ClassPass How-To Guide

Overview

This guide walks through how the ClassPass system was built, including:

- Loading and cleaning a CSV dataset
- Building a preprocessing pipeline
- Implementing three AI methods from scratch:
    - k-Nearest Neighbours
    - Decision Tree
    - Bayesian Network
- Evaluating all models

This guide assumes a standard level of familiarity with Python, scikit-learn, and AI concepts

Dataset Cleaning & Preprocessing

Processing The CSV

The raw file includes semicolons, tabs, quotes and spacing issues, thus we split on semicolons and stripped whitespace.

Generating the Binary Target

We also mapped the three class labels to a binary output:

- At Risk: Student exhibits signs of a student that is likely to dropout
- Continue: Student exhibits signs of a student that is likely to stay enrolled or graduate

```python
df["BinaryTarget"] = df["Target"].map({
    "Dropout": "At Risk",
    "Enrolled": "Continue",
    "Graduate": "Continue"
})
```

After this, we also dropped the “Target” column to ensure there was no data leakage.

Inferring Numeric vs Categorical Features

We detected types using pandas dtype.

```
cat_cols = [c for c in df.columns if df[c].dtype == object]
num_cols = [c for c in df.columns if c not in cat_cols and c != "BinaryTarget"]
```

One-Hot Encoding + Scaling

We used OneHotEncoder for categorical features and StandardScaler for numeric ones

```
from sklearn.preprocessing import OneHotEncoder, StandardScaler
from sklearn.compose import ColumnTransformer

preprocessor = ColumnTransformer([
    ("cat", OneHotEncoder(handle_unknown="ignore"), cat_cols),
    ("num", StandardScaler(), num_cols),
])
```

Splitting Data

To split our data, we used a 70/15/15 split.

```
X = df.drop(columns=["BinaryTarget"])
y = df["BinaryTarget"]

X_trainval, X_test, y_trainval, y_test = train_test_split(
    X, y, test_size=0.15, stratify=y, random_state=42)

X_train, X_val, y_train, y_val = train_test_split(
    X_trainval, y_trainval, test_size=0.176, stratify=y_trainval)
```

70% for training, 15% for validation and another 15% for testing.

Implementing the Models

k-Nearest Neighbours

We computed distances using Euclidean norm:

```
dists = np.linalg.norm(self.X_train - x, axis=1)
```

Then selected the k smallest distances:

```
neighbors = np.argsort(dists)[:self.k]
labels = self.y_train[neighbors]
```

With our final prediction being the majority vote.

Our model does this for various k values and outputs the results:

```
kNN validation results:
k=3  → F1=0.780
k=5  → F1=0.764
k=7  → F1=0.771
k=9  → F1=0.779
k=11 → F1=0.760
```

Decision Tree

We built:

- Entropy and gini impurity
- Recursive node splitting
- Stopping conditions:
  - Max depth
  - Min samples split

```
def entropy(y):
    p = np.bincount(y) / len(y)
    return -np.sum([pi * np.log2(pi) for pi in p if pi > 0])
```

Validation results:

```
depth=3 →  F1=0.723
depth=5 →  F1=0.735  (best)
depth=7 →  F1=0.727
depth=9 →  F1=0.731
```

Bayesian Network*

We created a small model that uses three binary indicator features:

- LowGrades — 1 if first semester grade < threshold
- FinancialRisk — 1 if the student is a debtor
- LowEngagement — 1 if the student has few attended courses

We then estimated:

- P(LowGrades=1), P(FinancialRisk=1), P(LowEngagement=1)
- P(AtRisk=1 | LowGrades, FinancialRisk, LowEngagement)

Training uses simple frequency estimation:

```
LG = (df["Curricular units 1st sem (grade)"] < self.grade_thresh).astype(int)
FR = (df["Debtor"] == 1).astype(int)
LE = (df["Curricular units 1st sem (approved)"] < self.engagement_thresh).astype(int)
Y  = (df[target_col] == "At Risk").astype(int)
```

For each combination, we compute:

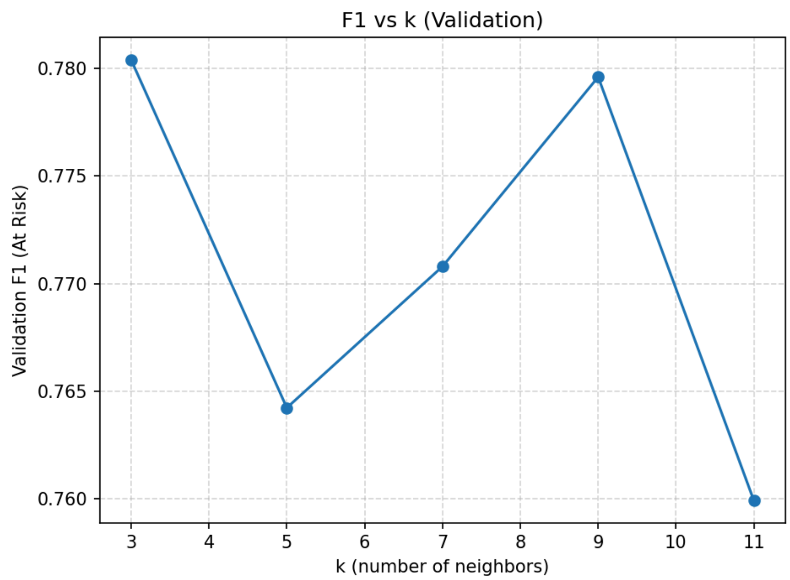
```
mask = (LG == lg) & (FR == fr) & (LE == le)
p = Y[mask].mean()
table[(lg, fr, le)] = p
```

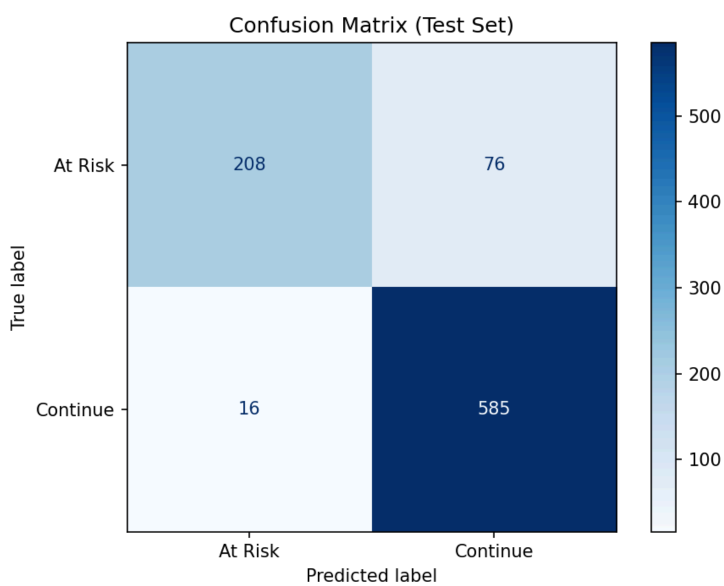
Inference is by direct lookup:

```
LG, FR, LE = self.extract_indicators(row)
p_dropout = self.dropout_cpt.table[(LG, FR, LE)]
return np.array([1 - p_dropout, p_dropout])
```

All of our models are handwritten, not library based.

Evaluating The Models





All generated evaluation files can be found in our provided the project's reports folder

Error Analysis & Troubleshooting

Data Leakage

The biggest error we made and the biggest challenge we faced was the introduction of a data leak. We initially reinserted the "Target" label as a feature which caused:

- F1 = 1.0 for all depths
- 100% accuracy
- A single split decision tree
- Faulty results

Troubleshooting

We fixed this by dropping the original "Target" after mapping binary labels and ensuring that preprocessing is fit only on training data

Bayesian Network Performance

Our BN accuracy and F1 are lower than our kNN & Decision Tree.
This is likely due to the fact that our BN only uses three handcrafted indicators and a fixed structure. Thus, it is unable to capture all the complex interactions in the dataset.
Had we more time to expand on this project, we would implement the following:

- Add more indicator variables
- Use more refined thresholds

A full step-by-step guide on how to run our code on your machine is in our provided README.md