Evaluating Model Performance: Takeaways

Syntax

- Splitting a dataset into a training and validation set:

```
X_train, X_val, y_train, y_val = train_test_split(X, y, test_size=0.20, random_state = 417)
```

- Instantiating a k-NN Classifier:

```
knn = KNeighborsClassifier(n_neighbors = 3)
```

- Fitting the classifier to the training data:

```
knn.fit(X_train, y_train)
```

- Creating dummy variables:

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

- Normalizing the data:

```
scaler = MinMaxScaler()
X_train_scaled = scaler.fit_transform(X_train[["marital_married", "marital_single",
"marital_unknown", "default_unknown", "age", "duration"]])
```

- Evaluating the model and calculating its accuracy score:

```
knn.score(X_val_scaled, y_val)
```

Concepts

- A **validation set** is a data set that can be used to evaluate a trained machine learning model.
- In scikit-learn, the `sklearn.preprocessing.MinMaxScaler.fit()` method of the `KNeighborsClassifier` sets up the data structure that allows for the model to efficiently identify the nearest neighbors.
- Both the validation and the test sets should be transformed or processed similarly to the way the training set was transformed in order to evaluate the model.
- A model **overfits** when it starts to memorize parts of the data instead of generalizing it.
- A model **underfits** when it struggles to accurately represent the data.

Resources

- Bank Marketing Dataset
- scikit-learn's train_test_split() function
- scikit-learn's KNeighborsClassifier
- pandas' get_dummies() function
- scikit-learn's MinMaxScaler