ML Code with Feature Selection – Explanation

Feature Importance (Tree based models)

Tree-based feature importance helps us find the most important features by checking how much each feature improves the model's prediction accuracy. (i.e.) It shows Which features have the strongest influence and Which features can be removed.

1. Importing required libraries

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier, GradientBoostingClassifier, ExtraTreesClassifier
from sklearn.tree import DecisionTreeClassifier
```

2. Creating the function for Feature Importance

```
def featureImportanceFeature(indep_X, dep_Y, n):
        filist = []

        # Define tree-based models
        RF = RandomForestClassifier(n_estimators=100, random_state=42)
        DT = DecisionTreeClassifier(random_state=42)
        GB = GradientBoostingClassifier(n_estimators=100, random_state=42)
        ET = ExtraTreesClassifier(n_estimators=100, random_state=42)

        fimodellist = [RF, DT, GB, ET]

        for model in fimodellist:
            print(model)
            model.fit(indep_X, dep_Y)

            # Get feature importances
            importances = model.feature_importances_

            # Sort and select top n features
            importance_df = pd.DataFrame({
                'Feature': indep_X.columns,
                'Importance': importances
            }).sort_values(by='Importance', ascending=False)

            top_features = importance_df['Feature'].head(n).tolist()
            print(f"Top {n} features for {type(model).__name__}: {top_features}")

            # Transform dataset with selected features
            fi_features = indep_X[top_features].values
            filist.append(fi_features)

        return filist
```

- Created empty list – filist[]
- Defined the tree based models
- fimodellist = [RF, DT, GB, ET] - Stores all the models in a list
- In For loop – loops through each model in the list print its name.
- model.fit(indep_X, dep_Y) - Train the model on full dataset
- importances = model.feature_importances_ - built-in property which will give numerical score for every feature
- Created a DataFrame showing each feature and its importance score.
- Sorts the features in descending order, so the most important features come first.
- top_features = importance_df['Feature'].head(n).tolist() - Selecting the top n features based on their importance scores and storing it to a list
- fi_features = indep_X[top_features].values – we are storing the dataset containing only its most important features.-- .values→ get the values of those features .
- Appending it to the list

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
```

```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
```

- 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

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

    fidataframe = pd.DataFrame(index=['RandomForest','DecisionTree','GradientBoosting','ExtraTrees'],
                               columns=['Logistic','SVMl','SVMnl','KNN','Navie','Decision','Random'])

    for number, idex in enumerate(fidataframe.index):
        fidataframe['Logistic'][idex] = acclog[number]
        fidataframe['SVMl'][idex] = accsvml[number]
        fidataframe['SVMnl'][idex] = accsvmnl[number]
        fidataframe['KNN'][idex] = accknn[number]
        fidataframe['Navie'][idex] = accnav[number]
        fidataframe['Decision'][idex] = accdes[number]
        fidataframe['Random'][idex] = accrf[number]

    return fidataframe
```

- This function collects accuracy scores from different classifiers when using feature importance-based feature selection.
- Creates an empty **Pandas DataFrame** with:
- Rows =RandomForest, DecisionTree, GradientBoosting, ExtraTrees
- Columns = Logistic, SVMl, SVMnl, KNN, Navie, Decision, Random

- This DataFrame will store accuracy values for each combination.
- For loop – here iteration occurs for each tree based model(row)
- 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 featureImportanceFeature function

```
filist = featureImportanceFeature(indep_X, dep_Y, 3)

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

```
RandomForestClassifier(random_state=42)
Top 3 features for RandomForestClassifier: ['hrmo', 'pcv', 'sc']
DecisionTreeClassifier(random_state=42)
Top 3 features for DecisionTreeClassifier: ['hrmo', 'sg_d', 'sg_c']
GradientBoostingClassifier(random_state=42)
Top 3 features for GradientBoostingClassifier: ['hrmo', 'sg_d', 'al']
ExtraTreesClassifier(random_state=42)
Top 3 features for ExtraTreesClassifier: ['htn_yes', 'hrmo', 'dm_yes']
```

- Parameters are input ,output and n value (selects the top n features to be selected)
- Empty Python lists created to store accuracy values for different classifiers
- Output of the code shown above - three features which are selected displayed

9. Model Evaluation

```
for i in filist:
    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)

fi_result = fi_classification(acclog, accsvml, accsvmnl, accknn, accnav, accdes, accrf)
```

- For each dataset in filist (which contains top features chosen by different tree-based models), split it into training and testing sets.
- Train multiple classifiers and measure accuracy
- Save each classifier’s accuracy in separate lists
- Use fi_classification() to combine all accuracies into a DataFrame

```
fi_result
#3
```

	Logistic	SVMl	SVMnl	KNN	Navie	Decision	Random
RandomForest	0.94	0.94	0.94	0.94	0.9	0.91	0.93
DecisionTree	0.99	0.96	0.96	0.99	0.78	0.99	0.99
GradientBoosting	0.98	0.94	0.96	0.98	0.87	0.99	0.96
ExtraTrees	0.96	0.94	0.97	0.95	0.8	0.96	0.94

- DecisionTree feature selection gave the highest accuracy, with classifiers (Logistic, KNN, Decision Tree, Random Forest) - 0.99.
- RandomForest and GradientBoosting also gave good performance, while ExtraTrees showed slightly lower performance for some classifiers.
- Naive Bayes had the lowest accuracy across most feature selection methods

```
fi_result
#4
```

	Logistic	SVMl	SVMnl	KNN	Navie	Decision	Random
RandomForest	0.93	0.93	0.94	0.93	0.91	0.91	0.92
DecisionTree	0.98	0.98	0.97	0.98	0.78	0.96	0.98
GradientBoosting	0.98	0.98	0.99	0.99	0.91	0.95	1.0
ExtraTrees	0.97	0.97	0.92	0.98	0.87	0.97	0.95

- GradientBoosting achieved the highest performance, with Random Forest classifier reaching perfect accuracy (1.0).
- DecisionTree and ExtraTrees also gave strong results, with most classifiers achieving above 0.95 accuracy.
- Naive Bayes performed worse compared to other classifiers

```
fi_result
#5
```

	Logistic	SVMl	SVMnl	KNN	Navie	Decision	Random
RandomForest	0.97	0.97	0.97	0.96	0.87	0.93	0.97
DecisionTree	0.98	0.99	0.98	0.99	0.94	0.96	0.96
GradientBoosting	0.97	0.97	0.98	1.0	0.91	0.96	0.99
ExtraTrees	0.96	0.96	0.96	0.96	0.95	0.98	0.97

- GradientBoosting and DecisionTree deliver very high performance, with KNN reaching perfect accuracy (1.0) for GradientBoosting features.
- RandomForest and ExtraTrees also perform strongly, with most accuracies above 0.95.
- Naive Bayes still performs lower compared to other models

```
fi_result
#6
```

	Logistic	SVMl	SVMnl	KNN	Navie	Decision	Random
RandomForest	0.98	0.98	0.99	0.97	0.93	0.96	0.97
DecisionTree	0.99	1.0	0.98	0.98	0.94	0.96	0.97
GradientBoosting	0.98	0.98	0.99	0.98	0.94	0.95	0.99
ExtraTrees	0.97	0.98	0.97	0.98	0.95	0.99	0.98

- Almost all classifiers perform well, with most accuracies above 0.95.
- The DecisionTree feature selection combined with SVM-linear achieves a perfect accuracy of 1.0.

Summary:

- ❖ Increasing features from 3 to 6 significantly improves accuracy showing that additional important features enhanced learning performance.
- ❖ DecisionTree and GradientBoosting consistently performed well and gave highest accuracies across classifiers.
- ❖ Naive Bayes showed lower and more variable accuracy
- ❖ Overall , 6 Features provides the best accuracy.