: cat, dtype: int64
```

In [123]:

```
lab_enc.fit(sms['cat'].unique())
```

Out[123]:

```
LabelEncoder()
```

In [124]:

```
sms['cat'] = lab_enc.transform(sms['cat'])
```

In [125]:

```
sms.head()
```

Out[125]:

	cat	text
0	0	Go until jurong point, crazy.. Available only ...
1	0	Ok lar... Joking wif u oni...
2	1	Free entry in 2 a wkly comp to win FA Cup fina...
3	0	U dun say so early hor... U c already then say...
4	0	Nah I don't think he goes to usf, he lives aro...

extract the frequency/count of each word
use CountVectorizer class

In [126]:

```
from sklearn.feature_extraction.text import CountVectorizer
```

In [127]:

```
CV = CountVectorizer(ngram_range=(1,2), analyzer = 'word')
```

In [135]:

```
ngrams = CV.fit_transform(sms['text'])
ngrams_id = CV.get_feature_names()
```

In [138]:

```
ngrams_id
```

Out[138]:

```
['00',
 '00 in',
 '00 per',
 '00 sub',
 '00 subs',
 '000',
 '000 bonus',
 '000 cash',
 '000 homeowners',
 '000 pounds',
 '000 price',
 '000 prize',
 '000 xmas',
 '000pes',
 '000pes so',
 '008704050406',
 '008704050406 sp',
 '0089',
 '0089 my',
 '0121',
 '0121 2025050',
 '01223585236',
 '01223585236 xx',
 '01223585334',
 '01223585334 to',
 '0125698789',
 '0125698789 ring',
 '02',
 '02 06',
 '02 09',
 '02 claimcode',
 '02 user',
 '0207',
 '0207 083',
 '0207 153',
 '02072069400',
 '02072069400 bx',
 '02073162414',
 '02073162414 now',
 '02085076972',
 '02085076972 reply',
 '021',
 '021 3680',
 '03',
 '03 05',
 '03 is',
 '03 our',
 '03 this',
 '04',
 '04 call',
 '0430',
 '0430 jul',
 '05',
 '05 05',
 '05 or',
 '050703',
 '050703 csbcm4235wc1n3xx',
 '0578',
 '0578 now',
 '06',
 '06 03',
 '06 05',
 '06 11',
 '06 good',
 '07',
 '07 11',
 '07008009200',
 '07046744435',
 '07046744435 now',
 '07090201529',
 '07090298926',
 '07090298926 to',
 '07099833605',
 '07099833605 to',
 '07123456789',
 '07123456789 to',
 '0721072',
 '0721072 to',
 '07732584351',
 '07732584351 rodger',
 '07734396839',
 '07734396839 ibh',
 '07742676969',
 '07742676969 shows',
 '07753741225',
 '07753741225 shows',
 '0776xxxxxxx',
 '0776xxxxxxx ve',
 '07781482378',
 '07781482378 com',
 '07786200117',
 '077xxx',
 '077xxx won',
 '078',
 '07801543489',
 '07801543489 are',
 '07808',
 '07808 xxxxxx',
 '07808247860',
 '07808247860 shows',
 '07808726822',
 '07808726822 was',
 '07815296484',
 '07815296484 shows',
 '07821230901',
 '078498',
 '078498 shows',
 '07880867867',
 '0789xxxxxxx',
 '0789xxxxxxx today',
 '07946746291',
 '07946746291 07880867867',
 '0796xxxxxx',
 '0796xxxxxx today',
 '07973788240',
 '07973788240 shows',
 '07xxxxxxxxx',
 '07xxxxxxxxx shows',
 '07xxxxxxxxx won',
 '08',
 '08 03',
 '0800',
 '0800 0721072',
 '0800 169',
 '0800 18',
 '0800 195',
 '0800 1956669',
 '0800 505060',
 '0800 542',
 '08000407165',
 '08000407165 18',
 '08000776320',
 '08000776320 now',
 '08000839402',
 '08000839402 now',
 '08000839402 or',
 '08000839402 or2optout',
 '08000930705',
 '08000930705 del',
 '08000930705 for',
 '08000930705 now',
 '08000930705 or',
 '08000938767',
 '08000938767 to',
 '08001950382',
 '08001950382 or',
 '08002888812',
 '08002888812 or',
 '08002986030',
 '08002986906',
 '08002988890',
 '08002988890 now',
 '08006344447',
 '08006344447 to',
 '0808',
 '0808 145',
 '08081263000',
 '08081263000 to',
 '08081560665',
 '08081560665 and',
 '0825',
 '0825 now',
 '083',
 '083 6089',
 '0844',
 '0844 861',
 '08448350055',
 '08448350055 from',
 '08448714184',
 '08448714184 stop',
 '0845',
 '0845 021',
 '0845 2814032',
 '08450542832',
 '08452810071',
 '08452810071 16',
 '08452810073',
 '08452810073 for',
 '08452810075over18',
 '0870',
 '0870 chatlines',
 '0870 is',
 '08700435505150p',
 '08700469649',
 '08700469649 po',
 '08700621170150p',
 '08700621170150p per',
 '08701213186',
 '08701237397',
 '08701237397 you',
 '08701417012',
 '08701417012 profit',
 '08701417012150p',
 '08701417012150p per',
 '0870141701216',
 '0870141701216 norm',
 '087016248',
 '08701752560',
 '08701752560 450p',
 '087018728737',
 '087018728737 national',
 '0870241182716',
 '0870241182716 wk',
 '08702490080',
 '08702490080 tells',
 '08702840625',
 '08702840625 comuk',
 '08704050406',
 '08704439680',
 '08704439680 when',
 '08704439680ts',
 '08704439680ts cs',
 '08706091795',
 '0870737910216yrs',
 '0870737910216yrs only',
 '08707500020',
 '08707500020 just',
 '08707509020',
 '08707509020 just',
 '0870753331018',
 '08707808226',
 '08708034412',
 '08708800282',
 '08708800282 hg',
 '08709222922',
 '08709222922 national',
 '08709501522',
 '08709501522 for',
 '0871',
 '0871 4719',
 '0871 872',
 '087104711148',
 '087104711148 now',
 '08712101358',
 '08712101358 now',
 '08712103738',
 '08712103738 now',
 '0871212025016',
 '08712300220',
 '08712300220 quoting',
 '08712300220 to',
 '087123002209am',
 '087123002209am 7pm',
 '08712317606',
 '08712317606 msg150p',
 '08712317606 stop',
 '08712400200',
 '08712400602450p',
 '08712400602450p provided',
 '08712400603',
 '08712402050',
 '08712402050 before',
 '08712402578',
 '08712402578 immediately',
 '08712402779',
 '08712402779 immediately',
 '08712402902',
 '08712402902 immediately',
 '08712402972',
 '08712402972 immediately',
 '08712404000',
 '08712404000 immediately',
 '08712405020',
 '08712405022',
 '08712405022 1x150p',
 '08712460324',
 '08712460324 10p',
 '08712460324 nat',
 '08712466669',
 '08712466669 at',
 '0871277810710p',
 '0871277810710p min',
 '0871277810810',
 '0871277810910p',
 '0871277810910p min',
 '08714342399',
 '08714342399 2stop',
 '087147123779am',
 '087147123779am 7pm',
 '08714712379',
 '08714712379 between',
 '08714712388',
 '08714712388 between',
 '08714712394',
 '08714712394 between',
 '08714712412',
 '08714712412 between',
 '08714714011',
 '08715203028',
 '08715203028 to',
 '08715203649',
 '08715203649 identifier',
 '08715203652',
 '08715203652 identifier',
 '08715203656',
 '08715203656 identifier',
 '08715203677',
 '08715203677 identifier',
 '08715203685',
 '08715203685 identifier',
 '08715203694',
 '08715203694 identifier',
 '08715205273',
 '08715500022',
 '08715500022 rpl',
 '08715705022',
 '08715705022 1x150p',
 '08717111821',
 '08717168528',
 '08717205546',
 '0871750',
 '0871750 77',
 '08717507382',
 '08717507382 now',
 '08717509990',
 '08717509990 poly',
 '08717890890å',
 '08717890890å 50',
 '08717895698',
 '08717895698 now',
 '08717898035',
 '08717898035 00',
 '08718711108',
 '08718720201',
 '08718720201 hg',
 '08718720201 po',
 '08718723815',
 '08718725756',
 '08718725756 140ppm',
 '08718726270',
 '087187262701',
 '087187262701 50gbp',
 '08718726970',
 '08718726970 now',
 '08718726971',
 '08718726971 now',
 '08718726978',
 '08718726978 now',
 '087187272008',
 '087187272008 now1',
 '08718727868',
 '08718727868 over',
 '08718727870',
 '08718727870150ppm',
 '08718730555',
 '08718730555 just',
 '08718730666',
 '08718730666 10p',
 '08718738001',
 '08718738001 identifier',
 '08718738002',
 '08718738002 identifier',
 '08718738034',
 '08719180219',
 '08719180219 identifier',
 '08719180248',
 '08719180248 identifier',
 '08719181259',
 '08719181259 identifier',
 '08719181503',
 '08719181513',
 '08719839835',
 '08719839835 future',
 '08719899217',
 '08719899217 identifier',
 '08719899229',
 '08719899229 identifier',
 '08719899230',
 '08719899230 identifier',
 '09',
 '09 02',
 '09 03',
 '09041940223',
 '09041940223 to',
 '09050000301',
 '09050000332',
 '09050000332 to',
 '09050000460',
 '09050000460 from',
 '09050000555',
 '09050000555 ba128nnfwfly150ppm',
 '09050000878',
 '09050000878 pobox45w2tg150p',
 '09050000928',
 '09050000928 pobox45w2tg150p',
 '09050001295',
 '09050001295 from',
 '09050001808',
 '09050001808 from',
 '09050002311',
 '09050002311 b4280703',
 '09050003091',
 '09050003091 from',
 '09050005321',
 '09050090044',
 '09050090044 now',
 '09050280520',
 '09050280520 to',
 '09053750005',
 '09053750005 b4',
 '09056242159',
 '09056242159 to',
 '09057039994',
 '09058091854',
 '09058091854 now',
 '09058091870',
 '09058091870 now',
 '09058094454',
 '09058094454 from',
 '09058094455',
 '09058094455 from',
 '09058094507',
 '09058094507 from',
 '09058094565',
 '09058094565 from',
 '09058094583',
 '09058094583 to',
 '09058094594',
 '09058094597',
 '09058094599',
 '09058095107',
 '09058095107 now',
 '09058095201',
 '09058095201 from',
 '09058097189',
 '09058097189 now',
 '09058097218',
 '09058097218 to',
 '09058098002',
 '09058098002 pobox1',
 '09058099801',
 '09058099801 b4190604',
 '09061104276',
 '09061104276 to',
 '09061104283',
 '09061104283 ts',
 '09061209465',
 '09061209465 now',
 '09061213237',
 '09061213237 from',
 '09061221061',
 '09061221061 from',
 '09061221066',
 '09061221066 fromm',
 '09061701444',
 '09061701444 valid',
 '09061701461',
 '09061701461 claim',
 '09061701851',
 '09061701851 claim',
 '09061701939',
 '09061701939 claim',
 '09061702893',
 '09061702893 acl03530150pm',
 '09061743386',
 '09061743806',
 '09061743806 from',
 '09061743810',
 '09061743810 from',
 '09061743811',
 '09061743811 from',
 '09061744553',
 '09061744553 now',
 '09061749602',
 '09061749602 from',
 '09061790121',
 '09061790121 from',
 '09061790125',
 '09061790125 from',
 '09061790126',
 '09061790126 from',
 '09063440451',
 '09063440451 from',
 '09063442151',
 '09063442151 to',
 '09063458130',
 '09063458130 now',
 '0906346330',
 '0906346330 your',
 '09064011000',
 '09064011000 ntt',
 '09064012103',
 '09064012103 box334sk38ch',
 '09064012160',
 '09064012160 claim',
 '09064015307',
 '09064015307 box334sk38ch',
 '09064017295',
 '09064017295 claim',
 '09064017305',
 '09064017305 pobox75ldns7',
 '09064018838',
 '09064018838 ntt',
 '09064019014',
 '09064019014 to',
 '09064019788',
 '09064019788 box42wr29c',
 '09065069120',
 '09065069154',
 '09065171142',
 '09065171142 stopsms',
 '09065174042',
 '09065174042 to',
 '09065394514',
 '09065394514 from',
 '09065394973',
 '09065394973 from',
 '09065989180',
 '09065989182',
 '09065989182 from',
 '09066350750',
 '09066350750 from',
 '09066358152',
 '09066358152 to',
 '09066358361',
 '09066358361 from',
 '09066361921',
 '09066362206',
 '09066362206 asap',
 '09066362220',
 '09066362220 asap',
 '09066362231',
 '09066362231 asap',
 '09066362231 urgent',
 '09066364311',
 '09066364311 to',
 '09066364349',
 '09066364349 now',
 '09066364589',
 '09066368327',
 '09066368327 now',
 '09066368470',
 '09066368753',
 '09066368753 asap',
 '09066380611',
 '09066382422',
 '09066382422 calls',
 '09066612661',
 '09066612661 from',
 '09066649731from',
 '09066649731from landline',
 '09066660100',
 '09066660100 now',
 '09071512432',
 '09071512432 b4',
 '09071512433',
 '09071512433 b4',
 '09071517866',
 '09071517866 now',
 '09077818151',
 '09077818151 to',
 '09090204448',
 '09090204448 and',
 '09090900040',
 '09090900040 listen',
 '09094100151',
 '09094100151 to',
 '09094646631',
 '09094646631 just',
 '09094646899',
 '09094646899 now',
 '09095350301',
 '09095350301 and',
 '09096102316',
 '09096102316 can',
 '09099725823',
 '09099725823 hope',
 '09099726395',
 '09099726395 lucy',
 '09099726429',
 '09099726429 janinexx',
 '09099726481',
 '09099726481 luv',
 '09099726553',
 '09099726553 reply',
 '09111030116',
 '09111030116 pobox12n146tf15',
 '09111032124',
 '09111032124 pobox12n146tf150p',
 '09701213186',
 '0a',
 '0a networks',
 '0quit',
 '0quit edrunk',
 '10',
 '10 000',
 '10 04',
 '10 06',
 '10 10',
 '10 1mega',
 '10 30',
 '10 at',
 '10 days',
 '10 den',
 '10 did',
 '10 free',
 '10 kilos',
 '10 ls1',
 '10 man',
 '10 min',
 '10 more',
 '10 mths',
 '10 pages',
 '10 smth',
 '10 to',
 '100',
 '100 000',
 '100 cash',
 '100 dating',
 '100 free',
 '100 gift',
 '100 high',
 '100 of',
 '100 percent',
 '100 to',
 '100 travel',
 '100 txts',
 '100 weekly',
 '100 wkly',
 '1000',
 '1000 cash',
 '1000 cashto',
 '1000 flirting',
 '1000 in',
 '1000 of',
 '1000 or',
 '1000 prize',
 '1000 to',
 '1000 txt',
 '1000 txts',
 '1000 winner',
 '1000call',
 '1000call 09071512432',
 '1000s',
 '1000s choose',
 '1000s of',
 '100p',
 '100p sms',
 '100percent',
 '100percent real',
 '100txt',
 '100txt mth',
 '1013',
 '1013 ig11',
 '1030',
 '1030 there',
 '1030 to',
 '10am',
 '10am 7pm',
 '10am 9pm',
 '10am till',
 '10k',
 '10k 5k',
 '10k cash',
 '10p',
 '10p min',
 '10p per',
 '10p reply',
 '10ppm',
 '10ppm 16',
 '10th',
 '10th sept',
 '11',
 '11 04',
 '11 48',
 '11 bt',
 '11 mnths',
 '11 months',
 '11 ok',
 '11 then',
 '11 tmr',
 '1120',
 '1120 to',
 '113',
 '113 bray',
 '1131',
 '1131 standard',
 '114',
 '114 14',
 '116',
 '116 but',
 '1172',
 '1172 for',
 '118p',
 '118p msg',
 '11mths',
 '11mths call',
 '11mths update',
 '11mths you',
 '11pm',
 '11pm as',
 '12',
 '12 000pes',
 '12 30',
 '12 anyway',
 '12 help',
 '12 hours',
 '12 kiosk',
 '12 mths',
 '12 rite',
 '1205',
 '1205 one',
 '120p',
 '121',
 '121 chat',
 '1225',
 '1225 are',
 '123',
 '123 congratulations',
 '125',
 '125 gift',
 '1250',
 '1250 call',
 '125gift',
 '125gift guaranteed',
 '128',
 '128 mb',
 '12hours',
 '12hours only',
 '12hrs',
 '12hrs 150p',
 '12hrs only',
 '12mths',
 '12mths 2price',
 '12mths half',
 '13',
 '13 04',
 '13 10',
 '130',
 '130 iriver',
 '1327',
 '1327 croydon',
 '139',
 '139 la3',
 '14',
 '14 tcr',
 '140',
 '140 ard',
 '1405',
 '1405 1680',
 '140ppm',
 '145',
 '145 4742',
 '1450',
 '1450 prize',
 '146tf150p',
 '14tcr',
 '14tcr w1',
 '14thmarch',
 '14thmarch apply',
 '15',
 '15 26',
 '15 as',
 '15 cos',
 '150',
 '150 ppm',
 '150 prize',
 '150 sae',
 '150 text',
 '150 textand',
 '150 voucher',
 '150 worth',
 '1500',
 '1500 bonus',
 '150p',
 '150p 08712400603',
 '150p 18',
 '150p daily',
 '150p day',
 '150p inc',
 '150p meg',
 '150p min',
 '150p msg',
 '150p msgrcvd',
 '150p msgrcvdhg',
 '150p mt',
 '150p mtmsg',
 '150p mtmsgrcvd18',
 '150p netcollex',
 '150p per',
 '150p pm',
 '150p poly',
 '150p rcvd',
 '150p reply',
 '150p sms',
 '150p stop',
 '150p text',
 '150p textoperator',
 '150p tone',
 '150p wk',
 '150p16',
 '150pm',
 '150pm dont',
 '150pm to',
 '150ppermesssubscription',
 '150ppm',
 '150ppm 16',
 '150ppm 18',
 '150ppm ave',
 '150ppm mobile',
 '150ppm mobiles',
 '150ppm mobilesvary',
 '150ppmpobox10183bhamb64xe',
 '150ppmsg',
 '150ppmsg 18',
 '150pw',
 '150pw to',
 '151',
 '151 to',
 '153',
 '153 9153',
 '153 9996',
 '15541',
 '15pm',
 '15pm to',
 '16',
 '16 118p',
 '16 after',
 '16 close',
 '16 club',
 '16 cs',
 '16 gbp1',
 '16 may',
 '16 norm150p',
 '16 only',
 '16 på',
 '16 remove',
 '16 reply',
 '16 sn',
 '16 stop',
 '16 tsandcs',
 '16 unsub',
 '16 wk',
 '16 ûï',
 '165',
 '165 or',
 '165 see',
 '1680',
 '1680 1843',
 '169',
 '169 6031',
 '177',
 '177 m227xy',
 '18',
 '18 11',
 '18 150p',
 '18 30pp',
 '18 50',
 '18 bt',
 '18 content',
 '18 days',
 '18 its',
 '18 msg',
 '18 only',
 '18 sender',
 '18 stop',
 '18 to',
 '18 www',
 '18 xxx',
 '18 years',
 '18 yrs',
 '180',
 '180 at',
 '1843',
 '1843 all',
 '18p',
 '18p txt',
 '18yrs',
 '195',
 '195 6669',
 '1956669',
 '1956669 or',
 '1apple',
 '1apple day',
 '1b6a5ecef91ff9',
 '1b6a5ecef91ff9 37819',
 '1cup',
 '1cup milk',
 '1da',
 '1da 150ppmsg',
 '1er',
 '1er until',
 '1hr',
 '1hr time',
 '1im',
 '1im talkin',
 '1lemon',
 '1lemon day',
 '1mega',
 '1mega pixels',
 '1million',
 '1million to',
 '1pm',
 '1pm orchard',
 '1st',
 '1st 5free',
 '1st 5wkg',
 '1st class',
 '1st free',
 '1st get',
 '1st june',
 '1st lor',
 '1st ringtone',
 '1st salary',
 '1st sept',
 '1st then',
 '1st tone',
 '1st ur',
 '1st wat',
 '1st week',
 '1st wk',
 '1st4terms',
 '1st4terms pobox84',
 '1stchoice',
 '1stchoice co',
 '1stone',
 '1stone sun',
 '1thing',
 '1thing got',
 '1tulsi',
 '1tulsi leaf',
 '1win150ppmx3',
 '1winaweek',
 '1winaweek age16',
 '1winawk',
 '1winawk age16',
 '1x150p',
 '1x150p wk',
 '1yf',
 '1yf 150ppm',
 '20',
 '20 000',
 '20 is',
 '20 its',
 '20 mins',
 '20 off',
 '20 photo',
 '20 poboxox36504w45wq',
 '20 that',
 '20 tones',
 '20 years',
 '200',
 '200 award',
 '200 free',
 '200 prize',
 '200 shopping',
 '200 summer',
 '200 this',
 '2000',
 '2000 award',
 '2000 bonus',
 '2000 cash',
 '2000 gift',
 '2000 plus',
 '2000 pound',
 '2000 prize',
 '2003',
 '2003 account',
 '2004',
 '2004 account',
 '2004 must',
 '2004 offer',
 '2005',
 '2005 text',
 '2006',
 '2006 fifa',
 '2007',
 '2007 name',
 '2007 uk',
 '200p',
 '200p 16',
 '2025050',
 '2025050 or',
 '20m12aq',
 '20m12aq 150ppm',
 '20p',
 '20p min',
 '20p per',
 '21',
 '21 11',
 '21 from',
 '21 is',
 '21 matches',
 '21 may',
 '21870000',
 '21870000 hi',
 '21st',
 '21st instead',
 '21st may',
 '22',
 '22 65',
 '22 days',
 '220',
 '220 cm2',
 '220cm2',
 '220cm2 9ae',
 '2309',
 '23f',
 '23f for',
 '23g',
 ...]
```

In [143]:

```
len(ngrams_id)
```

Out[143]:

```
50326
```

In [131]:

```
type(ngrams)
```

Out[131]:

```
scipy.sparse.csr.csr_matrix
```

In [146]:

```
ngrams = ngrams.toarray()
```

In [147]:

```
type(ngrams)
```

Out[147]:

```
numpy.ndarray
```

In [148]:

```
ngrams.shape
```

Out[148]:

```
(5572, 50326)
```

In [149]:

```
sms.shape
```

Out[149]:

```
(5572, 2)
```

In [152]:

```
ngrams_df = pd.DataFrame(data = ngrams, columns = ngrams_id)
ngrams_df.shape
```

Out[152]:

```
(5572, 50326)
```

In [154]:

```
ngrams_df.iloc[:,1:5].head()
```

Out[154]:

	00 in	00 per	00 sub	00 subs
0	0	0	0	0
1	0	0	0	0
2	0	0	0	0
3	0	0	0	0
4	0	0	0	0

In [155]:

```
x_sms_train, x_sms_test, y_sms_train, y_sms_test = train_test_split(ngrams_df,
                                                                     sms['cat'],
                                                                     test_size =
0.3,
                                                                     random_state
= 198)
```

Apply SVM
first linear kernel
then polynomial kernel

In [156]:

```
from sklearn.svm import SVC
```

In [157]:

```
svm_linear = SVC(kernel='linear')

svm_linear.fit( x_sms_train , y_sms_train)
```

Out[157]:

```
SVC(C=1.0, cache_size=200, class_weight=None, coef0=0.0,
    decision_function_shape='ovr', degree=3, gamma='auto_deprecate
d',
    kernel='linear', max_iter=-1, probability=False, random_state=No
ne,
    shrinking=True, tol=0.001, verbose=False)
```

In [158]:

```
svm_linear.support_vectors_
```

Out[158]:

```
array([[0., 0., 0., ..., 0., 0., 0.],
       [0., 0., 0., ..., 0., 0., 0.],
       [0., 0., 0., ..., 0., 0., 0.],
       ...,
       [0., 0., 0., ..., 0., 0., 0.],
       [0., 0., 0., ..., 0., 0., 0.],
       [0., 0., 0., ..., 0., 0., 0.]])
```

In [159]:

```
svm_linear.n_support_
```

Out[159]:

```
array([715, 271], dtype=int32)
```

In [161]:

```
svm_linear_pred = svm_linear.predict(x_sms_test)
```

In [162]:

```
confusion_matrix(y_sms_test, svm_linear_pred)
```

Out[162]:

```
array([[1461,    0],
       [  30,  181]])
```

In [163]:

```
accuracy_score(y_sms_test, svm_linear_pred)
```

Out[163]:

```
0.9820574162679426
```

In [164]:

```
precision_score(y_sms_test, svm_linear_pred)
```

Out[164]:

```
1.0
```

In [165]:

```
recall_score(y_sms_test, svm_linear_pred)
```

Out[165]:

```
0.8578199052132701
```

In [166]:

```
svm_poly = SVC(kernel = 'poly', degree = 3)
svm_poly.fit( x_sms_train , y_sms_train)
```

```
/Library/Frameworks/Python.framework/Versions/3.8/lib/python3.8/site
-packages/sklearn/svm/base.py:189: FutureWarning: The default value
of gamma will change from 'auto' to 'scale' in version 0.22 to accou
nt better for unscaled features. Set gamma explicitly to 'auto' or
'scale' to avoid this warning.
  warnings.warn("The default value of gamma will change "
```

Out[166]:

```
SVC(C=1.0, cache_size=200, class_weight=None, coef0=0.0,
    decision_function_shape='ovr', degree=3, gamma='auto_deprecate
d',
    kernel='poly', max_iter=-1, probability=False, random_state=Non
e,
    shrinking=True, tol=0.001, verbose=False)
```

In [167]:

```
svm_poly.n_support_
```

Out[167]:

```
array([536, 536], dtype=int32)
```

In [168]:

```
svm_poly_pred = svm_poly.predict(x_sms_test)
```

In [ ]:

```
confusion_matrix(y_sms_test, svm_poly_pred)
```

In [ ]:

```
accuracy_score(y_sms_test, svm_poly_pred)
```

In [ ]:

```
precision_score(y_sms_test, svm_poly_pred)
```

In [ ]:

```
recall_score(y_sms_test, svm_poly_pred)
```

Now try grid search to hyper tune polynomial svm's parameter (hyperparameters)
possible parameteres include
degree
gamma
C(regularisation parameter)
kernel

In [169]:

```
from sklearn.model_selection import GridSearchCV

param_grid = [
    {'kernel': ['poly','rbf'], 'degree': [2,3,4,7,8]}
]

grid = GridSearchCV(SVC(), param_grid, cv = 3)
```

In [170]:

```
grid.fit(x_sms_train, y_sms_train)
```

```
/Library/Frameworks/Python.framework/Versions/3.8/lib/python3.8/site
-packages/sklearn/svm/base.py:189: FutureWarning: The default value
of gamma will change from 'auto' to 'scale' in version 0.22 to accou
nt better for unscaled features. Set gamma explicitly to 'auto' or
'scale' to avoid this warning.
  warnings.warn("The default value of gamma will change "
/Library/Frameworks/Python.framework/Versions/3.8/lib/python3.8/site
-packages/sklearn/svm/base.py:189: FutureWarning: The default value
of gamma will change from 'auto' to 'scale' in version 0.22 to accou
nt better for unscaled features. Set gamma explicitly to 'auto' or
'scale' to avoid this warning.
  warnings.warn("The default value of gamma will change "
/Library/Frameworks/Python.framework/Versions/3.8/lib/python3.8/site
-packages/sklearn/svm/base.py:189: FutureWarning: The default value
of gamma will change from 'auto' to 'scale' in version 0.22 to accou
nt better for unscaled features. Set gamma explicitly to 'auto' or
'scale' to avoid this warning.
  warnings.warn("The default value of gamma will change "
/Library/Frameworks/Python.framework/Versions/3.8/lib/python3.8/site
-packages/sklearn/svm/base.py:189: FutureWarning: The default value
of gamma will change from 'auto' to 'scale' in version 0.22 to accou
nt better for unscaled features. Set gamma explicitly to 'auto' or
'scale' to avoid this warning.
  warnings.warn("The default value of gamma will change "
/Library/Frameworks/Python.framework/Versions/3.8/lib/python3.8/site
-packages/sklearn/svm/base.py:189: FutureWarning: The default value
of gamma will change from 'auto' to 'scale' in version 0.22 to accou
nt better for unscaled features. Set gamma explicitly to 'auto' or
'scale' to avoid this warning.
  warnings.warn("The default value of gamma will change "
/Library/Frameworks/Python.framework/Versions/3.8/lib/python3.8/site
-packages/sklearn/svm/base.py:189: FutureWarning: The default value
of gamma will change from 'auto' to 'scale' in version 0.22 to accou
nt better for unscaled features. Set gamma explicitly to 'auto' or
'scale' to avoid this warning.
  warnings.warn("The default value of gamma will change "
/Library/Frameworks/Python.framework/Versions/3.8/lib/python3.8/site
-packages/sklearn/svm/base.py:189: FutureWarning: The default value
of gamma will change from 'auto' to 'scale' in version 0.22 to accou
nt better for unscaled features. Set gamma explicitly to 'auto' or
'scale' to avoid this warning.
  warnings.warn("The default value of gamma will change "
/Library/Frameworks/Python.framework/Versions/3.8/lib/python3.8/site
-packages/sklearn/svm/base.py:189: FutureWarning: The default value
of gamma will change from 'auto' to 'scale' in version 0.22 to accou
nt better for unscaled features. Set gamma explicitly to 'auto' or
'scale' to avoid this warning.
  warnings.warn("The default value of gamma will change "
/Library/Frameworks/Python.framework/Versions/3.8/lib/python3.8/site
-packages/sklearn/svm/base.py:189: FutureWarning: The default value
of gamma will change from 'auto' to 'scale' in version 0.22 to accou
nt better for unscaled features. Set gamma explicitly to 'auto' or
'scale' to avoid this warning.
  warnings.warn("The default value of gamma will change "
/Library/Frameworks/Python.framework/Versions/3.8/lib/python3.8/site
-packages/sklearn/svm/base.py:189: FutureWarning: The default value
of gamma will change from 'auto' to 'scale' in version 0.22 to accou
nt better for unscaled features. Set gamma explicitly to 'auto' or
'scale' to avoid this warning.
  warnings.warn("The default value of gamma will change "
/Library/Frameworks/Python.framework/Versions/3.8/lib/python3.8/site
```

```
-packages/sklearn/svm/base.py:189: FutureWarning: The default value
of gamma will change from 'auto' to 'scale' in version 0.22 to accou
nt better for unscaled features. Set gamma explicitly to 'auto' or
'scale' to avoid this warning.
  warnings.warn("The default value of gamma will change "
/Library/Frameworks/Python.framework/Versions/3.8/lib/python3.8/site
-packages/sklearn/svm/base.py:189: FutureWarning: The default value
of gamma will change from 'auto' to 'scale' in version 0.22 to accou
nt better for unscaled features. Set gamma explicitly to 'auto' or
'scale' to avoid this warning.
  warnings.warn("The default value of gamma will change "
/Library/Frameworks/Python.framework/Versions/3.8/lib/python3.8/site
-packages/sklearn/svm/base.py:189: FutureWarning: The default value
of gamma will change from 'auto' to 'scale' in version 0.22 to accou
nt better for unscaled features. Set gamma explicitly to 'auto' or
'scale' to avoid this warning.
  warnings.warn("The default value of gamma will change "
/Library/Frameworks/Python.framework/Versions/3.8/lib/python3.8/site
-packages/sklearn/svm/base.py:189: FutureWarning: The default value
of gamma will change from 'auto' to 'scale' in version 0.22 to accou
nt better for unscaled features. Set gamma explicitly to 'auto' or
'scale' to avoid this warning.
  warnings.warn("The default value of gamma will change "
/Library/Frameworks/Python.framework/Versions/3.8/lib/python3.8/site
-packages/sklearn/svm/base.py:189: FutureWarning: The default value
of gamma will change from 'auto' to 'scale' in version 0.22 to accou
nt better for unscaled features. Set gamma explicitly to 'auto' or
'scale' to avoid this warning.
  warnings.warn("The default value of gamma will change "
/Library/Frameworks/Python.framework/Versions/3.8/lib/python3.8/site
-packages/sklearn/svm/base.py:189: FutureWarning: The default value
of gamma will change from 'auto' to 'scale' in version 0.22 to accou
nt better for unscaled features. Set gamma explicitly to 'auto' or
'scale' to avoid this warning.
  warnings.warn("The default value of gamma will change "
/Library/Frameworks/Python.framework/Versions/3.8/lib/python3.8/site
-packages/sklearn/svm/base.py:189: FutureWarning: The default value
of gamma will change from 'auto' to 'scale' in version 0.22 to accou
nt better for unscaled features. Set gamma explicitly to 'auto' or
'scale' to avoid this warning.
  warnings.warn("The default value of gamma will change "
/Library/Frameworks/Python.framework/Versions/3.8/lib/python3.8/site
-packages/sklearn/svm/base.py:189: FutureWarning: The default value
of gamma will change from 'auto' to 'scale' in version 0.22 to accou
nt better for unscaled features. Set gamma explicitly to 'auto' or
'scale' to avoid this warning.
  warnings.warn("The default value of gamma will change "
/Library/Frameworks/Python.framework/Versions/3.8/lib/python3.8/site
-packages/sklearn/svm/base.py:189: FutureWarning: The default value
of gamma will change from 'auto' to 'scale' in version 0.22 to accou
nt better for unscaled features. Set gamma explicitly to 'auto' or
'scale' to avoid this warning.
  warnings.warn("The default value of gamma will change "
/Library/Frameworks/Python.framework/Versions/3.8/lib/python3.8/site
-packages/sklearn/svm/base.py:189: FutureWarning: The default value
of gamma will change from 'auto' to 'scale' in version 0.22 to accou
nt better for unscaled features. Set gamma explicitly to 'auto' or
'scale' to avoid this warning.
  warnings.warn("The default value of gamma will change "
/Library/Frameworks/Python.framework/Versions/3.8/lib/python3.8/site
-packages/sklearn/svm/base.py:189: FutureWarning: The default value
```

```
of gamma will change from 'auto' to 'scale' in version 0.22 to accou
nt better for unscaled features. Set gamma explicitly to 'auto' or
'scale' to avoid this warning.
  warnings.warn("The default value of gamma will change "
/Library/Frameworks/Python.framework/Versions/3.8/lib/python3.8/site
-packages/sklearn/svm/base.py:189: FutureWarning: The default value
of gamma will change from 'auto' to 'scale' in version 0.22 to accou
nt better for unscaled features. Set gamma explicitly to 'auto' or
'scale' to avoid this warning.
  warnings.warn("The default value of gamma will change "
/Library/Frameworks/Python.framework/Versions/3.8/lib/python3.8/site
-packages/sklearn/svm/base.py:189: FutureWarning: The default value
of gamma will change from 'auto' to 'scale' in version 0.22 to accou
nt better for unscaled features. Set gamma explicitly to 'auto' or
'scale' to avoid this warning.
  warnings.warn("The default value of gamma will change "
/Library/Frameworks/Python.framework/Versions/3.8/lib/python3.8/site
-packages/sklearn/svm/base.py:189: FutureWarning: The default value
of gamma will change from 'auto' to 'scale' in version 0.22 to accou
nt better for unscaled features. Set gamma explicitly to 'auto' or
'scale' to avoid this warning.
  warnings.warn("The default value of gamma will change "
/Library/Frameworks/Python.framework/Versions/3.8/lib/python3.8/site
-packages/sklearn/svm/base.py:189: FutureWarning: The default value
of gamma will change from 'auto' to 'scale' in version 0.22 to accou
nt better for unscaled features. Set gamma explicitly to 'auto' or
'scale' to avoid this warning.
  warnings.warn("The default value of gamma will change "
/Library/Frameworks/Python.framework/Versions/3.8/lib/python3.8/site
-packages/sklearn/svm/base.py:189: FutureWarning: The default value
of gamma will change from 'auto' to 'scale' in version 0.22 to accou
nt better for unscaled features. Set gamma explicitly to 'auto' or
'scale' to avoid this warning.
  warnings.warn("The default value of gamma will change "
/Library/Frameworks/Python.framework/Versions/3.8/lib/python3.8/site
-packages/sklearn/svm/base.py:189: FutureWarning: The default value
of gamma will change from 'auto' to 'scale' in version 0.22 to accou
nt better for unscaled features. Set gamma explicitly to 'auto' or
'scale' to avoid this warning.
  warnings.warn("The default value of gamma will change "
/Library/Frameworks/Python.framework/Versions/3.8/lib/python3.8/site
-packages/sklearn/svm/base.py:189: FutureWarning: The default value
of gamma will change from 'auto' to 'scale' in version 0.22 to accou
nt better for unscaled features. Set gamma explicitly to 'auto' or
'scale' to avoid this warning.
  warnings.warn("The default value of gamma will change "
/Library/Frameworks/Python.framework/Versions/3.8/lib/python3.8/site
-packages/sklearn/svm/base.py:189: FutureWarning: The default value
of gamma will change from 'auto' to 'scale' in version 0.22 to accou
nt better for unscaled features. Set gamma explicitly to 'auto' or
'scale' to avoid this warning.
  warnings.warn("The default value of gamma will change "
/Library/Frameworks/Python.framework/Versions/3.8/lib/python3.8/site
-packages/sklearn/svm/base.py:189: FutureWarning: The default value
of gamma will change from 'auto' to 'scale' in version 0.22 to accou
nt better for unscaled features. Set gamma explicitly to 'auto' or
'scale' to avoid this warning.
  warnings.warn("The default value of gamma will change "
/Library/Frameworks/Python.framework/Versions/3.8/lib/python3.8/site
-packages/sklearn/svm/base.py:189: FutureWarning: The default value
of gamma will change from 'auto' to 'scale' in version 0.22 to accou
```

```
nt better for unscaled features. Set gamma explicitly to 'auto' or
'scale' to avoid this warning.
  warnings.warn("The default value of gamma will change "
```

Out[170]:

```
GridSearchCV(cv=3, error_score='raise-deprecating',
             estimator=SVC(C=1.0, cache_size=200, class_weight=None,
coef0=0.0,
                           decision_function_shape='ovr', degree=3,
                           gamma='auto_deprecated', kernel='rbf', ma
x_iter=-1,
                           probability=False, random_state=None, shr
inking=True,
                           tol=0.001, verbose=False),
             iid='warn', n_jobs=None,
             param_grid=[{'degree': [2, 3, 4, 7, 8],
                          'kernel': ['poly', 'rbf']}],
             pre_dispatch='2*n_jobs', refit=True, return_train_score
=False,
             scoring=None, verbose=0)
```

In [171]:

```
grid.best_params_
```

Out[171]:

```
{'degree': 2, 'kernel': 'poly'}
```

In [172]:

```
grid.cv_results_
```

Out[172]:

```
{'mean_fit_time': array([ 92.37450782,  97.9618245 , 274.70143954, 5
10.02646112,
        158.80167103, 145.29603648,  87.33926868,  94.94347684,
        705.04820458, 106.51352771]),
 'std_fit_time': array([2.96083596e-01, 1.75040287e+00, 2.63289147e+
02, 4.43279024e+02,
        1.00953928e+02, 7.13039557e+01, 2.61493287e-01, 1.36672163e+
00,
        8.74177963e+02, 9.50814588e-01]),
 'mean_score_time': array([ 44.70128846,  47.63206895, 400.54052504,
793.817662  ,
        797.2882603 ,  45.31920465,  42.75135652,  45.34162696,
         44.96676731,  51.17960437]),
 'std_score_time': array([2.34961420e-01, 8.71806758e-01, 5.05794394
e+02, 1.05843377e+03,
        1.06721127e+03, 1.48485699e-01, 1.18387215e-01, 4.25620832e-
01,
        2.75228631e+00, 4.33852389e-01]),
 'param_degree': masked_array(data=[2, 2, 3, 3, 4, 4, 7, 7, 8, 8],
              mask=[False, False, False, False, False, False, False,
False,
                    False, False],
        fill_value='?',
             dtype=object),
 'param_kernel': masked_array(data=['poly', 'rbf', 'poly', 'rbf', 'p
oly', 'rbf', 'poly',
                    'rbf', 'poly', 'rbf'],
              mask=[False, False, False, False, False, False, False,
False,
                    False, False],
        fill_value='?',
             dtype=object),
 'params': [{'degree': 2, 'kernel': 'poly'},
  {'degree': 2, 'kernel': 'rbf'},
  {'degree': 3, 'kernel': 'poly'},
  {'degree': 3, 'kernel': 'rbf'},
  {'degree': 4, 'kernel': 'poly'},
  {'degree': 4, 'kernel': 'rbf'},
  {'degree': 7, 'kernel': 'poly'},
  {'degree': 7, 'kernel': 'rbf'},
  {'degree': 8, 'kernel': 'poly'},
  {'degree': 8, 'kernel': 'rbf'}],
 'split0_test_score': array([0.86241353, 0.86241353, 0.86241353, 0.8
6241353, 0.86241353,
        0.86241353, 0.86241353, 0.86241353, 0.86241353, 0.8624135
3]),
 'split1_test_score': array([0.86230769, 0.86230769, 0.86230769, 0.8
6230769, 0.86230769,
        0.86230769, 0.86230769, 0.86230769, 0.86230769, 0.8623076
9]),
 'split2_test_score': array([0.86297152, 0.86297152, 0.86297152, 0.8
6297152, 0.86297152,
        0.86297152, 0.86297152, 0.86297152, 0.86297152, 0.8629715
2]),
 'mean_test_score': array([0.8625641, 0.8625641, 0.8625641, 0.862564
1, 0.8625641, 0.8625641,
        0.8625641, 0.8625641, 0.8625641, 0.8625641]),
 'std_test_score': array([0.00029114, 0.00029114, 0.00029114, 0.0002
9114, 0.00029114,
        0.00029114, 0.00029114, 0.00029114, 0.00029114, 0.0002911
4]),
 'rank_test_score': array([1, 1, 1, 1, 1, 1, 1, 1, 1, 1], dtype=int3
2)}
```

In [173]:

```
grid.best_estimator_.degree
```

Out[173]:

```
2
```

In [180]:

```
svm_grid_pred = grid.predict(x_sms_test)
```

In [181]:

```
confusion_matrix(y_sms_test, svm_grid_pred)
```

Out[181]:

```
array([[1461,    0],
       [ 211,    0]])
```

In [182]:

```
accuracy_score(y_sms_test, svm_grid_pred)
```

Out[182]:

```
0.8738038277511961
```

In [177]:

```
precision_score(y_sms_test, svm_grid_pred)
```

```
/Library/Frameworks/Python.framework/Versions/3.8/lib/python3.8/site
-packages/sklearn/metrics/classification.py:1436: UndefinedMetricWar
ning: Precision is ill-defined and being set to 0.0 due to no predic
ted samples.
  precision = _prf_divide(tp_sum, pred_sum,
```

Out[177]:

```
0.0
```

In [178]:

```
recall_score(y_sms_test, svm_grid_pred)
```

Out[178]:

```
0.0
```

In [179]:

```
recall_score(y_sms_test, svm_grid_pred)
```

Out[179]:

```
0.0
```

This is an interesting confusion matrix, nothing has been predicted positively, not even the actual positive instance.
try to think of the reason.

you are ready to compare NB and SVM

In [ ]:

```
```