Comprehensive Data Science Documentation

Prosenjit Mondol
Data Scientist
prosenjit1156@email.com

August 21, 2025

Contents

1 Data Preprocessing

- **Sampling**
  Sampling techniques are used to select a representative subset of data from a large population to reduce the computational complexity and improve the efficiency of the analysis.

- **Transformation**
  Transformation techniques involve manipulating raw data to create a single input, such as scaling, normalization, or encoding categorical data.

- **Denoising**
  Denoising techniques remove unwanted noise from the data that can lead to inaccurate results.

- **Imputation**
  Imputation techniques are used to fill in missing values in the data using statistical methods.

- **Feature extraction**
  Feature extraction techniques help to identify and extract relevant features from the data that are significant in a particular context.

- **Normalization**
  Normalization techniques are used to organize data for more efficient access and processing.

2 Handle Categorical Data

Categorical data is a type of data that represents qualitative or nominal characteristics, such as gender, occupation, Categorical data cannot be measured or compared using mathematical operations like addition or subtraction.

2.1 Different Encoding Methods for Categorical Data

- **One-Hot Encoding**
  One-Hot Encoding creates a new binary column for each category.

Listing 1: Logistic Regression Example

```
X = pd.get_dummies(X)
print(X)
```

- **Label Encoding**
  Label Encoding assigns a numerical value to each category.

```python
from sklearn.preprocessing import LabelEncoder

lencoders = {}

for col in data[features].columns:
    lencoders[col] = LabelEncoder()
    data[col] = lencoders[col].fit_transform(data[col])

data[features].nunique()
```

- **Binary Encoding**
  Binary Encoding creates new columns representing each category.

2.2 Looking at null or missing values

- **Mean Imputation**
  Mean imputation is a simple and widely used method for filling in missing values.

- **Mode Imputation**
  Mode imputation is a method for filling in missing values that is similar to mean imputation, but instead of using the mean, it uses the mode of the available values in a column.

- **K-Nearest Neighbor (KNN) Imputation**
  KNN imputation is a method for filling in missing values that is based on the distance between data poionts.

Listing 2: Logistic Regression Example

```python
# Multiple Imputation by Chained Equations
from sklearn.experimental import
    enable_iterative_imputer
from sklearn.impute import IterativeImputer

#mputed_data = df[numerical_columns].copy(deep=
    True)
mice_imputer = IterativeImputer()
data[numerical_columns] = mice_imputer.
    fit_transform(data[numerical_columns])
```

3 Checking imblanced in target variable

- **Handling imbalanced data using oversampling**
  oversampling is a method for handling imbalanced data by increasing the size of the minority class.

Listing 3: Logistic Regression Example

```
from sklearn.utils import resample

no = normalized_data[normalized_data.RainTomorrow ==
    0]
yes = normalized_data[normalized_data.RainTomorrow ==
    1]
yes_oversampled = resample(yes, replace=True,
    n_samples=len(no), random_state=123)
oversampled_data = pd.concat([no, yes_oversampled])

fig = plt.figure(figsize = (8,5))
sns.countplot(x='RainTomorrow', data =
    oversampled_data, palette = "Set1").set(title='
    RainTomorrow Indicator No(0) and Yes(1) after
    Oversampling (Balanced Dataset)')
```

- **How multicollinearity affects decision trees**
  Multicollinearity affects decision trees by reducing the importance and accuracy of the input features.

Listing 4: Logistic Regression Example

```
    #the heat map of the correlation
    plt.figure(figsize=(16,10))
    sns.heatmap(X.corr(), annot=True, cmap='RdYlGn')
```

4 Outliner Detection

- **Boxplot Method**
  One of the simplest and most popular methods for detecting outliers is the box-plot.

Listing 5: Logistic Regression Example

```
        plt.figure(figsize=(50,25))
        sns.boxplot(data=scaled_data[
            numerical_features])
```

- **Z-Score Method**
  The Z-Score method is a simple and widely used method for detecting outliers.

Listing 6: Logistic Regression Example

```
from scipy import stats
import numpy as np

# Calculate Z-scores for each value in the numerical
    features
z_scores = np.abs(stats.zscore(scaled_data[
    numerical_features]))

# Identify outliers (e.g., Z-score > 3)
outliers = (z_scores > 3)

# Print rows with outliers
print(scaled_data[outliers.any(axis=1)])
```

- **Transformation**
  Transformation involves transforming the data to a different scale to reduce the impact of the outliers.

Listing 7: Logistic Regression Example

```
        from sklearn.preprocessing import
            PowerTransformer
        # Apply Power Transformation to the numerical
             features
        power_transformer = PowerTransformer()
        scaled_data[numerical_features] =
            power_transformer.fit_transform(
            scaled_data[numerical_features])
```

5 Let's see how it fared in prediction using Logistic Regression

Listing 8: Logistic Regression Example

```
# Train a logistic regression model on the training set
from sklearn.metrics import accuracy_score
from sklearn.linear_model import LogisticRegression

# Instantiate the model
logreg = LogisticRegression(solver='liblinear',
    random_state=0)

# Fit the model
logreg.fit(X_train, y_train)

# Predict on the test set
y_pred_test = logreg.predict(X_test)

print('Model accuracy score: {0:0.4f}'.format(
    accuracy_score(y_test, y_pred_test)))
```

6 Algorithms

6.1 Simple Linear Regression

The output is shown in the best fit line.

$$
\begin{aligned}
y &= mx + C \\
h_0(x) &= \theta_0 + \theta_1 x \\
h_0(x) &= \hat{y} \quad \text{(predicted value)} \\
error &= y - \hat{y}
\end{aligned}
$$

Here, θ_0 is the intercept, θ_1 is the slope or cofficient.
if $x = 0$, then $h_0(x) = \theta_0$ (intercept).

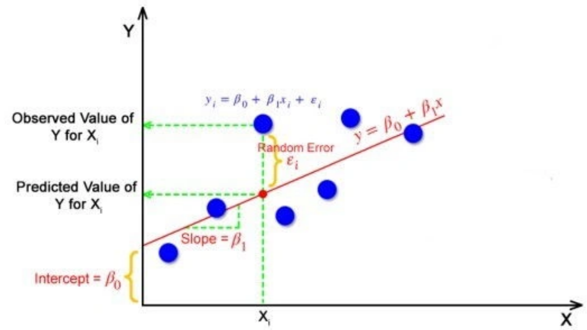

Figure 1: Simple Linear Regression: Best Fit Line, Intercept, Slope, and Error

6.2 Convergence Algorithm *(Optimize the changes of* θ_1 *values)*

Repeat until convergence:

$$\theta_j = \theta_j - \alpha \cdot \frac{\partial J(\theta_j)}{\partial \theta_j}$$

7 Handle missing values

why it not

8 Convolution Neural Networks (CNN)

Convolutional Neural Network (CNN) is an advanced version of artificial neural networks (ANNs), primarily designed to extract features from grid-like matrix datasets. This is particularly useful for visual datasets such as images or videos, where data patterns play a crucial role. CNNs are widely used in computer vision applications due to their effectiveness in processing visual data.

Listing 9: Applying CNN to an Image

```
# import the necessary libraries
import numpy as np
import tensorflow as tf
import matplotlib.pyplot as plt
from itertools import product

# set the param
plt.rc('figure', autolayout=True)
plt.rc('image', cmap='magma')

# define the kernel
kernel = tf.constant([[-1, -1, -1],
                      [-1,  8, -1],
                      [-1, -1, -1],
                     ])

# load the image
image = tf.io.read_file('Ganesh.png')
image = tf.io.decode_jpeg(image, channels=1)
image = tf.image.resize(image, size=[300, 300])

# plot the image
img = tf.squeeze(image).numpy()
plt.figure(figsize=(5, 5))
plt.imshow(img, cmap='gray')
plt.axis('off')
plt.title('Original Gray Scale image')
plt.show();


# Reformat
image = tf.image.convert_image_dtype(image, dtype=tf.
    float32)
image = tf.expand_dims(image, axis=0)
kernel = tf.reshape(kernel, [*kernel.shape, 1, 1])
kernel = tf.cast(kernel, dtype=tf.float32)
```

```python
# convolution layer
conv_fn = tf.nn.conv2d

image_filter = conv_fn(
    input=image,
    filters=kernel,
    strides=1, # or (1, 1)
    padding='SAME',
)

plt.figure(figsize=(15, 5))

# Plot the convolved image
plt.subplot(1, 3, 1)

plt.imshow(
    tf.squeeze(image_filter)
)
plt.axis('off')
plt.title('Convolution')

# activation layer
relu_fn = tf.nn.relu
# Image detection
image_detect = relu_fn(image_filter)

plt.subplot(1, 3, 2)
plt.imshow(
    # Reformat for plotting
    tf.squeeze(image_detect)
)

plt.axis('off')
plt.title('Activation')

# Pooling layer
pool = tf.nn.pool
image_condense = pool(input=image_detect,
                      window_shape=(2, 2),
                      pooling_type='MAX',
                      strides=(2, 2),
                      padding='SAME',
                     )

plt.subplot(1, 3, 3)
```

```
plt.imshow(tf.squeeze(image_condense))
plt.axis('off')
plt.title('Pooling')
plt.show()
```

9 Models

9.1 Visual Geometry Group (VGG)

Definition:
VGG (Visual Geometry Group) is a deep convolutional neural network architecture known for its effectiveness in image recognition tasks. It's characterized by its simple yet deep structure, utilizing small 3x3 convolutional filters repeatedly. VGG models, particularly VGG16 and VGG19, are widely used and have achieved notable performance on datasets like ImageNet.

9.1.1 VGG16

Definition:
VGG16 is a specific variant of the VGG architecture with 16 layers (13 convolutional layers and 3 fully connected layers). It is known for its depth and simplicity, making it effective for image classification tasks.

9.2 Ensemble Model

Definition:
An ensemble model in machine learning combines the predictions of multiple individual models (base estimators) to produce a more accurate and robust prediction than any single model alone.

9.3 SOTA Model

Definition:
In deep learning, SOTA model means State-of-the-Art model — basically, the best-performing architecture or method for a given task at a given time, according to benchmarks or competitions.

9.4 Utsu Method

Definition:
Otsu's method is a technique used in computer vision and image processing for automatic image thresholding.

10 Plotting