navis-baye

January 23, 2025

[1]:
```
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.naive_bayes import GaussianNB
from sklearn.metrics import accuracy_score, classification_report,␣
↪confusion_matrix, mean_squared_error, mean_absolute_error, r2_score
import matplotlib.pyplot as plt
import seaborn as sns
```

[2]:
```
# Load Titanic dataset
data = pd.read_csv(r"C:\Users\91703\OneDrive\Desktop\TITANIC.csv")
```

[25]:
```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 891 entries, 0 to 890
Data columns (total 7 columns):
 #   Column    Non-Null Count  Dtype
---  ------    --------------  -----
 0   Survived  891 non-null    int64
 1   Pclass    891 non-null    int64
 2   Sex       0 non-null      float64
 3   Age       891 non-null    float64
 4   SibSp     891 non-null    int64
 5   Parch     891 non-null    int64
 6   Fare      891 non-null    float64
dtypes: float64(3), int64(4)
memory usage: 48.9 KB
```

[27]:
```
data.head()
```

[27]:

	Survived	Pclass	Sex	Age	SibSp	Parch	Fare
0	0	3	NaN	22.0	1	0	7.2500
1	1	1	NaN	38.0	1	0	71.2833
2	1	3	NaN	26.0	0	0	7.9250
3	1	1	NaN	35.0	1	0	53.1000
4	0	3	NaN	35.0	0	0	8.0500

[29]:
```
data.isnull().sum()
```

[29]:
```
Survived    0
Pclass      0
Sex         0
Age         0
SibSp       0
Parch       0
Fare        0
dtype: int64
```

[28]:
```
data.dropna(subset = ['Sex'], inplace=True)
```

[ ]:
```

```

[3]:
```
# Select relevant columns columns and convert categorical variables
data = data[['Survived', 'Pclass', 'Sex', 'Age', 'SibSp', 'Parch', 'Fare']]
data['Sex'] = data['Sex'].map({'male': 0, 'female': 1})
```

[4]:
```
# Handle missing Age values
data['Age'] = pd.to_numeric(data['Age'], errors='coerce')
data['Age'] = data['Age'].fillna(data['Age'].mean())
```

[5]:
```
# Define target variable (y) and feature variables (X)
y = data['Survived']
X = data[['Pclass', 'Sex', 'Age', 'SibSp', 'Parch', 'Fare']]
```

[6]:
```
# Split data into training and testing sets
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
 ↪random_state=42)
```

[7]:
```
# Create and train Naive Bayes Classifier model
from sklearn.naive_bayes import GaussianNB
model = GaussianNB()
model.fit(X_train, y_train)
```

[7]:
```
GaussianNB()
```

[8]:
```
# Make predictions on test data
y_pred = model.predict(X_test)
print(y_pred)
```

```
[0 0 0 1 1 1 1 0 1 1 0 0 0 0 0 1 0 1 0 0 0 0 1 0 0 0 0 0 0 1 0 1 0 1 0 0 0
 1 1 0 0 0 0 0 1 0 0 0 0 1 1 1 0 1 0 1 0 1 1 1 0 1 1 0 0 1 0 0 0 1 1 1 0 1
 0 0 1 1 1 1 0 1 1 0 0 1 1 1 0 0 0 0 0 0 0 0 0 0 1 0 0 0 1 0 0 0 1 0 0 0 1
```

```
0 1 0 1 0 1 0 1 1 0 1 1 0 0 0 1 1 1 0 1 0 0 1 0 1 1 0 0 1 0 1 0 0 0 1 0 0
1 0 0 0 0 1 0 0 0 1 1 1 0 0 0 1 0 1 0 1 0 0 1 1 0 1 0 0 0 1 1]
```

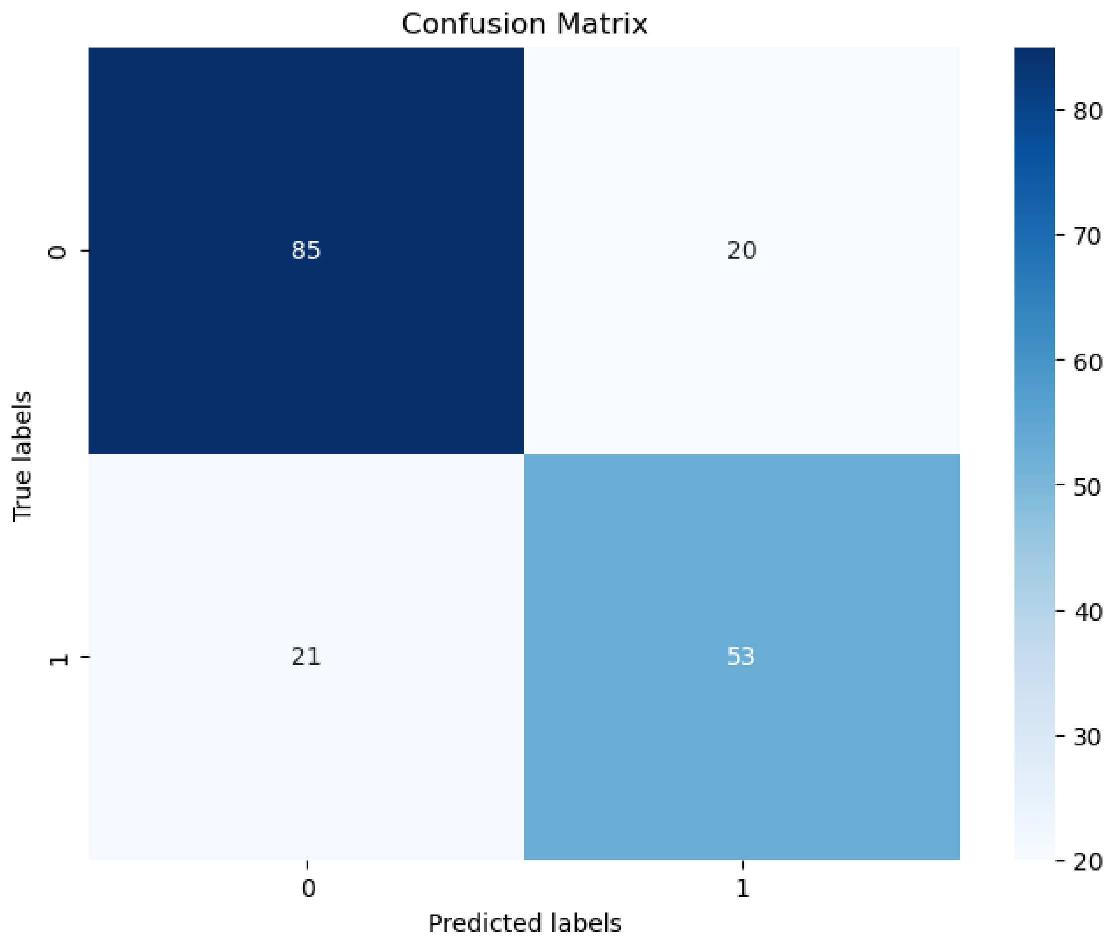
[9]:

```python
# Evaluate model performance
from sklearn.metrics import accuracy_score, classification_report,␣
↪confusion_matrix
accuracy = accuracy_score(y_test, y_pred)
print("Accuracy:", accuracy)
print("Classification Report:", classification_report(y_test, y_pred))
print("Confusion Matrix:", confusion_matrix(y_test, y_pred))
```

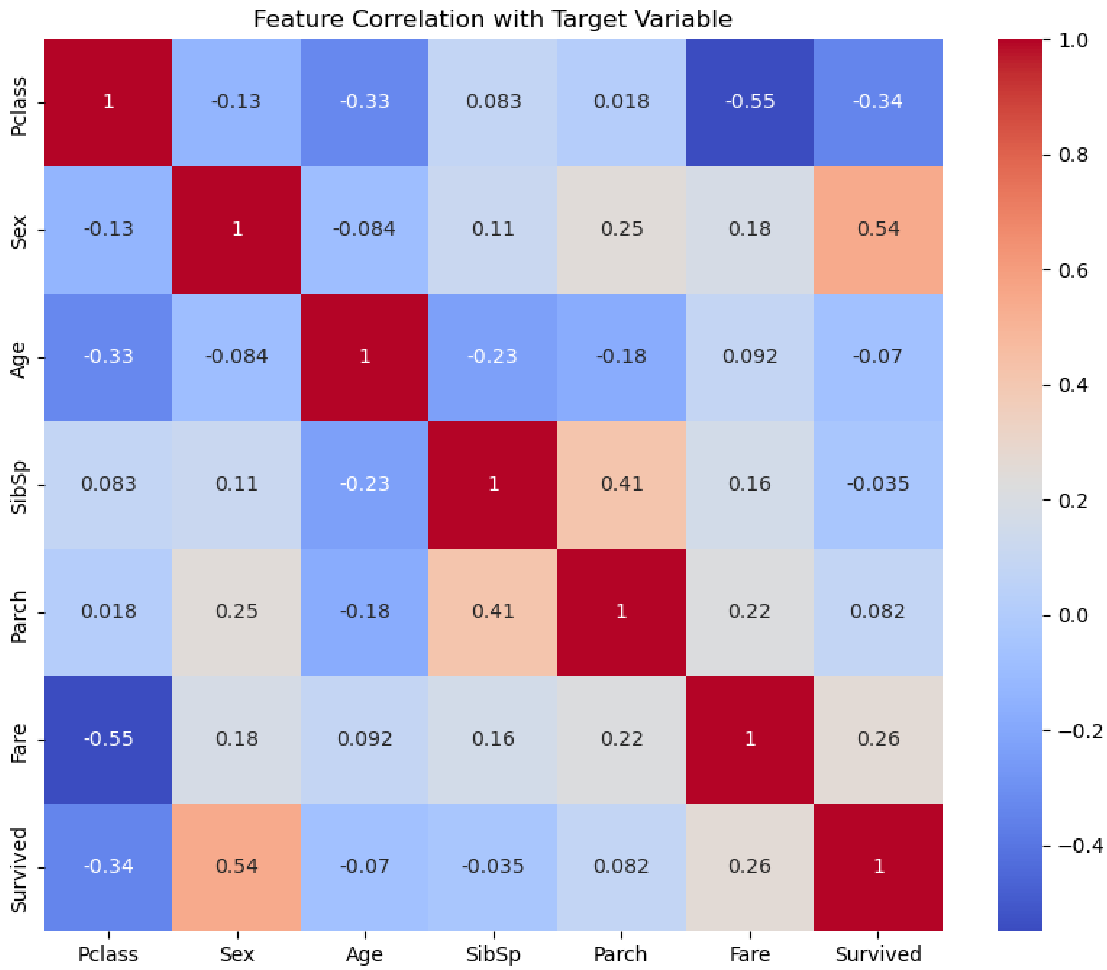
```
Accuracy: 0.770949720670391
Classification Report:               precision    recall  f1-score   support

           0       0.80      0.81      0.81       105
           1       0.73      0.72      0.72        74

    accuracy                           0.77       179
   macro avg       0.76      0.76      0.76       179
weighted avg       0.77      0.77      0.77       179

Confusion Matrix: [[85 20]
 [21 53]]
```

[10]:

```python
# Plot Confusion Matrix
plt.figure(figsize=(8, 6))
sns.heatmap(confusion_matrix(y_test, y_pred), annot=True, cmap='Blues')
plt.xlabel("Predicted labels")
plt.ylabel("True labels")
plt.title("Confusion Matrix")
plt.show()
```




```
[11]: # Plot feature importance - Note: Naive Bayes doesn't have feature importance␣
       ↪like tree-based models
      # But we can plot feature correlation with target variable
      corr = data[['Pclass', 'Sex', 'Age', 'SibSp', 'Parch', 'Fare', 'Survived']].
       ↪corr()
      plt.figure(figsize=(10, 8))
      sns.heatmap(corr, annot=True, cmap='coolwarm')
      plt.title("Feature Correlation with Target Variable")
      plt.show()
```



[14]:
```
# Separate features (X) and target variable (y)
X = data[['Pclass', 'Sex', 'SibSp', 'Parch', 'Fare']]
y = data['Age']
```

[15]:
```
# Split data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,␣
↪random_state=42)
```

[30]:
```
# from sklearn.linear_model import BayesianRidge

model = BayesianRidge()
model.fit(X_train, y_train)
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
Cell In[30], line 4
      2 from sklearn.linear_model import BayesianRidge
```

```
      3 model = BayesianRidge()
----> 4 model.fit(X_train, y_train)

File ~\anaconda3\Lib\site-packages\sklearn\base.py:1474, in _fit_context.
 ↪<locals>.decorator.<locals>.wrapper(estimator, *args, **kwargs)
   1467     estimator._validate_params()
   1469 with config_context(
   1470     skip_parameter_validation=(
   1471         prefer_skip_nested_validation or global_skip_validation
   1472     )
   1473 ):
-> 1474     return fit_method(estimator, *args, **kwargs)

File ~\anaconda3\Lib\site-packages\sklearn\linear_model\_bayes.py:296, in␣
 ↪BayesianRidge.fit(self, X, y, sample_weight)
    274 """Fit the model.
    275
    276 Parameters
   (…)
    292     Returns the instance itself.
    293 """
    294 max_iter = _deprecate_n_iter(self.n_iter, self.max_iter)
--> 296 X, y = self._validate_data(X, y, dtype=[np.float64, np.float32],␣
 ↪y_numeric=True)
    297 dtype = X.dtype
    299 if sample_weight is not None:

File ~\anaconda3\Lib\site-packages\sklearn\base.py:650, in BaseEstimator.
 ↪_validate_data(self, X, y, reset, validate_separately, cast_to_ndarray,␣
 ↪**check_params)
    648         y = check_array(y, input_name="y", **check_y_params)
    649     else:
--> 650         X, y = check_X_y(X, y, **check_params)
    651     out = X, y
    653 if not no_val_X and check_params.get("ensure_2d", True):

File ~\anaconda3\Lib\site-packages\sklearn\utils\validation.py:1263, in␣
 ↪check_X_y(X, y, accept_sparse, accept_large_sparse, dtype, order, copy,␣
 ↪force_all_finite, ensure_2d, allow_nd, multi_output, ensure_min_samples,␣
 ↪ensure_min_features, y_numeric, estimator)
   1258         estimator_name = _check_estimator_name(estimator)
   1259     raise ValueError(
   1260         f"{estimator_name} requires y to be passed, but the target y is␣
 ↪None"
   1261     )
-> 1263 X = check_array(
   1264     X,
   1265     accept_sparse=accept_sparse,
```

```
   1266     accept_large_sparse=accept_large_sparse,
   1267     dtype=dtype,
   1268     order=order,
   1269     copy=copy,
   1270     force_all_finite=force_all_finite,
   1271     ensure_2d=ensure_2d,
   1272     allow_nd=allow_nd,
   1273     ensure_min_samples=ensure_min_samples,
   1274     ensure_min_features=ensure_min_features,
   1275     estimator=estimator,
   1276     input_name="X",
   1277 )
   1279 y = _check_y(y, multi_output=multi_output, y_numeric=y_numeric, estimator=estimator)
   1281 check_consistent_length(X, y)

File ~\anaconda3\Lib\site-packages\sklearn\utils\validation.py:1049, in check_array(array, accept_sparse, accept_large_sparse, dtype, order, copy, force_all_finite, ensure_2d, allow_nd, ensure_min_samples, ensure_min_features, estimator, input_name)
   1043     raise ValueError(
   1044         "Found array with dim %d. %s expected <= 2."
   1045         % (array.ndim, estimator_name)
   1046     )
   1048 if force_all_finite:
-> 1049     _assert_all_finite(
   1050         array,
   1051         input_name=input_name,
   1052         estimator_name=estimator_name,
   1053         allow_nan=force_all_finite == "allow-nan",
   1054     )
   1056 if copy:
   1057     if _is_numpy_namespace(xp):
   1058         # only make a copy if `array` and `array_orig` may share memory

File ~\anaconda3\Lib\site-packages\sklearn\utils\validation.py:126, in _assert_all_finite(X, allow_nan, msg_dtype, estimator_name, input_name)
    123 if first_pass_isfinite:
    124     return
--> 126 _assert_all_finite_element_wise(
    127     X,
    128     xp=xp,
    129     allow_nan=allow_nan,
    130     msg_dtype=msg_dtype,
    131     estimator_name=estimator_name,
    132     input_name=input_name,
    133 )
```

```
File ~\anaconda3\Lib\site-packages\sklearn\utils\validation.py:175, in _assert_all_finite_element_wise(X, xp, allow_nan, msg_dtype, estimator_name, input_name)
    158 if estimator_name and input_name == "X" and has_nan_error:
    159     # Improve the error message on how to handle missing values in
    160     # scikit-learn.
    161     msg_err += (
    162         f"\n{estimator_name} does not accept missing values"
    163         " encoded as NaN natively. For supervised learning, you might want"
   (…)
    173         "#estimators-that-handle-nan-values"
    174     )
--> 175 raise ValueError(msg_err)

ValueError: Input X contains NaN.
BayesianRidge does not accept missing values encoded as NaN natively. For supervised learning, you might want to consider sklearn.ensemble.HistGradientBoostingClassifier and Regressor which accept missing values encoded as NaNs natively. Alternatively, it is possible to preprocess the data, for instance by using an imputer transformer in a pipeline or drop samples with missing values. See https://scikit-learn.org/stable/modules/impute.html You can find a list of all estimators that handle NaN values at the following page: https://scikit-learn.org/stable/modules/impute.html#estimators-that-handle-nan-values
```

[17]:

```python
# Make predictions on test data
y_pred = model.predict(X_test)
print(y_pred)
```

```
[30.0036202  33.71844352 30.84651263 24.94828922 15.47466078 37.94189475
 32.49441394 23.57690294 32.49441394 27.35040362 28.32853175 29.94263848
 28.66065294 29.54517209 31.93358046 34.52109824 31.82624603 28.22877406
 31.93358046 33.93058431 29.15733613 50.2495245  25.37056818 19.18591446
 25.71490402  7.18230201 34.1038374  31.93358046  7.18230201 25.96931882
 29.15733613 32.49441394 38.86785889 26.2378375  29.94263848 24.60156922
 42.50423025 32.49441394 33.24963738 29.15733613 26.58508333 19.22202381
 29.94263848 32.49441394 20.79439118 31.25798235 29.75350365 19.18591446
 24.8422381  20.51890476  2.92833333 42.906      40.62332451 27.85160317
 32.49441394 22.34148676 33.71844352 27.2094101  40.33020519 29.42028989
 28.62839075 31.78926929 33.37512882 42.40774412 32.49441394 33.71844352
 21.7356979  29.21418457 18.09517353 51.37416667  2.90681746 21.08645333
 39.23928467 46.70766667 25.96931882 29.6954598  32.49441394 23.6758527
 31.93358046 17.92009608  5.92715063 33.71844352 42.64463137 27.46650018
 38.86785889 26.04584118 24.18998235 28.22959039 34.66786206 31.46821955
 19.07407843  5.74862302 50.69207081 24.19316827 29.15733613 25.96931882
 36.25692857 29.94263848 31.93358046 29.15733613 35.55877745 31.277
 28.33954762 24.8422381  29.94263848 25.99151042 47.53       35.03481735
 26.20136276 33.24963738 36.25692857 31.46821955 44.45266667 39.64129911
```

```
35.65101901 20.09333333 35.57918319 31.93358046 27.62590476 31.25798235
 7.79877167 29.06163137 49.45016667 12.33533333 31.46821955 50.2495245
42.40774412 39.73266475 25.73126926 29.94263848 29.42028989 30.0036202
29.78826029 27.17621691 26.25566667 24.60156922 27.79483333 28.62839075
26.2378375  27.31182892 29.11033186 33.37512882 31.93358046 33.37512882
35.94166667 23.62149118 31.93358046 33.71844352 37.47274412 29.94263848
28.91967136 27.55360315 24.24683442 30.09151397 29.94263848 24.8422381
29.58947197 24.26549961 33.37512882 30.21039118 26.70335334 50.2495245
31.93358046 32.77833333 29.11033186 45.57466667 33.71844352 42.53942857
29.15733613 24.8422381  44.27666667 22.93836474 39.64129911 30.0036202
31.277      29.54517209 39.105      33.71844352 17.63025   ]
```

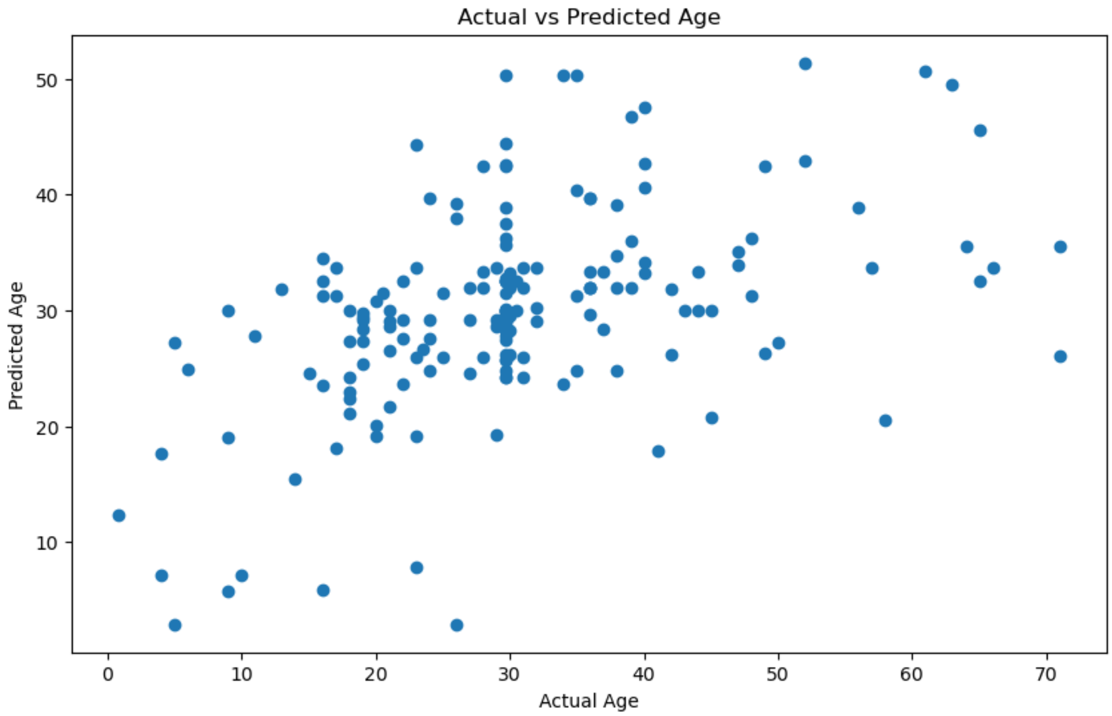
[18]:

```
# Evaluate model performance using metrics
mse = mean_squared_error(y_test, y_pred)
mae = mean_absolute_error(y_test, y_pred)
r2 = r2_score(y_test, y_pred)
print("Model Performance Metrics:")
print(f"Mean Squared Error (MSE): {mse:.2f}")
print(f"Mean Absolute Error (MAE): {mae:.2f}")
print(f"R-Squared (R2): {r2:.2f}")
```

```
Model Performance Metrics:
Mean Squared Error (MSE): 134.22
Mean Absolute Error (MAE): 8.39
R-Squared (R2): 0.21
```

[19]:

```
# Plot actual vs predicted values
plt.figure(figsize=(10,6))
plt.scatter(y_test, y_pred)
plt.xlabel("Actual Age")
plt.ylabel("Predicted Age")
plt.title("Actual vs Predicted Age")
plt.show()
```


Actual vs Predicted Age
Predicted Age
Actual Age
0
10
20
30
40
50
60
70
10
20
30
40
50