SelectK_Classification – Documentation

What your code is trying to do:

1. Load data →
2. One-hot encode categorical columns →
3. Split features (X) & target (y) →
4. Select top-k features (chi-square) →
5. Train/test split + scale →
6. Train many classifiers (LogReg, SVMs, KNN, NB, DT, RF) →
7. Predict on test set →
8. Collect Accuracy & Report →
9. Put results in a small summary table.

Line-by-line (grouped) explanation

Imports

python Copy Edit

```python
import pandas as pd, numpy as np
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.feature_selection import SelectKBest, chi2, RFE
from sklearn.linear_model import LogisticRegression
import matplotlib.pyplot as plt
import pickle, time
```

- pandas/numpy: For data handling
- scikit-learn: For splitting, scaling, feature selection, models, metrics
- matplotlib/pickle/time aren't actually used here.

1. selectkbest

```python
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
```

- Creates a **chi-square** selector and picks the **top `n` features** with the highest association to the target.

- **Important constraint:** `chi2` requires **non-negative features** (e.g., 0/1 dummies, counts). If you feed negatives here, it will error.

2. split_scalar

```
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
```

- Splits data (75/25).

- **Standardizes** features (zero mean, unit variance) using parameters learned from **training only**, then applies to test. Good practice for SVM/KNN/LogReg.

3.cm_prediction

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

    from sklearn.metrics import confusion_matrix, accuracy_score, classification_report
    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
```

Uses the trained `classifier` to predict the test set, then computes:

- Confusion matrix
- Accuracy score
- Classification report (precision/recall/F1 by class)

4.The model trainers

Each function trains a different classifier, then calls `cm_prediction` to evaluate:

```
def logistic(X_train,y_train,X_test):
    classifier = LogisticRegression(random_state=0)
    classifier.fit(X_train, y_train)
    return cm_prediction(classifier,X_test)

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)
    return cm_prediction(classifier,X_test)

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)
    return cm_prediction(classifier,X_test)
```

```
def Navie(X_train,y_train,X_test):
    from sklearn.naive_bayes import GaussianNB
    classifier = GaussianNB()
    classifier.fit(X_train, y_train)
    return cm_prediction(classifier,X_test)

def knn(X_train,y_train,X_test):
    from sklearn.neighbors import KNeighborsClassifier
    classifier = KNeighborsClassifier(n_neighbors=5, metric='minkowski', p=2)
    classifier.fit(X_train, y_train)
    return cm_prediction(classifier,X_test)

def Decision(X_train,y_train,X_test):
    from sklearn.tree import DecisionTreeClassifier
    classifier = DecisionTreeClassifier(criterion='entropy', random_state=0)
    classifier.fit(X_train, y_train)
    return cm_prediction(classifier,X_test)

def random(X_train,y_train,X_test):
    from sklearn.ensemble import RandomForestClassifier
    classifier = RandomForestClassifier(n_estimators=10, criterion='entropy', random_state=0)
    classifier.fit(X_train, y_train)
    return cm_prediction(classifier,X_test)
```

5.Build the results table

- You only create **one row** (“ChiSquare”), then fill each column with the first element of each accuracy list.

```
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
```

Visual flow (at a glance):

CSV → DataFrame

→ get_dummies

→X/y split

→[LEAKAGE] SelectKBest(chi2) on FULL X,y

→ kbest_X

→ train_test_split(kbest_X, y)

→ StandardScaler (fit on train, transform test)

→ Train many models

→ Predict on X_test → Accuracy/Report/CM

→ Summary DataFrame