ML Code with Feature Selection – Explanation

Recursive Feature Elimination

1. Importing required libraries

```
import pandas as pd
from sklearn.model_selection import train_test_split
import time
import numpy as np
from sklearn.preprocessing import StandardScaler
from sklearn.feature_selection import SelectKBest
from sklearn.feature_selection import chi2
from sklearn.feature_selection import RFE
from sklearn.linear_model import LogisticRegression
import pickle
import matplotlib.pyplot as plt
from sklearn.neighbors import KNeighborsClassifier
from sklearn.svm import SVC
from sklearn.ensemble import RandomForestClassifier
from sklearn.naive_bayes import GaussianNB
from sklearn.tree import DecisionTreeClassifier
```

2. Creating the function for RFE

```
def rfeFeature(indep_X,dep_Y,n):
        rfelist=[]

        log_model = LogisticRegression(solver='lbfgs')
        RF = RandomForestClassifier(n_estimators = 10, criterion = 'entropy', random_state = 0)
       # NB = GaussianNB()
        DT= DecisionTreeClassifier(criterion = 'gini', max_features='sqrt',splitter='best',random_state = 0)
        svc_model = SVC(kernel = 'linear', random_state = 0)
        #knn = KNeighborsClassifier(n_neighbors = 5, metric = 'minkowski', p = 2)
        rfemodellist=[log_model,svc_model,RF,DT]
        for i in   rfemodellist:
            print(i)
            log_rfe = RFE(i, n_features_to_select=n)
            log_fit = log_rfe.fit(indep_X, dep_Y)
            log_rfe_feature=log_fit.transform(indep_X)
            rfelist.append(log_rfe_feature)
        return rfelist
```

- created four different models to use inside Recursive Feature Elimination
- Stored all models in a list to Loops over them.
- RFE Loop

  For each model:

  - Prints which model is currently running.
  - Creates an RFE (Recursive Feature Elimination) object with that model and selects n features.
  - Fits RFE to your training data (indep_X, dep_Y).

- log_rfe = RFE(i, n_features_to_select=n) – In scikit-learn's documentation – it is mention to give the parameter in this format not just n.
- Transforms your dataset to only include the selected top n features.
- Appends the reduced feature set to the list rfelist.

3. Created Split scalar function

```
def split_scalar(indep_X,dep_Y):
        X_train, X_test, y_train, y_test = train_test_split(indep_X, dep_Y, test_size = 0.25, random_state = 0)
        #X_train, X_test, y_train, y_test = train_test_split(indep_X,dep_Y, test_size = 0.25, random_state = 0)

        #Feature Scaling
        #from sklearn.preprocessing import StandardScaler
        sc = StandardScaler()
        X_train = sc.fit_transform(X_train)
        X_test = sc.transform(X_test)

        return X_train, X_test, y_train, y_test
```

- **Splits data** into training (75%) and test (25%) sets.
- **Standardizes features** so they have mean = 0 and standard deviation = 1.
- Returns the scaled train/test data.

4. Created a function for prediction using confusion matrix

```
def cm_prediction(classifier,X_test):
     y_pred = classifier.predict(X_test)

        # Making the Confusion Matrix
     from sklearn.metrics import confusion_matrix
     cm = confusion_matrix(y_test, y_pred)

     from sklearn.metrics import accuracy_score
     from sklearn.metrics import classification_report
        #from sklearn.metrics import confusion_matrix
        #cm = confusion_matrix(y_test, y_pred)

     Accuracy=accuracy_score(y_test, y_pred )

     report=classification_report(y_test, y_pred)
     return  classifier,Accuracy,report,X_test,y_test,cm
```

- classifier.predict(X_test) – making predictions and storing to y_pred
- By using Confusion matrix – comparing y_test (true values) and y_pred values (predicted values)
- accuracy_score(y_test, y_pred) – calculating accuracy i.e. the ratio of correct predictions to total predictions.
- classification_report(y_test, y_pred) – generating classification report

5. Created functions for Algorithms

```
def logistic(X_train,y_train,X_test):
        # Fitting K-NN to the Training set
        from sklearn.linear_model import LogisticRegression
        classifier = LogisticRegression(random_state = 0)
        classifier.fit(X_train, y_train)
        classifier,Accuracy,report,X_test,y_test,cm=cm_prediction(classifier,X_test)
        return  classifier,Accuracy,report,X_test,y_test,cm

def svm_linear(X_train,y_train,X_test):

        from sklearn.svm import SVC
        classifier = SVC(kernel = 'linear', random_state = 0)
        classifier.fit(X_train, y_train)
        classifier,Accuracy,report,X_test,y_test,cm=cm_prediction(classifier,X_test)
        return  classifier,Accuracy,report,X_test,y_test,cm

def svm_NL(X_train,y_train,X_test):

        from sklearn.svm import SVC
        classifier = SVC(kernel = 'rbf', random_state = 0)
        classifier.fit(X_train, y_train)
        classifier,Accuracy,report,X_test,y_test,cm=cm_prediction(classifier,X_test)
        return  classifier,Accuracy,report,X_test,y_test,cm

def Navie(X_train,y_train,X_test):
        # Fitting K-NN to the Training set
        from sklearn.naive_bayes import GaussianNB
        classifier = GaussianNB()
        classifier.fit(X_train, y_train)
        classifier,Accuracy,report,X_test,y_test,cm=cm_prediction(classifier,X_test)
        return  classifier,Accuracy,report,X_test,y_test,cm
```

```
def knn(X_train,y_train,X_test):

        # Fitting K-NN to the Training set
        from sklearn.neighbors import KNeighborsClassifier
        classifier = KNeighborsClassifier(n_neighbors = 5, metric = 'minkowski', p = 2)
        classifier.fit(X_train, y_train)
        classifier,Accuracy,report,X_test,y_test,cm=cm_prediction(classifier,X_test)
        return  classifier,Accuracy,report,X_test,y_test,cm
def Decision(X_train,y_train,X_test):

        # Fitting K-NN to the Training set
        from sklearn.tree import DecisionTreeClassifier
        classifier = DecisionTreeClassifier(criterion = 'entropy', random_state = 0)
        classifier.fit(X_train, y_train)
        classifier,Accuracy,report,X_test,y_test,cm=cm_prediction(classifier,X_test)
        return  classifier,Accuracy,report,X_test,y_test,cm


def random(X_train,y_train,X_test):

        # Fitting K-NN to the Training set
        from sklearn.ensemble import RandomForestClassifier
        classifier = RandomForestClassifier(n_estimators = 10, criterion = 'entropy', random_state = 0)
        classifier.fit(X_train, y_train)
        classifier,Accuracy,report,X_test,y_test,cm=cm_prediction(classifier,X_test)
        return  classifier,Accuracy,report,X_test,y_test,cm
```

- Each of these functions (logistic, svm_linear, svm_NL, Navie,knn,decision,random) is designed to:
- Train a classifier on (X_train, y_train).
- Predict labels on X_test.

- ❖ Call cm_prediction() to compute below and return them
    - i. Confusion matrix
    - ii. Accuracy
    - iii. Classification report

6. Created function for summarising accuracy results

```
def rfe_classification(acclog,accsvml,accsvmnl,accknn,accnav,accdes,accrf):

    rfedataframe=pd.DataFrame(index=['Logistic','SVC','Random','DecisionTree'],columns=['Logistic','SVMl','SVMnl',
                                                                                        'KNN','Navie','Decision','Random'])

    for number,idex in enumerate(rfedataframe.index):

        rfedataframe['Logistic'][idex]=acclog[number]
        rfedataframe['SVMl'][idex]=accsvml[number]
        rfedataframe['SVMnl'][idex]=accsvmnl[number]
        rfedataframe['KNN'][idex]=accknn[number]
        rfedataframe['Navie'][idex]=accnav[number]
        rfedataframe['Decision'][idex]=accdes[number]
        rfedataframe['Random'][idex]=accrf[number]
    return rfedataframe
```

- ➢ Created dataframe –
- ➢ index=['Logistic','SVC','Random','DecisionTree'],----→ *RFE model type* used for feature selection
- ➢ columns=['Logistic','SVMl','SVMnl', 'KNN','Navie','Decision','Random']---→ *classification model* used to train and test
- ➢ for loop – here iteration occurs for each RFE model(row)
- ➢ the first iteration will be (number=0, idex='Logistic')-→ fill first row
- ➢ the second iteration (number=1, idex='SVC')-→ fills second row and so on.
- ➢ Dataframe will be populated with all the accuracy values at the end.

7. Reading Dataset

```
dataset1=pd.read_csv("prep.csv",index_col=None)
df2=dataset1
df2 = pd.get_dummies(df2, drop_first=True)

indep_X=df2.drop('classification_yes', 1)
dep_Y=df2['classification_yes']
```

- dataset1 = pd.read_csv("prep.csv", index_col=None) – reading the csv file and index_col=None → means don't treat any column as the row index; just use default numeric indexes (0, 1, 2, ...).
- df2 = pd.get_dummies(df2, drop_first=True) – to convert the categorical data to numerical we are using get_dummies
- indep_X – input col in df without classification_yes col
- dep_Y – output is classification_yes col

8. Calling rfeFeature function

```
rfelist=rfeFeature(indep_X,dep_Y,6)

acclog=[]
accsvml=[]
accsvmnl=[]
accknn=[]
accnav=[]
accdes=[]
accrf=[]
```

- Parameters are input ,output and n value (selects the top n features)
- Empty Python lists created to store accuracy values for different classifiers

9. Model Evaluation

```
for i in rfelist:
    X_train, X_test, y_train, y_test=split_scalar(i,dep_Y)


    classifier,Accuracy,report,X_test,y_test,cm=logistic(X_train,y_train,X_test)
    acclog.append(Accuracy)

    classifier,Accuracy,report,X_test,y_test,cm=svm_linear(X_train,y_train,X_test)
    accsvml.append(Accuracy)

    classifier,Accuracy,report,X_test,y_test,cm=svm_NL(X_train,y_train,X_test)
    accsvmnl.append(Accuracy)

    classifier,Accuracy,report,X_test,y_test,cm=knn(X_train,y_train,X_test)
    accknn.append(Accuracy)

    classifier,Accuracy,report,X_test,y_test,cm=Navie(X_train,y_train,X_test)
    accnav.append(Accuracy)

    classifier,Accuracy,report,X_test,y_test,cm=Decision(X_train,y_train,X_test)
    accdes.append(Accuracy)

    classifier,Accuracy,report,X_test,y_test,cm=random(X_train,y_train,X_test)
    accrf.append(Accuracy)

result=rfe_classification(acclog,accsvml,accsvmnl,accknn,accnav,accdes,accrf)
```

- This iterates over each element in rfelist
- Splitting the dataset using split scalar
- For each element in the rfelist , the code trains multiple machine learning classifiers and store the accuracy values
    - Logistic Regression
    - SVM with linear kernel
    - SVM with nonlinear kernel
    - K-Nearest Neighbors (KNN)
    - Naive Bayes

- Decision Tree
- Random Forest

- rfe_classification(acclog,accsvml,accsvmnl,accknn,accnav,accdes,accrf) - will create the dataframe using the accuracy values .

```
result
#3
```

	Logistic	SVMl	SVMnl	KNN	Navie	Decision	Random
Logistic	0.94	0.94	0.94	0.94	0.94	0.94	0.94
SVC	0.87	0.87	0.87	0.87	0.87	0.87	0.87
Random	0.94	0.94	0.94	0.94	0.9	0.91	0.92
DecisionTree	0.98	0.98	0.98	0.98	0.79	0.97	0.97

- Here Decision Tree gave the highest accuracy (**98%**), for 3 selected features.
- Logistic Regression and Random Forest performed good (**92–94% accuracy**).
- SVM performance is low (**87% accuracy**).
- Naive Bayes shows lower accuracy compared to others

```
result
#4
```

	Logistic	SVMl	SVMnl	KNN	Navie	Decision	Random
Logistic	0.95	0.95	0.95	0.95	0.95	0.95	0.95
SVC	0.96	0.96	0.96	0.96	0.96	0.96	0.96
Random	0.97	0.97	0.97	0.98	0.87	0.95	0.97
DecisionTree	0.98	0.98	0.92	0.98	0.81	0.98	0.98

- Decision Tree and Random Forest achieved the highest accuracy (**~98%)**
- SVM and Logistic Regression were consistent (**95–96% accuracy**).

```
result
#5
```

	Logistic	SVMl	SVMnl	KNN	Navie	Decision	Random
Logistic	0.98	0.98	0.98	0.98	0.98	0.98	0.98
SVC	0.99	0.99	0.99	0.99	0.99	0.99	0.99
Random	0.97	0.97	0.98	0.97	0.91	0.96	0.98
DecisionTree	0.95	0.98	0.93	0.94	0.85	0.97	0.98

- SVM achieved the highest accuracy (**99%**)
- Logistic Regression also performed very well (**98% accuracy**).
- Random Forest had slightly variable performance (**96–98% accuracy**).
- Decision Tree was strong but showed lower accuracy in Naive Bayes.

```
result
#6
```

	Logistic	SVMl	SVMnl	KNN	Navie	Decision	Random
Logistic	0.98	0.98	0.98	0.98	0.98	0.99	0.98
SVC	0.99	0.99	0.99	0.99	0.99	0.99	0.99
Random	0.98	0.98	0.99	0.96	0.92	0.95	0.98
DecisionTree	0.96	0.96	0.97	0.95	0.85	0.97	0.96

- SVM Linear achieved the highest accuracy (**99%**)
- Logistic Regression also performed very well (**98–99% accuracy**)
- Random Forest had high accuracy (**96–99%**) but some variations
- Decision Tree had good performance overall (**85–97%**)

Summary:

- ❖ Increasing features from 3 to 5 significantly improves accuracy.
- ❖ SVM Linear performs best (~99%), followed by Logistic Regression (~98–99%) and Random Forest (~96–98%).
- ❖ Naive Bayes is most sensitive to feature selection.
- ❖ Overall , 5 Features provides the best accuracy.