In this project, we focus on feature engineering to improve machine learning model performance. Using a synthetic dataset generated with sklearn's make_classification, we create polynomial features and select the most impactful ones using techniques like SelectKBest and Recursive Feature Elimination (RFE). The final step involves comparing the performance of classification models built with all features and the selected features to evaluate the impact of feature engineering.

WEEK 3

Unsupervised Learning & Feature Engineering

ML-WEEK 3.2

Noor Ul Ain

Week 3: Unsupervised Learning and Feature Engineering

OBJECTIVE

Develop skills in creating and selecting features to improve machine learning model performance.

DATASET

Synthetic dataset using sklearn's make_classification. This allows you to generate a large random dataset tailored to the task, with control over the number of informative, redundant, and noisy features.

DATASET GENERATION COMMAND:

from sklearn.datasets import make_classification

X, y = make_classification(n_samples=1000, n_features=20, n_informative=2, n_redundant=10, n_clusters_per_class=1, weights=[0.99], flip_y=0, random_state=1)

ACTIVITIES

1. **Feature Creation**:
    - Generate the dataset and add manually created features that could be interaction terms or polynomial features.
2. **Feature Selection**:
    - Implement various feature selection techniques to determine the most impactful features.
    - Evaluate the effect of feature selection on model performance.
3. **Model Building**:
    - Rebuild classification models using selected features to compare performance against the baseline model with all features.

CODE

```
import pandas as pd
import numpy as np
from sklearn.datasets import make_classification
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import PolynomialFeatures, MinMaxScaler
from sklearn.feature_selection import SelectKBest, chi2, RFE
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score

# Step 1: Generate the synthetic dataset
X, y = make_classification(n_samples=1000, n_features=20, n_informative=2, n_redundant=10,
                           n_clusters_per_class=1, weights=[0.99], flip_y=0, random_state=1)

# Convert to DataFrame for easier manipulation
df = pd.DataFrame(X, columns=[f'feature_{i}' for i in range(X.shape[1])])
df['target'] = y

# Step 2: Feature Creation
# Create polynomial features (interaction terms)
poly = PolynomialFeatures(degree=2, interaction_only=True, include_bias=False)
poly_features = poly.fit_transform(df.drop('target', axis=1))
poly_feature_names = poly.get_feature_names_out(df.columns[:-1])

# Convert to DataFrame
df_poly = pd.DataFrame(poly_features, columns=poly_feature_names)
df_poly['target'] = y
```

```
# Step 3: Feature Selection


# 3.1: Split data into training and testing sets

X_train, X_test, y_train, y_test = train_test_split(df_poly.drop('target', axis=1), df_poly['target'], test_size=0.3, random_state=1)


# 3.2: Scale the data to be non-negative

scaler = MinMaxScaler()

X_train_scaled = scaler.fit_transform(X_train)

X_test_scaled = scaler.transform(X_test)


# 3.3: SelectKBest using chi2

selector_kbest = SelectKBest(score_func=chi2, k=20)

X_train_kbest = selector_kbest.fit_transform(X_train_scaled, y_train)

X_test_kbest = selector_kbest.transform(X_test_scaled)


# 3.4: Recursive Feature Elimination (RFE) with RandomForestClassifier

model_rfe = RandomForestClassifier(random_state=1)

rfe = RFE(model_rfe, n_features_to_select=20)

X_train_rfe = rfe.fit_transform(X_train, y_train)

X_test_rfe = rfe.transform(X_test)


# Step 4: Model Building


# 4.1: Baseline model with all features

model_baseline = RandomForestClassifier(random_state=1)

model_baseline.fit(X_train, y_train)

y_pred_baseline = model_baseline.predict(X_test)

baseline_accuracy = accuracy_score(y_test, y_pred_baseline)
```

```
print(f'Baseline Model Accuracy with All Features: {baseline_accuracy:.4f}')


# 4.2: Model with selected features (SelectKBest)
model_kbest = RandomForestClassifier(random_state=1)
model_kbest.fit(X_train_kbest, y_train)
y_pred_kbest = model_kbest.predict(X_test_kbest)
kbest_accuracy = accuracy_score(y_test, y_pred_kbest)
print(f'Model Accuracy with SelectKBest Features: {kbest_accuracy:.4f}')


# 4.3: Model with selected features (RFE)
model_rfe = RandomForestClassifier(random_state=1)
model_rfe.fit(X_train_rfe, y_train)
y_pred_rfe = model_rfe.predict(X_test_rfe)
rfe_accuracy = accuracy_score(y_test, y_pred_rfe)
print(f'Model Accuracy with RFE Features: {rfe_accuracy:.4f}')
```

LIBRARIES USED AND THEIR APPLICATIONS

- **Pandas**: Used for data manipulation and analysis, specifically for creating and handling DataFrames.
- **NumPy**: Provides support for large multi-dimensional arrays and matrices, along with a collection of mathematical functions.
- **Scikit-learn**: A machine learning library for Python that provides simple and efficient tools for data mining and data analysis. It includes:
    - make_classification: To generate a synthetic classification dataset.
    - train_test_split: To split the dataset into training and testing sets.
    - PolynomialFeatures: To generate polynomial and interaction features.
    - MinMaxScaler: To scale features to a given range.
    - SelectKBest and chi2: To select the top k features based on the chi-squared test.

- RFE (Recursive Feature Elimination): To select features by recursively considering smaller sets of features.
- RandomForestClassifier: An ensemble learning method for classification that operates by constructing multiple decision trees.

CODE'S EXPLANATION

Generate the Synthetic Dataset:

- X, y = make_classification(...) generates a dataset with 1000 samples, 20 features (2 informative, 10 redundant), and a binary target with imbalanced classes.

Convert to DataFrame:

- df = pd.DataFrame(X, columns=[f'feature_{i}' for i in range(X.shape[1])]) creates a DataFrame with feature names.
- df['target'] = y adds the target variable to the DataFrame.

Feature Creation:

- poly = PolynomialFeatures(degree=2, interaction_only=True, include_bias=False) initializes polynomial feature generation.
- poly_features = poly.fit_transform(df.drop('target', axis=1)) creates polynomial features for interaction terms only.
- df_poly = pd.DataFrame(poly_features, columns=poly_feature_names) converts polynomial features to a DataFrame.

Feature Selection:

- **Split Data**:
  - X_train, X_test, y_train, y_test = train_test_split(...) splits the dataset into training and testing sets.
- **Scale Data**:
  - scaler = MinMaxScaler() initializes the scaler.
  - X_train_scaled = scaler.fit_transform(X_train) scales training data.

- X_test_scaled = scaler.transform(X_test) scales testing data.

- **SelectKBest**:
  - selector_kbest = SelectKBest(score_func=chi2, k=20) selects the top 20 features based on the chi-squared test.
  - X_train_kbest = selector_kbest.fit_transform(X_train_scaled, y_train) transforms training data to include only the best features.
  - X_test_kbest = selector_kbest.transform(X_test_scaled) applies the same transformation to testing data.
- **Recursive Feature Elimination (RFE)**:
  - model_rfe = RandomForestClassifier(random_state=1) initializes the RandomForestClassifier.
  - rfe = RFE(model_rfe, n_features_to_select=20) sets up RFE to select 20 features.
  - X_train_rfe = rfe.fit_transform(X_train, y_train) transforms training data to include only selected features.
  - X_test_rfe = rfe.transform(X_test) applies the same transformation to testing data.

 Model Building:

- **Baseline Model**:
  - model_baseline = RandomForestClassifier(random_state=1) initializes the RandomForestClassifier.
  - model_baseline.fit(X_train, y_train) trains the model with all features.
  - y_pred_baseline = model_baseline.predict(X_test) makes predictions on the test set.
  - accuracy_score(y_test, y_pred_baseline) calculates and prints the accuracy.
- **Model with SelectKBest Features**:
  - model_kbest = RandomForestClassifier(random_state=1) initializes a new RandomForestClassifier.
  - model_kbest.fit(X_train_kbest, y_train) trains the model with features selected by SelectKBest.
  - y_pred_kbest = model_kbest.predict(X_test_kbest) makes predictions on the test set.
  - accuracy_score(y_test, y_pred_kbest) calculates and prints the accuracy.

- **Model with RFE Features**:
    - model_rfe = RandomForestClassifier(random_state=1) initializes another RandomForestClassifier.
    - model_rfe.fit(X_train_rfe, y_train) trains the model with features selected by RFE.
    - y_pred_rfe = model_rfe.predict(X_test_rfe) makes predictions on the test set.
    - accuracy_score(y_test, y_pred_rfe) calculates and prints the accuracy.