Feature Engineering and Unsupervised Learning Project

Overview

This project is part of the "Introduction to Machine Learning" course, focusing on feature engineering and unsupervised learning. The primary objective is to develop skills in creating and selecting features to enhance machine learning model performance using a synthetic dataset generated with sklearn.

Dataset

Dataset Generation

The synthetic dataset was generated using the following command from sklearn:

python

Copy code

```
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
)
```

- **Samples**: 1000
- **Features**: 20 (2 informative, 10 redundant, 8 noisy)
- **Class Imbalance**: 99% of one class
- **No Label Noise**

Project Structure

- notebooks/
  - Feature_Engineering_Project.ipynb: Main notebook with the complete code and analysis.
- data/
  - synthetic_data.csv: Generated synthetic dataset (optional, if you save it).
- README.md: Project overview and instructions.
- LICENSE: License information.

Feature Engineering

Feature Creation

Created polynomial and interaction features to enrich the dataset:

python

Copy code

```
from sklearn.preprocessing import PolynomialFeatures


poly = PolynomialFeatures(degree=2, interaction_only=True, include_bias=False)

poly_features = poly.fit_transform(X)
```

Feature Selection

1. **Tree-based Feature Importance**: Used a Random Forest classifier to determine feature importances.
2. **Recursive Feature Elimination (RFE)**: Selected the top features iteratively.
3. **SelectKBest (Chi-Square)**: Selected top features based on Chi-Square statistics after normalizing features to non-negative values.

Normalization for Chi-Square

Normalized features to a range [0, 1] for Chi-Square feature selection:

python

Copy code

from sklearn.preprocessing import MinMaxScaler

scaler = MinMaxScaler()

X_scaled = scaler.fit_transform(X)

Model Building and Evaluation

Built and evaluated classification models using Random Forest with different sets of features:

1. **Baseline Model**: Using all features.
2. **RFE-selected Features Model**.
3. **SelectKBest-selected Features Model**.

Model Evaluation

Used classification metrics to compare model performance:

python

Copy code

from sklearn.metrics import classification_report

print(classification_report(y_test, y_pred))

Results

The results demonstrate the significant impact of feature engineering and selection on model performance. Detailed steps, code, and performance metrics for each model are included in the Jupyter notebook.

Usage

1. **Clone the repository**:

bash

Copy code

git clone https://github.com/yourusername/feature-engineering-unsupervised-learning.git

cd feature-engineering-unsupervised-learning

2. **Install the required packages**:

bash

Copy code

```
pip install numpy pandas scikit-learn matplotlib
```

3. **Run the Jupyter notebook**:

bash

Copy code

```
jupyter notebook
```

Requirements

- Python 3.x
- NumPy
- Pandas
- Scikit-learn
- Matplotlib

```
# Import necessary libraries
import numpy as np
import pandas as pd
from sklearn.datasets import make_classification
from sklearn.preprocessing import PolynomialFeatures, MinMaxScaler
from sklearn.ensemble import RandomForestClassifier
from sklearn.feature_selection import RFE, SelectKBest, chi2
from sklearn.model_selection import train_test_split
from sklearn.metrics import classification_report
import matplotlib.pyplot as plt

# Step 1: Generate the 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
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
poly = PolynomialFeatures(degree=2, interaction_only=True, include_bias=False)
poly_features = poly.fit_transform(df.drop('target', axis=1))

# Convert polynomial features to DataFrame
poly_feature_names = poly.get_feature_names_out(df.columns[:-1])
df_poly = pd.DataFrame(poly_features, columns=poly_feature_names)

# Combine original and polynomial features
df_combined = pd.concat([df.drop('target', axis=1), df_poly], axis=1)
```

```python
df_combined['target'] = df['target']

# Normalize features to non-negative values for chi2
scaler = MinMaxScaler()
df_combined_scaled = pd.DataFrame(scaler.fit_transform(df_combined.drop('target', axis=1)), columns=df_combined.columns[:-1])
df_combined_scaled['target'] = df_combined['target']

# Step 3: Feature Selection

# a. Feature Importance using Tree-based Models
model = RandomForestClassifier(random_state=1)
model.fit(df_combined.drop('target', axis=1), df_combined['target'])

# Get feature importances
importances = model.feature_importances_
indices = np.argsort(importances)[::-1]

# Plot feature importances
plt.figure(figsize=(10, 6))
plt.title("Feature importances")
plt.bar(range(df_combined.shape[1]-1), importances[indices])
plt.xticks(range(df_combined.shape[1]-1), df_combined.columns[indices], rotation=90)
plt.show()

# b. Recursive Feature Elimination (RFE)
```

```python
rfe = RFE(estimator=model, n_features_to_select=10, step=1)
rfe.fit(df_combined.drop('target', axis=1), df_combined['target'])

# Get selected features
selected_features_rfe = df_combined.columns[:-1][rfe.support_]

# c. SelectKBest using Chi-Square
selector = SelectKBest(chi2, k=10)
selector.fit(df_combined_scaled.drop('target', axis=1), df_combined_scaled['target'])

# Get selected features
selected_features_kbest = df_combined_scaled.columns[:-1][selector.get_support()]

# Step 4: Model Building
# Split the data
X_train, X_test, y_train, y_test = train_test_split(df_combined.drop('target', axis=1), df_combined['target'], test_size=0.2, random_state=1)

# Baseline model using all features
model_baseline = RandomForestClassifier(random_state=1)
model_baseline.fit(X_train, y_train)
y_pred_baseline = model_baseline.predict(X_test)
print("Baseline Model Performance:\n", classification_report(y_test, y_pred_baseline))

# Model using RFE selected features
X_train_rfe = X_train[selected_features_rfe]
```

```python
X_test_rfe = X_test[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)
print("RFE Model Performance:\n", classification_report(y_test, y_pred_rfe))

# Model using SelectKBest selected features
X_train_kbest = X_train[selected_features_kbest]
X_test_kbest = X_test[selected_features_kbest]
model_kbest = RandomForestClassifier(random_state=1)
model_kbest.fit(X_train_kbest, y_train)
y_pred_kbest = model_kbest.predict(X_test_kbest)
print("SelectKBest Model Performance:\n", classification_report(y_test, y_pred_kbest))
```