Introduction to Machine Learning in Python – Workshop 5

Presented by Jiajia Li

21.08.2024

Australian National University

Agenda

Recap – Data preprocessing

Last week, we have talked about data preprocessing. Proper preprocessing can significantly improve the performance of our machine learning models.

1. Data collection
    - Gather data
    - Merge datasets
2. Data cleaning
    - Handle missing values
    - Handle duplicates
    - Handle outliers
3. Data transformation
    - Scaling/Normalisation
    - Encoding categorical variables
    - Feature engineering
4. Feature selection
    - Remove irrelevant features
    - Select important features
5. Handling imbalanced data
    - Resampling techniques
    - Class weight adjustment
6. Splitting the data
    - Train-test split
    - Cross-validation

Handle missing values

Here are several common techniques to handle missing data in a dataset:

1. **Remove missing data**: drop columns/rows

2. **Imputation**:
   - Fill with a specific value, such as 0.
   - Fill with Mean/Median/Mode
   - Forward Fill/Backward Fill: use the previous or next value to fill, useful for time series data.

3. **Advanced imputation techniques**:
   - K-Nearest Neighbours (KNN) imputation
   - Multiple imputation

Handle missing values

4. **Flag and Fill**: create a new binary column that flags whether data was missing, then fill missing values with an imputed value.

5. **Using machine learning models for imputation**: use a machine learning model to predict missing values based on other features.

6. **Leave missing data as is**, some machine learning models (e.g., XGBoost, LightGBM) can handle missing data natively.

How to choose which method to use?

- Experiment with different methods to see what works best for your data.
- Consider why the data is missing (missing completely at random, missing at random, missing not at random) and choose an appropriate method.

Handle duplicates

Identifying duplicates:

- Find duplicate rows:

```
[4]: housing.duplicated().sum()

[4]: 0
```

- Find rows that are duplicates based on specific columns:

```
[7]: housing.duplicated(subset=['Rooms', 'Bedroom2']).sum()

[7]: 13537
```

Our data doesn’t have duplicates.

Handle outliers

Identifying outliers:

Visual methods:

- Box plot
- Scatter plot

Statistical methods:

- **Z-score**: the Z-score measures how many standard deviations a data point is from the mean. Typically, data points with a Z-score greater than 3 or less than -3 are considered outliers.
- **Interquartile Range (IQR):** the IQR is the range between the first quartile (25th percentile) and the third quartile (75th percentile). Data points outside the range defined by `Q1 - 1.5 * IQR` and `Q3 + 1.5 * IQR` are considered outliers.

Handle outliers

- Remove outliers.
- Impute outliers
  - Replace with **mean/median**.
  - **Cap/Floor** outliers: set a cap or floor on outliers, limiting their values to the upper and lower bounds.
- Use robust models: **tree-based models** (e.g., Random Forest, Gradient Boosting), are more robust to outliers.

Data Transformation - Normalisation

Data normalisation is a preprocessing technique used to scale numerical features in a dataset to a common range, typically [0,1] or [-1,1]. This is especially important when different features have different ranges or units, as it ensures that each feature contributes equally to the model's performance.

Min-Max Scaling: scales the data to a fixed range, usually [0,1].

$$X_{\text{norm}} = \frac{X - X_{\text{min}}}{X_{\text{max}} - X_{\text{min}}}$$

Z-Score Normalisation: scales the data based on the mean and standard deviation, resulting in a distribution with a mean of 0 and a standard deviation of 1.

$$X_{\text{norm}} = \frac{X - \mu}{\sigma}$$

μ is the mean.
σ is the standard deviation.

```
from sklearn.preprocessing import StandardScaler
```

Data Transformation - Normalisation

Max Absolute Scaling: scales each feature by its maximum absolute value. This method preserves the sign of the data and scales it within the range [-1,1].

```
from sklearn.preprocessing import MaxAbsScaler
```

Robust Scaling: uses the median and interquartile range (IQR) for scaling, making it more robust to outliers.

$$X_{\text{scaled}} = \frac{X - \text{median}}{\text{IQR}}$$

```
from sklearn.preprocessing import RobustScaler
```

Logarithmic Scaling: can be used to reduce the impact of large outliers and skewed distributions.

```
df['feature1_log'] = np.log1p(df['feature1'])
```

Data Transformation - Normalisation

Choosing the right normalisation technique:

- **Min-Max Scaling**: Use when you want the data to have a specific range, e.g., [0, 1]. Commonly used in deep learning.
- **Z-Score Normalization**: Use when you want to standardize features with different units or scales to have a mean of 0 and a standard deviation of 1.
- **Max Absolute Scaling**: Useful when data has a range with both positive and negative values, and you want to preserve the sign.
- **Robust Scaling**: Use when your data contains outliers, and you want to scale without being influenced by them.
- **Logarithmic Scaling**: Useful for skewed data to reduce the impact of outliers.

Data Transformation – Encoding categorical variables

Label Encoding: Assigns a unique integer to each category. This method is suitable when the categorical variable has an ordinal relationship (e.g., "Low", "Medium", "High").

Using scikit-learn:

```
from sklearn.preprocessing import LabelEncoder

le = LabelEncoder()
df['encoded_column'] = le.fit_transform(df['categorical_column'])
```

Manually:

```
df['encoded_column'] = df['categorical_column'].astype('category').cat.codes
```

Consideration: Label Encoding should be used carefully, as it can introduce unintended ordinal relationships between categories.

Data Transformation – Encoding categorical variables

One-Hot Encoding: converts each category into a new binary column (0 or 1). This method is suitable when there is no ordinal relationship between the categories.

Using Pandas `get_dummies()`:

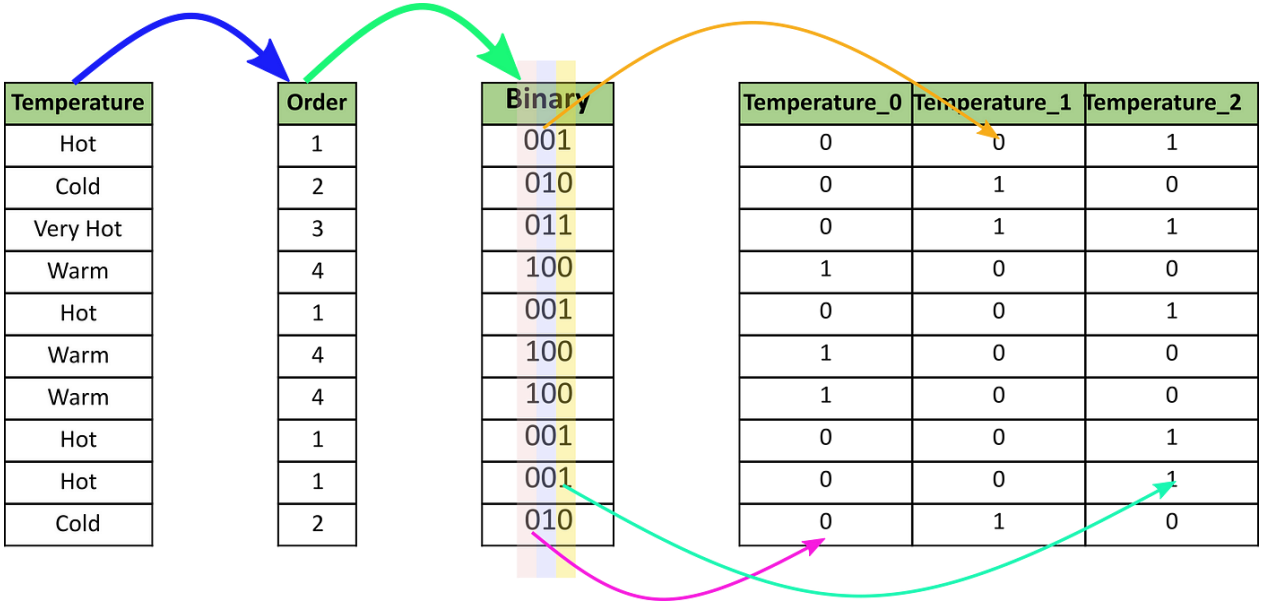
```
pd.get_dummies(housing, columns=['Regionname'])
```

Regionname_Eastern Metropolitan	Regionname_Eastern Victoria	Regionname_Northern Metropolitan	Regionname_Northern Victoria
False	False	True	False
False	False	True	False
False	False	True	False
False	False	True	False
False	False	True	False

Consideration: One-Hot Encoding can lead to a high-dimensional feature space, especially when the categorical variable has many unique values.

Data Transformation – Encoding categorical variables

Binary Encoding combines the benefits of Label Encoding and One-Hot Encoding. Categories are first converted to integers, then to binary, and each binary digit becomes a new column.



Temperature
Hot
Cold
Very Hot
Warm
Hot
Warm
Warm
Hot
Hot
Cold

Order
1
2
3
4
1
4
4
1
1
2

Binary
001
010
011
100
001
100
100
001
001
010

Temperature_0	Temperature_1	Temperature_2
0	0	1
0	1	0
0	1	1
1	0	0
0	0	1
1	0	0
1	0	0
0	0	1
0	0	1
0	1	0

Data Transformation – Encoding categorical variables

Using the `category_encoders` library:

```
import category_encoders as ce

encoder = ce.BinaryEncoder(cols=['categorical_column'])
df_encoded = encoder.fit_transform(df)
```

Consideration: Binary Encoding reduces the dimensionality compared to One-Hot Encoding, making it useful for high-cardinality features.

Data Transformation – Encoding categorical variables

Target Encoding replaces each category with the mean of the target variable. This method is often used in supervised learning where you want to capture the relationship between a categorical feature and the target.

Using the `category_encoders` library:

```
import category_encoders as ce

encoder = ce.TargetEncoder(cols=['Regionname'])
housing['Region_enc'] = encoder.fit_transform(housing['Regionname'], housing['Price'])
```

Consideration: Target Encoding can lead to data leakage if not done properly, especially when encoding on the training set and then applying the transformation on the test set.

Data Transformation – Encoding categorical variables

Frequency Encoding replaces each category with the frequency (or count) of that category in the dataset.

```
housing['Region_enc'] = housing['Regionname'].map(housing['Regionname'].value_counts())
```

Consideration: Frequency Encoding preserves the count information of the categories, which can be useful for some models.

Data Transformation – Encoding categorical variables

Choosing the Right Encoding Technique:

- **Label Encoding**: Use for ordinal categorical variables.
- **One-Hot Encoding**: Use for nominal categorical variables with a small number of categories.
- **Binary Encoding**: Useful for high-cardinality categorical variables.
- **Target Encoding**: Use when you have a strong relationship between the categorical variable and the target variable but be careful of overfitting.
- **Frequency Encoding**: Use when the frequency of categories carries important information.

Feature Selection

Feature selection is the process of identifying and selecting the most relevant features in your dataset that contribute to the predictive power of a machine learning model. This step is crucial for improving model performance, reducing overfitting, and speeding up the training process.

Below are several methods for feature selection:

- Filter Methods
- Wrapper Methods
- Embedded Methods

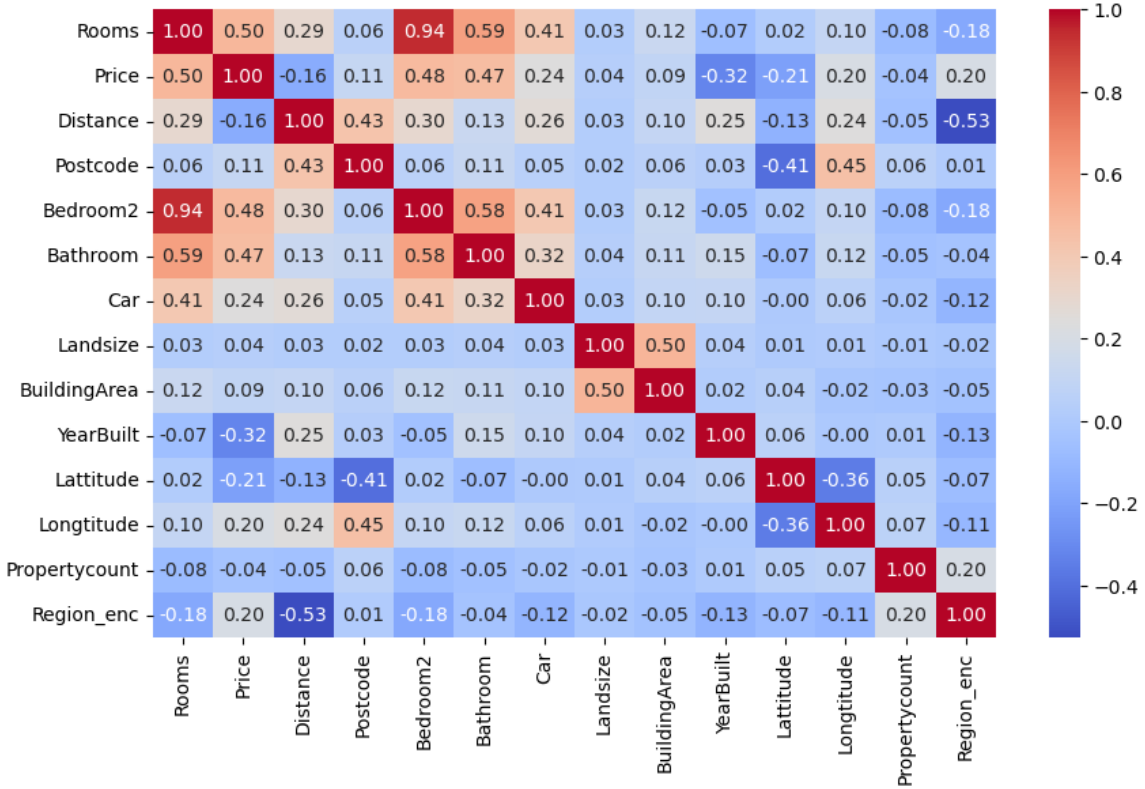
Feature Selection – Filter Methods

Filter methods rely on the statistical properties of the data to select features independently of the model.

Correlation Coefficient: select features that have a high correlation with the target variable but low correlation with each other. This method only works for numeric data.

```
import seaborn as sns
import matplotlib.pyplot as plt

corr_matrix = housing.corr(numeric_only=True)

plt.figure(figsize=(10,6))
sns.heatmap(corr_matrix, annot=True, fmt='.2f', cmap='coolwarm')
plt.show()

high_corr_features = corr_matrix.index[abs(corr_matrix['Price']) > 0.5]
```

Feature Selection – Correlation Matrix



Feature Selection – Chi-Square Test

Chi-Square Test: used between **two categorical features or more** to measure the dependence between features and the target variable.

Assess the relationship between ‘gender’ and ‘actualhospitalmortality’ for eICU dataset.

```
import pandas as pd
from sklearn.feature_selection import chi2
from sklearn.preprocessing import LabelEncoder

df = pd.read_csv('./eicu_cohort.csv')

le = LabelEncoder()

x = le.fit_transform(df['gender']).reshape(-1,1)
y = le.fit_transform(df['actualhospitalmortality'])

chi2_stat, p_values = chi2(x, y)

print(f'The Chi-Square statistic is {chi2_stat[0]}')
print(f'The p-value is {p_values[0]}')
```

Feature Selection – Chi-Square Test

```
The Chi-Square statistic is 1.8334401709401695
The p-value is 0.17572174951176397
```

Chi-Square Statistic: This value measures how much the observed frequencies deviate from the expected frequencies under the null hypothesis (i.e., assuming the variables are independent). A **higher** Chi-Square statistic indicates a **greater deviation**, suggesting a stronger association between the variables.

P-Value: The p-value tells you the probability that the observed association (or something more extreme) would occur if the null hypothesis were true.

- Small p-value (< 0.05): Reject the null hypothesis, indicating a significant association between the variables.
- Large p-value (≥ 0.05): Fail to reject the null hypothesis, indicating no significant association between the variables.

Feature Selection – ANOVA

ANOVA (Analysis of Variance) can be used as a feature selection method in machine learning, particularly when dealing with **categorical independent variables** and **a continuous dependent variable**.

The goal is to determine whether the means of the dependent variable differ significantly across the levels of a categorical feature.

If a feature has a significant **F-value** (indicating a significant difference in means across levels), it can be considered important for predicting the target variable.

Let's examine the relationships between independent variables "Regionname", "Suburb" and dependent variable "Price".

Feature Selection – ANOVA

```python
import pandas as pd
from sklearn.preprocessing import LabelEncoder
from sklearn.feature_selection import f_classif

le = LabelEncoder()

housing['Regionname'] = le.fit_transform(housing['Regionname'])
housing['Suburb'] = le.fit_transform(housing['Suburb'])

X = housing[['Regionname', 'Suburb']]
y = housing['Price']

f_values, p_values = f_classif(X, y)

for i, (f_val, p_val) in enumerate(zip(f_values, p_values)):
    print(f"Feature '{X.columns[i]}': F-Value = {f_val:.2f}, p-value = {p_val:.2e}")

significant_features = X.columns[p_values < 0.05]
print(f"Significant features: {list(significant_features)}")
```

Feature Selection – ANOVA

If p-value < 0.05:

- There is a significant difference between the means of at least two groups.
- The F-value indicates how strong this difference is.

If p-value >= 0.05:

- There is no significant difference between the group means, meaning that any observed differences are likely due to random chance rather than a true effect.

```
Feature 'Regionname': F-Value = 1.10, p-value = 1.60e-03
Feature 'Suburb': F-Value = 1.12, p-value = 2.45e-04
Significant features: ['Regionname', 'Suburb']
```

Feature Selection – Wrapper Methods

Wrapper methods evaluate the model performance by selecting a subset of features and training the model iteratively.

Recursive Feature Elimination (RFE): recursively removes features and builds a model on the remaining features to identify the most significant ones.

Feature Selection – RFE

```python
import numpy as np
from sklearn.feature_selection import RFE
from sklearn.linear_model import LinearRegression

numeric_df = housing.select_dtypes(include=['number'])
x = numeric_df.drop('Price', axis=1)
y = numeric_df['Price']

x = x.fillna(x.median())

model = LinearRegression()
rfe = RFE(model, n_features_to_select=5)
fit = rfe.fit(x, y)

ranking = rfe.ranking_
selected_features = x.columns[rfe.support_]

print("Feature Ranking: ", ranking)
print("Selected Features: ", selected_features)
```

```
Feature Ranking:  [1 2 5 3 1 1 7 6 4 1 1 8]
Selected Features:  Index(['Rooms', 'Bathroom', 'Car', 'Lattitude', 'Longtitude'], dtype='object')
```

Feature Selection – Sequential Feature Selection

Sequential feature selection adds or removes features one by one, evaluating the model's performance at each step.

```
from sklearn.feature_selection import SequentialFeatureSelector

model = LinearRegression()

sfs_forward = SequentialFeatureSelector(model, n_features_to_select=5, direction='forward')
sfs_forward = sfs_forward.fit(x, y)

sfs_backward = SequentialFeatureSelector(model, n_features_to_select=5, direction='backward')
sfs_backward = sfs_backward.fit(x, y)

selected_features_forward = x.columns[sfs_forward.get_support()]
selected_features_backward = x.columns[sfs_backward.get_support()]

print("Selected Features (Forward):", selected_features_forward)
print("Selected Features (Backward):", selected_features_backward)
```

```
Selected Features (Forward): Index(['Rooms', 'Distance', 'Bathroom', 'YearBuilt', 'Lattitude'], dtype='object')
Selected Features (Backward): Index(['Rooms', 'Distance', 'Bathroom', 'YearBuilt', 'Lattitude'], dtype='object')
```

Feature Selection – Embedded Methods

Embedded methods perform feature selection during the model training process and are specific to the learning algorithm.

Lasso Regression (L1 Regularisation): Lasso performs feature selection by penalising the coefficients of the least important features, effectively shrinking them to zero.

Standardising the data is important because Lasso is sensitive to the scale of the features. We can use `StandardScaler` from scikit-learn for this purpose.

Choosing Alpha: The value of alpha is crucial. A very small alpha may not sufficiently shrink the coefficients, while a very large alpha might shrink too many coefficients to zero. **Cross-validation** (e.g., using LassoCV) can help you find the optimal alpha.

Feature Selection – Lasso Regression

```python
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LassoCV

scaler = StandardScaler()
x_scaled = scaler.fit_transform(x)

lasso_cv = LassoCV(cv=10, random_state=42)
lasso_cv.fit(x_scaled, y)

optimal_alpha = lasso_cv.alpha_
print(f"Optimal alpha: {optimal_alpha}")

lasso_coefficients = pd.Series(lasso_cv.coef_, index=x.columns)

print("Lasso Coefficients with Optimal Alpha:")
print(lasso_coefficients)
```

```
Optimal alpha: 13743.45106690631
Lasso Coefficients with Optimal Alpha:
Rooms            223841.860820
Distance        -227317.237984
Postcode         54514.587318
Bedroom2         31274.906280
Bathroom        136962.404205
Car              51479.036420
Landsize          1728.575184
BuildingArea     10784.892467
YearBuilt       -111556.615781
Lattitude        -95323.189277
Longtitude       67343.349754
Propertycount       -0.000000
dtype: float64
```

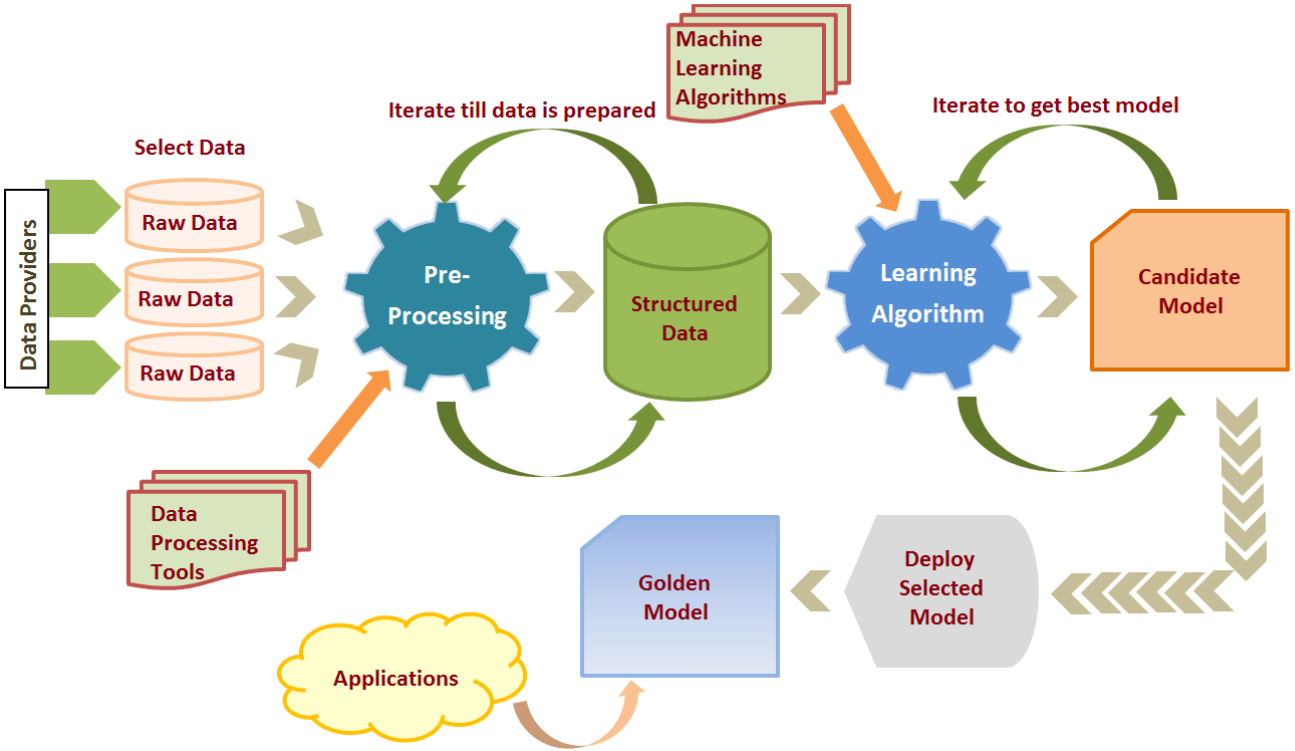
Feature Selection – Tree-Based Methods

Tree-based models like Random Forest, Gradient Boosting, and XGBoost naturally rank features by their importance.

```python
from sklearn.ensemble import RandomForestRegressor

model = RandomForestRegressor()
model.fit(x_scaled, y)

importances = model.feature_importances_

feature_importance_df = pd.DataFrame({
    'Feature': x.columns,
    'Importance': importances
}).sort_values(by='Importance', ascending=False)

print("Feature Importances:")
print(feature_importance_df)
```

```
Feature Importances:
          Feature  Importance
1           Rooms    0.228379
14     Region_enc    0.201550
2        Distance    0.143347
7        Landsize    0.116515
11     Longtitude    0.061104
8    BuildingArea    0.059556
10      Lattitude    0.050452
9       YearBuilt    0.030172
5        Bathroom    0.022212
13  Propertycount    0.019211
12     Regionname    0.018002
0           Suburb    0.016670
6              Car    0.014439
3         Postcode    0.013544
4         Bedroom2    0.004847
```

Machine Learning Process



Algorithm Choice

- **Start Simple**: Begin with simpler models (e.g., Linear Regression, Decision Trees) before moving to more complex ones (e.g., Neural Networks).

- **Benchmark Algorithms**: Compare different algorithms to choose the best performer for your specific problem.

If there is enough computational power, try a wide range of algorithms.

Best Practices

1. **Document Workflow**: Keep thorough documentation of your workflow, including data sources, feature engineering steps, and model configurations.

2. **Ensure Reproducibility**: Use version control (e.g., Git) and containerisation (e.g., Docker) to ensure that experiments can be reproduced, and results can be verified.

3. **Collaborate with Peers**: Work with colleagues to review and refine models, share insights, and improve overall performance.

4. **Follow Trends**: Stay updated with the latest advancements and best practices in machine learning through research papers, blogs (e.g., LinkedIn), and conferences.

5. **Iterative Approach**: Continuously experiment with different models, features, and techniques to find the best solution for your problem.

Q & A

Exercise

1. Try different data preprocessing techniques and benchmark algorithms.

2. Select different feature subsets and benchmark algorithms.

3. Try one-hot encoding a categorical variable and use feature selection to reduce the dimensions.

THANK YOU

Contact Us

Biological Data Science Institute
ANU College of Science

RN Robertson Building
46 Sullivans Creek Rd
Acton ACT 2601

jiajia.li1@anu.edu.au

Australian National University