ML Code with Feature Selection – Explanation

SelectKBest

```
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
```

1. Importing required libraries

```
def selectkbest(indep_X,dep_Y,n):
        test = SelectKBest(score_func=chi2, k=n)
        fit1= test.fit(indep_X,dep_Y)
        selectk_features = fit1.transform(indep_X)
        return selectk_features

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)
        sc = StandardScaler()
        X_train = sc.fit_transform(X_train)
        X_test = sc.transform(X_test)
        return X_train, X_test, y_train, y_test

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

     Accuracy=accuracy_score(y_test, y_pred )

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

2. Created a function for Feature selection

- SelectKBest(score_func=chi2, k=n) will pick the **top n features** using the **Chi-Square test**.
- fit1.fit(indep_X,dep_Y) will learn which features are important.
- fit1.transform(indep_X) will keep only the best n features then return it.

3. Created a function for Split Scalar

- train_test_split(...) will Split the data into training (75%) and testing (25%).
- StandardScaler() - scales features so they are on the same range.
- fit_transform on training data - learn scaling values (mean, std) and apply.
- transform on test data - apply the same scaling.

4. Created a function for Prediction using confusion matrix

- Predict(X_test) - The model predict using test data.

- Confusion_matrix (y_test, y_pred) will create the confusion matrix
- Accuracy – Correct predictions saved in accuracy
- Classification_report – detailed report created for the predictions.

```
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
```

5. Created a function for logistic regression Algorithm

- LogisticRegression(random_state = 0) – creates an instance of the logistic regression classifier
- classifier.fit(X_train, y_train)- Trains the logistic regression model with training data
- cm_prediction(classifier,X_test) – calls prediction function and make prediction using test data and returns accuracy, report and confusion matrix

6. Created a function for SVM Linear Algorithm

- SVC(…) – creates SVMLinear classifier with a linear kernel, meaning it tries to find a straight line
- classifier.fit(X_train, y_train)- Trains the model with training data
- cm_prediction(classifier,X_test) – calls prediction function and make prediction using test data and returns accuracy, report and confusion matrix

7. Created a function for SVM Non Linear Algorithm

- SVC(…) – creates SVM classifier using the RBF kernel. The RBF kernel maps data into a higher-dimensional space to handle cases where data is not linear
- classifier.fit(X_train, y_train)- Trains the model with training data
- cm_prediction(classifier,X_test) – calls prediction function and make prediction using test data and returns accuracy, report and confusion matrix

8. Created a function for Navies' bayes Algorithm

- GaussianNB()– Creates an instance of the Gaussian Naive Bayes classifier.
- classifier.fit(X_train, y_train)- Trains the model with training data
- cm_prediction(classifier,X_test) – calls prediction function and make prediction using test data and returns accuracy, report and confusion matrix

```python
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
```

9. Created a function for K-Nearest Neighbors Algorithm

- KNeighborsClassifier(n_neighbors=5, metric='minkowski', p=2) –
    a. n_neighbors=5 means the algorithm will look at the 5 closest neighbors to classify a new point.
    b. metric='minkowski' with p=2 means it uses Euclidean distance to measure closeness between points.
- classifier.fit(X_train, y_train)- Trains the model with training data
- cm_prediction(classifier,X_test) – calls prediction function and make prediction using test data and returns accuracy, report and confusion matrix

10. Created a function for Decision Tree Algorithm

- DecisionTreeClassifier(criterion = 'entropy', random_state = 0) - criterion='entropy' tells the tree to use information gain / entropy to decide where to split nodes
- classifier.fit(X_train, y_train)- Trains the model with training data
- cm_prediction(classifier,X_test) – calls prediction function and make prediction using test data and returns accuracy, report and confusion matrix

11. Created a function for Random Forest Algorithm

- RandomForestClassifier(n_estimators = 10, criterion = 'entropy', random_state = 0)
    a. n_estimators = 10,-- random forest will create **10 decision trees**, More trees – better accuracy
    b. criterion = 'entropy' -- Each tree will split its nodes using the **entropy (information gain)** method. Measures how pure the data is at each split
- classifier.fit(X_train, y_train)- Trains the model with training data
- cm_prediction(classifier,X_test) – calls prediction function and make prediction using test data and returns accuracy, report and confusion matrix

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

    dataframe=pd.DataFrame(index=['ChiSquare'],columns=['Logistic','SVMl','SVMnl','KNN','Navie','Decision','Random'])
    for number,idex in enumerate(dataframe.index):
        dataframe['Logistic'][idex]=acclog[number]
        dataframe['SVMl'][idex]=accsvml[number]
        dataframe['SVMnl'][idex]=accsvmnl[number]
        dataframe['KNN'][idex]=accknn[number]
        dataframe['Navie'][idex]=accnav[number]
        dataframe['Decision'][idex]=accdes[number]
        dataframe['Random'][idex]=accrf[number]
    return dataframe
```

12. Created a function for SelectK
    - It accepts inputs that hold accuracy values for different classifiers
    - pd.DataFrame(index=['ChiSquare'], columns=['Logistic…] - Creates an empty dataframe with one row index - 'ChiSquare' & several col
    - enumerate(dataframe.index): - is ['ChiSquare'] --- so this loop runs once
    - enumerate(...) gives two values each iteration:
        - number → the integer position (0 for the first row)
        - index → the actual index label (the string 'ChiSquare')
    - dataframe['Logistic'][index] = acclog[number] - Sets the cell at row 'ChiSquare' and column 'Logistic' to the value acclog[0].
    - Acclog[0] – accuracy value for logistic is set to chiSquare row and logistic col of the dataframe.
    - Like above all the accuracy values will be placed to specific columns.

```
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']
```

13. Reading Dataset
    - 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

```
kbest=selectkbest(indep_X,dep_Y,6)

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

14. Calling a function selectkbest
    - Parameters are input ,output and n value (6 most important features)
    - Empty Python lists created to store accuracy values for different classifiers

```
X_train, X_test, y_train, y_test=split_scalar(kbest,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=selectk_Classification(acclog,accsvml,accsvmnl,accknn,accnav,accdes,accrf)
```

15. Training and Evaluation
    - X_train, X_test, y_train, y_test = split_scalar(kbest, dep_Y) – split the dataset to training and test – kbest(input var selected features)
    - classifier, Accuracy, report, X_test, y_test, cm = logistic(X_train, y_train, X_test)
    - acclog.append(Accuracy)
        - Trains a **Logistic Regression** model.
        - Returns:
        - classifier → trained model
        - Accuracy → test accuracy
        - report → classification report
        - cm → confusion matrix

- Appends the accuracy to the list acclog.
- Like above all the model will be trained and tested and the accuracy value is appended to the list.
- result = selectk_Classification(acclog, accsvml, accsvmnl, accknn, accnav, accdes, accrf) – will create the dataframe using the accuracy values .

```
result
#5
```

	Logistic	SVMl	SVMnl	KNN	Navie	Decision	Random
ChiSquare	0.94	0.94	0.95	0.89	0.83	0.96	0.95

Printing the result for 5 features. This way we can check by different n features and select the best n number of features which will give more accuracy.

```
result
#4
```

	Logistic	SVMl	SVMnl	KNN	Navie	Decision	Random
ChiSquare	0.85	0.82	0.83	0.86	0.79	0.89	0.89

```
result
#6
```

	Logistic	SVMl	SVMnl	KNN	Navie	Decision	Random
ChiSquare	0.96	0.96	0.96	0.93	0.89	0.97	0.97

Summary:

- Model performance improves as we increase the number of selected features from 4 to 6.
- Decision Tree and Random Forest has the highest accuracy (0.97) with 6 features, making this the best configuration for classification.
- Naive Bayes performs the weakest across all feature sets.