preprocessing

May 30, 2025

1 Data Preprocessing

This notebook focuses on preprocessing and feature selection for the given dataset. It tests different feature selection techniques and builds multiple datasets using different combinations of feature selection and preprocessing techniques. Due to the limited size of the dataset, this project relies on preprocessors to handle the skewness of the data.

[2]:
```
import os

import joblib
import pandas as pd
from sklearn.feature_selection import SequentialFeatureSelector,␣
 ↪mutual_info_classif
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import StratifiedKFold
from sklearn.preprocessing import (
    StandardScaler,
    MinMaxScaler,
    QuantileTransformer,
    PowerTransformer,
)
```

[4]:
```
data_path = os.path.join(os.path.dirname(os.getcwd()), "data", "raw_data.csv")
df = pd.read_csv(data_path)
df["ID"] = df["ID"].astype("category")
df["Diagnosis"] = df["Diagnosis"].astype("category")
df = df.drop(
    "compactness2", axis=1
)  # Exclude compactness2 from further analysis as previously determined
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 501 entries, 0 to 500
Data columns (total 10 columns):
 #   Column               Non-Null Count  Dtype
---  ------               --------------  -----
 0   ID                   501 non-null    category
 1   Diagnosis            501 non-null    category
```

```
 2   radius2             501 non-null    float64
 3   texture2            501 non-null    float64
 4   perimeter2          501 non-null    float64
 5   area2               501 non-null    float64
 6   concavity2          501 non-null    float64
 7   concave_points2     501 non-null    float64
 8   symmetry2           501 non-null    float64
 9   fractal_dimension2  501 non-null    float64
dtypes: category(2), float64(8)
memory usage: 53.1 KB
```

1.1 Feature Selection

[6]:
```python
X = df[
    [
        "radius2",
        "texture2",
        "perimeter2",
        "area2",
        "concavity2",
        "concave_points2",
        "symmetry2",
        "fractal_dimension2",
    ]
]
X
```

[6]:
```
     radius2  texture2  perimeter2   area2  concavity2  concave_points2  \
0     0.8245    2.6640       4.073  49.850    0.396000         0.052790
1     0.3419    1.6780       2.331  29.630    0.005812         0.007039
2     0.3777    1.4620       2.492  19.140    0.000000         0.000000
3     0.2366    1.4280       1.822  16.970    0.025950         0.010370
4     0.4062    1.2100       2.635  28.470    0.011680         0.007445
..       …        …          …       …         …                …
496   0.1194    1.4340       1.778   9.549    0.043050         0.016670
497   0.2143    0.7712       1.689  16.640    0.015100         0.007584
498   0.3677    1.4710       1.597  22.680    0.040040         0.015440
499   0.2954    0.8836       2.109  23.240    0.005383         0.005623
500   0.3834    1.0030       2.495  28.620    0.019770         0.009199

     symmetry2  fractal_dimension2
0      0.03546            0.029840
1      0.02014            0.002326
2      0.02882            0.006872
3      0.01357            0.003040
4      0.02406            0.001769
..         …                  …
```

```
496    0.02470           0.007358
497    0.02104           0.001887
498    0.02719           0.007596
499    0.01940           0.001180
500    0.01805           0.003629

[501 rows x 8 columns]
```

[7]:
```
y = df["Diagnosis"]
y.shape
```

[7]:
```
(501,)
```

1.1.1 Mutual Information

Mutual information (MI) is a non-parametric measure of the dependency between two variables. In feature selection, MI quantifies how much knowing the value of a feature reduces uncertainty about the target variable. Features with higher MI scores are considered more informative for predicting the target. MI can capture both linear and non-linear relationships. However, MI does not account for feature redundancy; multiple features with high MI may be correlated with each other. MI values are not standardized, so scores should be interpreted relative to each other within the dataset.

[8]:
```
mi_scores = mutual_info_classif(X, y, discrete_features="auto", random_state=42)

mi_series = pd.Series(mi_scores, index=X.columns).sort_values(ascending=False)
mi_series
```

[8]:
```
area2                 0.333547
perimeter2            0.280269
radius2               0.241028
concave_points2       0.160719
concavity2            0.103252
fractal_dimension2    0.061076
symmetry2             0.034221
texture2              0.007854
dtype: float64
```

- **High Scores:** `area2`, `perimeter2`, and `radius2` have the highest MI scores. This suggests these features - previously labeled as size-related features - contain the most information about the diagnosis and are likely the most important predictors. These features relate to the size and shape irregularities of the cell nuclei.
- **Moderate Scores:** `concave_points2`, and `concavity2` show moderate MI scores, indicating some relevance to the diagnosis. These features were previously labeled as concavity-related features
- **Lower Scores:** `fractal_dimension2`, and `symmetry2` have lower scores, suggesting they provide less information for predicting the diagnosis based on this measure.
- **Lowest Score:** `texture2` has a very low score, indicating little predictive power.

In summary, features related to the size (`area2`, `perimeter2`, `radius2`) and concave points (`concave_points2`, `concavity2`) of the cell nuclei appear most strongly associated with the diagnosis according to the MI analysis. Features previously labeled as looks-related have the least predictive properties.

1.1.2 Model-based Feature Selection

Sequential Feature Selection (SFS) is an iterative method used to select a subset of features that optimizes a specific model's performance metric. In this case, **Backward SFS** is used. It starts with all available features and progressively removes the least important feature one at a time. The importance of a feature subset is evaluated using a `LogisticRegression` model, assessed by its **F1-weighted score** through **5-fold stratified cross-validation**. The process continues until removing a feature does not improve the score beyond a specified tolerance (`tol=1e-5`), automatically determining the optimal number of features. This approach aims to find a compact feature set that maintains or improves predictive performance while potentially reducing model complexity and overfitting. `LogisticRegression` is chosen as the underlying model due to its simplicity, interpretability, and efficiency for binary classification tasks like this one. The **F1-weighted score** is selected as the evaluation metric because it provides a balanced measure between precision and recall, which is particularly important for imbalanced datasets. The 'weighted' aspect ensures that the score accounts for the proportion of each class.

Scaling

The underlying model used for SFS is sensitive to skewed data, thus a power transformation is applied to the data before running the SFS. The Yeo-Johnson transformation is a power transformation technique used to stabilize variance and make data more closely resemble a normal distribution. It works by applying a formula involving a parameter lambda ( ), which is estimated from the data. The transformation aims to find the optimal that minimizes skewness and makes the data distribution more symmetric and Gaussian-like. This makes it particularly well-suited for skewed datasets, as it helps to normalize the distribution.

[9]:
```python
transformer = PowerTransformer()
transformer.set_output(transform="pandas")
X = transformer.fit_transform(X)
X
```

[9]:

	radius2	texture2	perimeter2	area2	concavity2	concave_points2	\
0	1.604859	2.013091	1.048349	0.872311	2.580592	3.143171	
1	0.074949	1.016776	0.039760	0.191275	-1.432812	-0.781828	
2	0.281498	0.693706	0.174242	-0.514851	-1.991698	-2.737673	
3	-0.705899	0.637693	-0.484052	-0.733280	0.011452	-0.084759	
4	0.429160	0.238213	0.283893	0.132144	-0.940462	-0.690052	
..	…	…	…	…	…	…	
496	-2.025629	0.647689	-0.538153	-1.932207	0.809726	0.937731	
497	-0.913239	-0.859055	-0.652885	-0.769958	-0.682990	-0.659087	
498	0.226288	0.708279	-0.779578	-0.224841	0.690670	0.764135	
499	-0.234162	-0.529989	-0.168031	-0.184846	-1.471518	-1.118048	
500	0.312154	-0.220347	0.176633	0.139984	-0.362736	-0.315755	

```
     symmetry2  fractal_dimension2
0     1.722460            2.399088
1     0.253991           -0.682989
2     1.277235            1.531871
3    -1.090639           -0.125685
4     0.796561           -1.202170
..         …                   …
496   0.871294            1.641652
497   0.392668           -1.085314
498   1.131017            1.690222
499   0.132911           -1.846963
500  -0.105652            0.257162

[501 rows x 8 columns]
```

[ ]:
```
model = LogisticRegression(max_iter=10000)
cv = StratifiedKFold(n_splits=5)

sfs = SequentialFeatureSelector(
    model,
    n_features_to_select="auto",
    tol=1e-5,
    direction="backward",
    scoring="f1_weighted",
    cv=cv,
    n_jobs=-1,
)

sfs.fit(X, y)
selected_features = X.columns[sfs.get_support()] # type: ignore
selected_features
```

[ ]:
```
Index(['radius2', 'perimeter2', 'area2', 'concavity2', 'symmetry2',
       'fractal_dimension2'],
      dtype='object')
```

The SFS identified the following optimal feature subset:

`['radius2', 'perimeter2', 'area2', 'concavity2', 'symmetry2', 'fractal_dimension2']`

Comparison with Mutual Information (MI) Results:

- **Overlap:** The SFS results overlap significantly with the top MI features. `area2`, `radius2`, and `perimeter2` were selected by SFS and also ranked highly in the MI analysis, indicating their strong individual and combined predictive power. `texture2` was excluded by both methods indicating very little standalone or combined predictive properties.
- **Differences:**
  - `concave_points2` was not selected by SFS, likely due to the previously observed

high correlation to `concavity2`. Further, `concavity2` carries information about `concave_points2` by definition, since the former can only assume a value >0 if the latter is also >0.
  – `symmetry2` and `fractal_dimension2` were included by SFS even though they got relatively low individual scores in MI.

1.2 Build Datasets

In the following section, multiple datasets are built using different preprocessing and feature selection techniques.

Used Preprocessors are:

- **StandardScaler**: Scales features to have zero mean and unit variance. This is beneficial for algorithms sensitive to feature scales, such as those using distance metrics or regularization. It assumes the data is approximately normally distributed.
- **MinMaxScaler**: Scales features to a fixed range, typically [0, 1]. It preserves the shape of the original distribution and is useful when the algorithm requires features within a specific range.
- **PowerTransformer**: Applies Yeo-Johnson transformations to make the data more Gaussian-like. This helps stabilize variance and reduce skewness, which is useful for models that assume normality or perform better with symmetric distributions.
- **QuantileTransformer**: Transforms features based on quantiles to follow a uniform or normal distribution. It is robust to outliers and can handle various data distributions effectively, mapping them to a predefined distribution.

The choice of these four preprocessing techniques provides a comprehensive approach to handling feature scaling and distribution. Each method targets different characteristics and assumptions about the data: StandardScaler corrects for differences in scale, MinMaxScaler ensures all features fall within a specific range, PowerTransformer addresses skewness and promotes normality, and QuantileTransformer improves robustness to outliers and non-standard distributions. This allows for the selection of an optimal preprocessor for each individual model used in the modeling task.

Used Feature Selection techniques are:

- **Include all features**: Uses the complete set of original features without any selection process. Does not consider `compactness2` as this was previously determined to be excluded from analysis.
- **Mutual Information**: Selects features based on their mutual information score with the target variable, capturing non-linear dependencies.
- **Sequential Feature Selector (SFS)**: Selects features iteratively based on model performance (using Logistic Regression with F1-weighted score in this case).

[11]:
```
data_path = os.path.join(os.path.dirname(os.getcwd()), "data", "raw_data.csv")
df = pd.read_csv(data_path)
df["ID"] = df["ID"].astype("category")
df["Diagnosis"] = df["Diagnosis"].astype("category")
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 501 entries, 0 to 500
```

```
Data columns (total 11 columns):
 #   Column              Non-Null Count  Dtype
---  ------              --------------  -----
 0   ID                  501 non-null    category
 1   Diagnosis           501 non-null    category
 2   radius2             501 non-null    float64
 3   texture2            501 non-null    float64
 4   perimeter2          501 non-null    float64
 5   area2               501 non-null    float64
 6   compactness2        501 non-null    float64
 7   concavity2          501 non-null    float64
 8   concave_points2     501 non-null    float64
 9   symmetry2           501 non-null    float64
 10  fractal_dimension2  501 non-null    float64
dtypes: category(2), float64(9)
memory usage: 57.0 KB
```

[12]:

```
def fs_all_features() -> tuple[pd.DataFrame, pd.Series]:
    """Returns the full feature set and the target variable.

    Returns:
        tuple[pd.DataFrame, pd.Series]: A tuple containing the feature␣
↪DataFrame (X)
                                        and the target Series (y).
    """
    X = df[
        [
            "radius2",
            "texture2",
            "perimeter2",
            "area2",
            "concavity2",
            "concave_points2",
            "symmetry2",
            "fractal_dimension2",
        ]
    ]

    y = df["Diagnosis"]

    return X, y


def fs_mi() -> tuple[pd.DataFrame, pd.Series]:
    """Returns the optimal subset of features determined by MI and the target␣
↪variable.
```

```python
    Returns:
        tuple[pd.DataFrame, pd.Series]: A tuple containing the feature␣
↪DataFrame (X)
                                        and the target Series (y).
    """
    X = df[
        [
            "radius2",
            "perimeter2",
            "area2",
            "concavity2",
            "concave_points2",
        ]
    ]

    y = df["Diagnosis"]

    return X, y


def fs_sfs() -> tuple[pd.DataFrame, pd.Series]:
    """Returns the optimal subset of features determined by SFS and the target␣
↪variable.

    Returns:
        tuple[pd.DataFrame, pd.Series]: A tuple containing the feature␣
↪DataFrame (X)
                                        and the target Series (y).
    """
    X = df[
        [
            "radius2",
            "perimeter2",
            "area2",
            "concavity2",
            "symmetry2",
            "fractal_dimension2",
        ]
    ]

    y = df["Diagnosis"]

    return X, y
```

[13]:
```python
def get_scaler_suffix(scaler) -> str:
    """
    Returns a short string suffix for the given scaler type.
```

```
    Args:
        scaler: A scaler instance.

    Raises:
        TypeError: If the scaler type is not recognized.

    Returns:
        str: Short suffix representing the scaler type.
    """
    if type(scaler).__name__ == "StandardScaler":
        return "std"
    elif type(scaler).__name__ == "MinMaxScaler":
        return "minmax"
    elif type(scaler).__name__ == "QuantileTransformer":
        return "qt"
    elif type(scaler).__name__ == "PowerTransformer":
        return "pt"
    else:
        raise TypeError
```

[ ]:

```
# This cell generates and saves multiple preprocessed datasets using different␣
↪scalers and feature selection methods.
# For each scaler (StandardScaler, MinMaxScaler, QuantileTransformer,␣
↪PowerTransformer), it:
#   1. Applies the scaler to three feature sets: all features, mi-selected␣
↪features, and SFS-selected features.
#   2. Saves the fitted scaler object for each combination.
#   3. Combines the scaled features with the target variable and saves the␣
↪resulting dataset as a parquet file.
# This enables downstream modeling with consistent preprocessing and feature␣
↪selection pipelines.

SCALERS = [
    StandardScaler(),
    MinMaxScaler(),
    QuantileTransformer(random_state=42, n_quantiles=df.shape[0]),
    PowerTransformer(),
]

for scaler in SCALERS:
    scaler.set_output(transform="pandas")
    scaler_suffix = get_scaler_suffix(scaler)

    X, y = fs_all_features()
    X = scaler.fit_transform(X)
    joblib.dump(
```

```
        scaler,
        os.path.join(
            os.path.dirname(os.getcwd()), "artifacts", "preprocessors",␣
↪scaler_suffix + "_all" + ".pkl"
        ),
    )
    dataset = pd.concat([X, y], axis=1)
    dataset_name = "processed_data_" + scaler_suffix + "_all" + ".parquet"
    dataset.to_parquet(
        os.path.join(
            os.path.dirname(os.getcwd()), "data", "processed_data", dataset_name
        )
    )

    X, y = fs_mi()
    X = scaler.fit_transform(X)
    joblib.dump(
        scaler,
        os.path.join(
            os.path.dirname(os.getcwd()), "artifacts", "preprocessors",␣
↪scaler_suffix + "_mi" + ".pkl"
        ),
    )
    dataset = pd.concat([X, y], axis=1)
    dataset_name = "processed_data_" + scaler_suffix + "_mi" + ".parquet"
    dataset.to_parquet(
        os.path.join(
            os.path.dirname(os.getcwd()), "data", "processed_data", dataset_name
        )
    )

    X, y = fs_sfs()
    X = scaler.fit_transform(X)
    joblib.dump(
        scaler,
        os.path.join(
            os.path.dirname(os.getcwd()), "artifacts", "preprocessors",␣
↪scaler_suffix + "_sfs" + ".pkl"
        ),
    )
    dataset = pd.concat([X, y], axis=1)
    dataset_name = "processed_data_" + scaler_suffix + "_sfs" + ".parquet"
    dataset.to_parquet(
        os.path.join(
            os.path.dirname(os.getcwd()), "data", "processed_data", dataset_name
        )
    )
```

1.3 Summary

- **Feature Selection Analysis:**
  - **Mutual Information (MI):** MI scores are calculated to assess the individual predictive power of each feature regarding the `Diagnosis`. Features related to size (`area2`, `perimeter2`, `radius2`) and concave points (`concave_points2`, `concavity2`) showed the highest MI scores.
  - **Sequential Feature Selection (SFS):** Backward SFS with a `LogisticRegression` model (evaluated using F1-weighted score and 5-fold stratified cross-validation) is performed after applying a `PowerTransformer` to handle potential data skewness. SFS identified `['radius2', 'perimeter', 'area2', 'concavity2', 'fractal_dimension2', 'symmetry2']` as the optimal feature subset.
  - **Comparison:** The results from MI and SFS are compared, highlighting overlaps and differences, suggesting that SFS considers feature interactions and redundancy, unlike MI.
- **Dataset Generation:**
  - Three feature sets are defined: all features, MI-selected features, and SFS-selected features.
  - Four different scaling techniques (`StandardScaler`, `MinMaxScaler`, `QuantileTransformer`, `PowerTransformer`) are applied to each feature set.
  - For each combination of feature set and scaler:
    * The scaler is fitted and saved.
    * The resulting processed dataset is saved.

This process generates multiple preprocessed datasets, enabling experimentation with different feature sets and scaling methods during the subsequent modeling phase. The results from both feature selection methods confirm the observations made in the multivariate analysis in the data exploration step.