Introduction to K-Nearest Neighbors: Takeaways

Syntax

- Splitting a DataFrame into a training and test set:

```
train_df = banking_df.sample(frac=0.85, random_state=417)
test_df = banking_df.drop(train_df.index)
```

- Calculating the Euclidean distance between two observations with just one feature:

```
abs(X_train[feature] - test_input[feature])
```

- Calculating the accuracy of a model:

```
(X_test["predicted_y"] == y_test).value_counts(normalize=True)[0]*100
```

- Creating dummy variables:

```
pd.get_dummies(data = banking_df_copy, columns = ["marital"], drop_first = True)
```

- Calculating the Euclidean distance between two observations with multiple features:

```
distance = 0
for feature in features:
    distance += (X_train[feature] - test_input[feature])**2
    X_train["distance"] = (distance)**0.5
```

Concepts

- The **K-Nearest Neighbors** algorithm:
    - For an unseen data point, the algorithm calculates the distance between that point and all the observations across all features in the training dataset.
    - It sorts those distances in ascending order.
    - It selects **K** observations with the smallest distances from the above step. These **K** observations are the K-nearest neighbors of that unseen data point.
    - It calculates which label of those neighbors is the most common, and assigns that label to the unseen data point.
- A **distance metric** calculates the distance between two observations.
- One of the most common distance metrics is the **Euclidean distance**. For n features, this distance can be calculated as:
  $\text{distance} = \sqrt{(x_1 - y_1)^2 + (x_2 - y_2)^2 ... + (x_n - y_n)^2}$
  Where, for each i in $\{1, \ldots, n\}$,
    - x_i is the value for a feature for one observation, and
    - y_i is the value for the same feature for another observation.

- K-nearest neighbors does not technically have a "training phase". The model classifies every new input by comparing it to its neighbors. Those neighbors are data points from the training set.
- **Accuracy** of a model can be calculated as the percentage of correct predictions it makes out of all predictions.
- **Feature Engineering** is the process of transforming features so that they can be effectively used to train models and yield better performance. For example:
    - **One-hot encoding** encodes categorical columns as numerical values.
    - **Min-max Scaling** or **Min-max Normalization** scales the values of a feature into the range `[0, 1]`.
- Formula for min-max scaling:
  $x' = \frac{x - \min(x)}{\max(x) - \min(x)}$
  Where x is the original value of the feature.

Resources

- Bank Marketing Dataset
- pandas' shape() function
- pandas' sample() function
- pandas' drop() function
- pandas' nsmallest() function
- pandas' mode() function
- pandas' get_dummies() function