Lab2MD

April 8, 2025

[1]:

```
# Data manipulation and visualization
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

# Model development
from sklearn.model_selection import train_test_split, cross_val_score,␣
 ↪GridSearchCV
from sklearn.preprocessing import StandardScaler, MinMaxScaler, OneHotEncoder
from sklearn.pipeline import Pipeline
from sklearn.compose import ColumnTransformer
from sklearn.impute import SimpleImputer

# Models
from sklearn.linear_model import LogisticRegression
from sklearn.neighbors import KNeighborsClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.tree import plot_tree

# Model evaluation
from sklearn.metrics import (
    accuracy_score,
    precision_score,
    recall_score,
    f1_score,
    classification_report,
    confusion_matrix
)

# For displaying results nicely in the notebook
from IPython.display import display, HTML

# Set visualization style
plt.style.use('seaborn-v0_8-whitegrid')
sns.set_palette('viridis')
```

```
# For reproducibility
import random
random.seed(42)
np.random.seed(42)
```

[2]:

```
# Read the student performance dataset
df = pd.read_csv('student_performance_data.csv')

# Remove StudentID column from the dataset, irrelevant for what I am finding
df = df.drop(columns=['StudentID'])

# Basic info about the dataset
print("Dataset shape:", df.shape)
print("\nFirst 5 rows of the dataset:")
display(df.head())

# Descriptive statistics
print("\nDescriptive statistics:")
display(df.describe())

# Check for missing values
print("\nMissing values per column:")
display(df.isnull().sum())

# Display information about data types and non-null counts
print("\nDataset information:")
display(df.info())
```

```
---------------------------------------------------------------------------
FileNotFoundError                         Traceback (most recent call last)
Cell In[2], line 2
      1 # Read the student performance dataset
----> 2 df = pd.read_csv('student_performance_data.csv')
      4 # Remove StudentID column from the dataset, irrelevant for what I am␣
  ↪finding
      5 df = df.drop(columns=['StudentID'])

File /opt/conda/envs/anaconda-panel-2023.05-py310/lib/python3.11/site-packages/
  ↪pandas/io/parsers/readers.py:912, in read_csv(filepath_or_buffer, sep,␣
  ↪delimiter, header, names, index_col, usecols, dtype, engine, converters,␣
  ↪true_values, false_values, skipinitialspace, skiprows, skipfooter, nrows,␣
  ↪na_values, keep_default_na, na_filter, verbose, skip_blank_lines, parse_dates ␣
  ↪infer_datetime_format, keep_date_col, date_parser, date_format, dayfirst,␣
  ↪cache_dates, iterator, chunksize, compression, thousands, decimal,␣
  ↪lineterminator, quotechar, quoting, doublequote, escapechar, comment,␣
  ↪encoding, encoding_errors, dialect, on_bad_lines, delim_whitespace,␣
  ↪low_memory, memory_map, float_precision, storage_options, dtype_backend)
    899 kwds_defaults = _refine_defaults_read(
    900     dialect,
    901     delimiter,
   (…)
    908     dtype_backend=dtype_backend,
    909 )
    910 kwds.update(kwds_defaults)
--> 912 return _read(filepath_or_buffer, kwds)

File /opt/conda/envs/anaconda-panel-2023.05-py310/lib/python3.11/site-packages/
 ↪pandas/io/parsers/readers.py:577, in _read(filepath_or_buffer, kwds)
    574 _validate_names(kwds.get("names", None))
    576 # Create the parser.
--> 577 parser = TextFileReader(filepath_or_buffer, **kwds)
    579 if chunksize or iterator:
    580     return parser

File /opt/conda/envs/anaconda-panel-2023.05-py310/lib/python3.11/site-packages/
 ↪pandas/io/parsers/readers.py:1407, in TextFileReader.__init__(self, f, engine ␣
 ↪**kwds)
   1404     self.options["has_index_names"] = kwds["has_index_names"]
   1406 self.handles: IOHandles | None = None
-> 1407 self._engine = self._make_engine(f, self.engine)

File /opt/conda/envs/anaconda-panel-2023.05-py310/lib/python3.11/site-packages/
 ↪pandas/io/parsers/readers.py:1661, in TextFileReader._make_engine(self, f,␣
 ↪engine)
   1659     if "b" not in mode:
   1660         mode += "b"
-> 1661 self.handles = get_handle(
   1662     f,
   1663     mode,
   1664     encoding=self.options.get("encoding", None),
   1665     compression=self.options.get("compression", None),
   1666     memory_map=self.options.get("memory_map", False),
   1667     is_text=is_text,
   1668     errors=self.options.get("encoding_errors", "strict"),
   1669     storage_options=self.options.get("storage_options", None),
   1670 )
   1671 assert self.handles is not None
   1672 f = self.handles.handle

File /opt/conda/envs/anaconda-panel-2023.05-py310/lib/python3.11/site-packages/
 ↪pandas/io/common.py:859, in get_handle(path_or_buf, mode, encoding,␣
 ↪compression, memory_map, is_text, errors, storage_options)
    854 elif isinstance(handle, str):
    855     # Check whether the filename is to be opened in binary mode.
    856     # Binary mode does not support 'encoding' and 'newline'.
    857     if ioargs.encoding and "b" not in ioargs.mode:
    858         # Encoding
```

```
--> 859         handle = open(
    860             handle,
    861             ioargs.mode,
    862             encoding=ioargs.encoding,
    863             errors=errors,
    864             newline="",
    865         )
    866     else:
    867         # Binary mode
    868         handle = open(handle, ioargs.mode)

FileNotFoundError: [Errno 2] No such file or directory: 'student_performance_data.csv'
```

[ ]:

```
# Step 1: Create binary target based on GPA
gpa_column = 'GPA'
df['target'] = (df[gpa_column] >= 3.2).astype(int)

# Display the distribution of the target variable
print("Target distribution:")
print(df['target'].value_counts())
print(f"Percentage of students with GPA   3.2: {df['target'].mean()*100:.2f}%")

# Step 2: Prepare the data
# Exclude GPA and target from features
X = df.drop(columns=[gpa_column, 'target'])
y = df['target']

# Handle categorical features
categorical_features = X.select_dtypes(include=['object', 'category']).columns.tolist()
numerical_features = X.select_dtypes(include=['int64', 'float64']).columns.tolist()

# Create preprocessing pipelines
numeric_transformer = Pipeline(steps=[
    ('imputer', SimpleImputer(strategy='median')),
    ('scaler', StandardScaler())
])

categorical_transformer = Pipeline(steps=[
    ('imputer', SimpleImputer(strategy='most_frequent')),
    ('onehot', OneHotEncoder(handle_unknown='ignore'))
])

# Combine preprocessing steps
```

```
preprocessor = ColumnTransformer(
    transformers=[
        ('num', numeric_transformer, numerical_features),
        ('cat', categorical_transformer, categorical_features)
    ])

# Step 3: Split the 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, stratify=y)

# Step 4: Build the models
# Logistic Regression
log_reg_pipeline = Pipeline(steps=[
    ('preprocessor', preprocessor),
    ('classifier', LogisticRegression(max_iter=1000, random_state=42))
])

# k-NN
knn_pipeline = Pipeline(steps=[
    ('preprocessor', preprocessor),
    ('classifier', KNeighborsClassifier(n_neighbors=5))
])

# Decision Tree
dt_pipeline = Pipeline(steps=[
    ('preprocessor', preprocessor),
    ('classifier', DecisionTreeClassifier(random_state=42))
])

# Dictionary of models
models = {
    'Logistic Regression': log_reg_pipeline,
    'k-Nearest Neighbors': knn_pipeline,
    'Decision Tree': dt_pipeline
}

# Step 5: Train and evaluate each model
results = {}

for name, model in models.items():
    print(f"\n{'-'*50}")
    print(f"Training and evaluating {name}...")

    # Train the model
    model.fit(X_train, y_train)

    # Make predictions
    y_pred = model.predict(X_test)

    # Calculate metrics
    accuracy = accuracy_score(y_test, y_pred)

    # Store results
    results[name] = {
        'accuracy': accuracy,
        'predictions': y_pred
    }

    # Print classification report
    print(f"\nClassification Report for {name}:")
    print(classification_report(y_test, y_pred))

# Create and display confusion matrix
    cm = confusion_matrix(y_test, y_pred)
    plt.figure(figsize=(8, 6))
    sns.heatmap(cm, annot=True, fmt='d', cmap='Blues',
                xticklabels=['GPA < 3.2', 'GPA   3.2'],
                yticklabels=['GPA < 3.2', 'GPA   3.2'])
    plt.xlabel('Predicted')
    plt.ylabel('Actual')
    plt.title(f'Confusion Matrix - {name}')
    plt.tight_layout()
    plt.show()

# Step 6: Compare model performance
model_comparison = pd.DataFrame({
    'Model': list(results.keys()),
    'Accuracy': [results[model]['accuracy'] for model in results]
})

# Sort by accuracy
model_comparison = model_comparison.sort_values('Accuracy', ascending=False).
  ↪reset_index(drop=True)

# Display comparison
print("\nModel Comparison:")
display(model_comparison)

# Visualize model comparison
plt.figure(figsize=(10, 6))
sns.barplot(x='Model', y='Accuracy', data=model_comparison)
plt.title('Model Accuracy Comparison')
plt.ylim(0, 1)
plt.xticks(rotation=45)
plt.tight_layout()
plt.show()
```