

# UO Predictive Analytics

## kNN Hands-on Task A: kNN classifier for adult income prediction

This notebook is an exercise for developing a kNN classifier for adult income prediction. We apply the concepts discussed in **Weeks 1-3**.

You should use the codes from **kNN Practical Part 1**.

🚩 **Note:** This assessment is unmarked. It is designed to develop your skills in predictive model development. We will release the solution for this exercise. You should check your solution against the supplied solution, and if you have any concerns, please discuss them with the teaching team.

### 1.1 The Adult Dataset

Source: [UCI Adult Dataset](#)

We are using a subset of the above dataset in this exercise. The prediction task is to determine whether a person makes over **\$50K** a year.

- 
- **Q1:** Classify this task according to the classification discussed in **Concept 1.3: Types of Predictive Problems**

**Answer:**

- **Q2:** How many samples are there in the dataset?

```
In [ ]: #write your code here
```

- **Q3:** What are the features here?

```
In [ ]: #write your code here
# Get feature names (excluding the target column 'income-level')

# Print the features
```

- **Q4:** What are the target classes? How many samples are in each target class?

```
In [ ]: # Get the unique target classes

# Count the number of samples in each target class

# Print the results
```

- **Q5:** What is the data type of each feature?

```
In [ ]: #write your code here
```

- **Q6:** Write code to divide the dataset into training and test sets. Use a 70/30 split.

```
In [ ]: #write your code here
```

- **Q7:** How many samples are there for each class in the test set?

```
In [ ]: #write your code here
```

---

Let us try to build a kNN model using this dataset. Run the code cell below.

🚩 **Note:** Make sure `X_train` and `y_train` exist before calling `.fit()`.

They should be properly formatted as **NumPy arrays** or **Pandas DataFrames**.

```
In [ ]: #write your code here
```

---

We may encounter the following error when training the `KNeighborsClassifier`:

```
ValueError: could not convert string to float: 'Private'
```

This occurs because `KNeighborsClassifier()` expects numerical **input features**, but our dataset contains **categorical features** (such as occupation, work class, and marital status), which are stored as **object** types. kNN cannot directly process categorical data.

---

## 1. First Approach: Using `LabelEncoder` to bypass the error

`LabelEncoder` from the `sklearn.preprocessing` package allows us to convert categorical features into numeric values.

🚩 **Read more about LabelEncoder:**

🔗 [Scikit-learn LabelEncoder Documentation](#)

**Example: Using LabelEncoder** Below is an example of how to apply `LabelEncoder` to categorical features:

```
In [ ]: from sklearn.preprocessing import LabelEncoder

# Creating a dataset of categorical values
dic_data = {
    1: 'paris',
    2: 'paris',
    3: 'tokyo',
    4: 'amsterdam'
}

# Convert dictionary to DataFrame
df_temp = pd.DataFrame(list(dic_data.items()), columns=['id', 'city'])

# Initialize LabelEncoder
le = LabelEncoder()

# Apply LabelEncoder to the 'country' column
df_temp['en_city'] = le.fit_transform(df_temp['city'])

# Display the updated DataFrame
print(df_temp)
```

- 
- **Q8:** Following the above example, encode the categorical features and the target.

```
In [ ]: #write your code here
```

- **Q9:** Develop a kNN classifier using the numerical and encoded features.

```
In [ ]: #write your code here
```

- **Q10:** What is the accuracy of the model?


```
In [ ]: #write your code here
```

- **Q11:** Show a graph of k values vs accuracy.

```
In [ ]: #write your code here
```

```
In [ ]: #write your code here to generate the plot
```

---

 **Reflect on your process.** Are all steps correct? Would you change anything?

---

## 1.1 Label Encoding: When to Use It?

Using `LabelEncoder` in `KNeighborsClassifier` (or any supervised learning method) depends on how your categorical data is structured.

### When NOT to use LabelEncoder

For **nominal categories** (e.g., colors: Red, Blue, Green), `LabelEncoder` assigns arbitrary numbers (Red=0, Blue=1, Green=2). Since KNN relies on distance, this falsely creates an ordinal relationship (e.g., KNN might think "Green" (2) is closer to "Blue" (1) than "Red" (0)).

- **Better alternative:**

Use **One-Hot Encoding** ( `pd.get_dummies()` or `OneHotEncoder` ) to avoid misleading distances.

### When LabelEncoder is okay

- Encoding **target labels (y)** in classification—KNN doesn't compute distances on the target.
- **Ordinal categories** (e.g., Low=0, Medium=1, High=2) where order matters.
- **binary categories** (e.g., "yes", "no"). Manual mapping (0/1) is also ok.

---

## 2. A second (better) approach: Using One-Hot Encoding

Currently, categorical variables like `occupation` are label-encoded. For example:

- **Adm-clerical** → 0
- **Exec-managerial** → 3
- **Handlers-cleaners** → 5

In Euclidean space, 0 is closer to 3 than 5, meaning that **Adm-clerical is more similar to Exec-managerial than to Handlers-cleaners**. However, in reality, there is no meaningful numerical distance between these categories.

### Distance Metric for Mixed Features

For mixed feature types, we should ideally use the distance metric discussed in **Concept 3.3 Distance Metrics**:

$$d(x^{(i)}, x^{(j)}) = \frac{\sum_{k=1}^m \delta(x_k^{(i)}, x_k^{(j)}) \times d(x_k^{(i)}, x_k^{(j)})}{\sum_{k=1}^m \delta(x_k^{(i)}, x_k^{(j)})}$$

where:

- $d(x_k^{(i)}, x_k^{(j)})$  is the distance between two feature values.
- $\delta(x_k^{(i)}, x_k^{(j)})$  is an indicator function that ensures only comparable features contribute to the distance.

Unfortunately, **scikit-learn does not support this metric** natively.

To improve distance measurement, we can apply **One-Hot Encoding** to categorical features instead of **Label Encoding**. This technique converts categorical values into **binary vectors**, preventing misleading numerical distances.

For example:

Occupation	Label Encoding	One-Hot Encoding
Adm-clerical	0	[1, 0, 0]
Exec-managerial	3	[0, 1, 0]
Handlers-cleaners	5	[0, 0, 1]

This method ensures **equal Euclidean distance** between different categories, making kNN perform better with categorical features.

**Note:** One-hot encoding ensures equal distances between categories in Euclidean, Manhattan, and Cosine spaces, though the absolute values differ depending on the metric.

 **Read more about One-Hot Encoding here:**

[StackAbuse - One-Hot Encoding in Python with Pandas and Scikit-Learn](#)

- **Q12:** Create a new df by applying one-hot encoding to the categorical features.

 **Note:**

We do not need to apply One-Hot Encoding to the **sex** variable because:

- It is **binary** (only two categories: Male & Female).
- It has **already been encoded**.
- Dummy encoding a binary variable would create two columns, which is unnecessary since one column already captures all the needed information.

Thus, we can **keep it as is** without additional transformations.

```
In [ ]: #write your code here
        # Apply One-Hot Encoding using pandas get_dummies.

        # Display the first 3 rows of the transformed dataset
        df_new.head(3)
```

- **Q13:** Add the numerical and binary features as well as the target. Get all feature column names.

```
In [ ]: #add the features and the target to df_new
```

```
In [ ]: # Get all feature column names

        # Display the updated feature list
```

- **Q14:** Use those features to build a kNN model. Compare the performance of the new model with the previous one.

🚩 **Note:** Notice the training time difference between the approaches. Time increases as

we increase number of features.

```
In [ ]: #write your code here
```

```
In [ ]: #write your code here to compare with respect to k values
```

```
In [ ]: #write your code here to generate the plot
```

### 3. A third KNN classifier: Applying label encoding to ordinal Features

Did you notice that the **education** feature has **16 different categories**. However, these levels have a **natural order**—for example, **Preschool is closer to 1st-4th grade than to a Doctorate**. Using **One-Hot Encoding** for this feature would create **16 new columns**, which increases the number of features significantly.

To **reduce complexity**, we can use the **Label Encoded** version of the feature `education`, but since education levels follow an order, we need to **manually assign numeric values** in the correct sequence.

Below is the order we propose (**feel free to suggest a different one if it makes sense**).

```
In [ ]: # Define the ordinal mapping for education levels
        education_mapping = {
            "Preschool": 0,
```

```

    "1st-4th": 1,
    "5th-6th": 2,
    "7th-8th": 3,
    "9th": 4,
    "10th": 5,
    "11th": 6,
    "12th": 7,
    "HS-grad": 8,
    "Some-college": 9,
    "Assoc-voc": 10,
    "Assoc-acdm": 11,
    "Bachelors": 12,
    "Masters": 13,
    "Prof-school": 14,
    "Doctorate": 15
}

# Apply the mapping to create an ordinally encoded education column
df["en_education"] = df["education"].map(education_mapping)

# Display the first few rows
print(df[["education", "en_education"]].head())

```

```

In [ ]: #remove all column to start with a fresh "df_new"
df_new.drop(df_new.columns, axis=1, inplace=True)

# Apply One-Hot Encoding to the categorical features
df_new = pd.get_dummies(df[['workclass','occupation']],
                        , prefix=['w', 'o'])

```

```

In [ ]: #add the remaining features and the target in df_new
df_new['en_sex'] = df['en_sex']
df_new['en_education'] = df['en_education']
df_new['age'] = df['age']
df_new['hours-per-week'] = df['hours-per-week']
df_new['income-level'] = df['income-level'] # Ensure correct column name

```

```

In [ ]: # Get all feature column names
features3 = df_new.drop(columns=['income-level']).columns

# Display the updated feature List
features3

```

- **Q15:** Build a kNN classifier. Compare the performance of the classifier with the previous one.

```

In [ ]: #write your code here

```

```

In [ ]: #write your code here to compare with respect to k values

```

```

In [ ]: #write your code here to generate the plot

```

After applying **Label Encoding**, we noticed that:

- **Performance has slightly increased**
- **Training time has improved**

🚩 **Note:** In **KNN**, distances between data points determine classification. If one feature has a **much larger range** than others, it will **dominate the distance calculation**, making smaller-scale features less impactful. Since KNN calculates distances (often using **Euclidean distance**), features with **larger values contribute more**, even if they are not more important. In our example, **Education (encoded):** Values range from **0 to 15**, while **Sex (encoded):** Values range from **0 to 1**

👉 **How to approach: Normalize the Features**

By scaling all numerical features to a **similar range** (e.g., 0 to 1 using MinMax Scaling), we ensure that:

- **No single feature dominates the distance metric.**
- **All features contribute equally** to the classification.

---

### 3. A fourth KNN classifier: Normalising Features

We observed that categorical features have much smaller values compared to one-hot encoded features.

As a result, the distance calculation in kNN is **dominated** by the differences in the numeric attributes.

To prevent this, we can **normalise** numerical features so that they do not disproportionately influence the distance calculation.

We can achieve this using the `MinMaxScaler()` method from the `sklearn.preprocessing` package.

🚩 **Read more about MinMaxScaler:**

🔗 [Scikit-learn MinMaxScaler Documentation](#)

### Example: Using MinMaxScaler

Below is an example in a small (synthetic) data of how to normalize numerical features:

```
In [ ]: from sklearn.preprocessing import MinMaxScaler

# Creating a dataset of numeric values
dic_data = {
    1: 50,
```



```

    2: 60,
    3: 5,
    4: 63
}

# Convert dictionary to DataFrame
df_temp = pd.DataFrame(list(dic_data.items()),
                        columns=['id', 'age'])

# Initialize MinMaxScaler
min_max_scaler = MinMaxScaler()

# Normalize 'age' column to [0, 1] scale
# [[ ]] ensures a 2D array input
df_temp['n_age'] = min_max_scaler.fit_transform(df_temp[['age']])

# Display the normalized DataFrame
print(df_temp)

```

- **Q16:** Normalise the features in our dataset that require it.

```
In [ ]: #write your code here to normalise the features.
```

```
In [ ]: # Get all feature column names that we are using for the classifier

# Display the updated feature list
```

```
In [ ]: # write your code here for KNN
```

```
In [ ]: #write your code here to compare with respect to k values
```

```
In [ ]: #write your code here to generate the plot
```

## Conclusion & Next Steps

### Key Takeaways:

1. We built and tested multiple **KNN classifiers** using different encoding techniques.
2. **One-Hot Encoding** is better for nominal features, while **Label Encoding** can be used for ordinal ones (with proper normalization).
3. **Feature scaling** was performed to ensure fair distance calculations in KNN.
4. Our best model achieved **81% accuracy**, showing that our preprocessing choices impact performance.

### Next Steps for Improvement:

- **Try different distance metrics** (e.g., Manhattan, Hamming) to see if they improve classification.

- **Tune hyperparameters** like `n_neighbors` for better accuracy.
- **Experiment with feature selection** to remove less important features and improve efficiency.
- **Further data cleaning** may be required as our feature engineering suggest that some data is missing or irrelevant (e.g., registers encoded as `'w_?'` )
- **Compare with other classifiers** (e.g., Decision Trees, SVM) to evaluate whether KNN is the best choice for this dataset.