**Supervised Learning with scikit-learn**

**Classification**

scikit-learn offers a repeatable workflow for using supervised learning models to predict the target variable values when presented with new data.

* Learn how to solve classification problems using supervised learning techniques
* Split data into training and test sets
* Fit a model
* Make predictions
* Evaluate accuracy
* Examine the relationship between model complexity and performance

There are two types of supervised learning: **Classification** versus **Regression**

A feature is also called a predictor variable or independent variable

A target variable is also called a dependent variable or response variable

Requirements prior to using supervised learning:

* No missing values
* Data in numeric format
* Data store in pandas DataFrame or Numpy array

Use Exploratory Data Analysis (EDA) to ensure that the above requirements are met.

The scikit-learn framework and syntax:

1. From sklearn.module import Model
   1. k-Nearest Neighbors (kNN)
2. model = Model()
3. model.fit(X, y)
   1. Model learns from the labeled data we pass to it in this step.
4. predictions = model.predict(X\_new)
   1. Pass unlabeled data to the model as input and use the “fitted” model to predict the labels

Using scikit-learn implement kNN

from sklearn.neighbors import KNeighborsClassifier

X = churn\_df[[“total\_day\_charge”, “total\_eve\_charge”]].values

y = churn\_df[“churn”].values

knn = KNeighborsClassifier(n\_neighbors=15)

knn.fit(X, y)

*knn is referred to as a* ***classifier***

X\_new = np.array([[56.8, 17.5], [24.4, 24.1], [50.1, 10.9]])

predictions = knn.predict(X\_new)

print(‘Predictions: {}.format(predictions))

Now that a kNN classifier is fit, the next step is to measure the classifier’s performance. There are various performance measures for classifiers:

* Accuracy = (correct predictions) / (total obesrvations)

from sklearn.model\_selection import train\_test\_split

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.3, random\_state=21, stratify=y)

knn.KNeighborsClassifier(n\_neighbors=6)

knn.fit(X\_train, y\_train)

print(knn.score((X\_test, y\_test))

In kNN classifiers in is noteworthy that:

* Larger k 🡪 less complex models 🡪 can cause underfitting
* Smaller k 🡪 more complex models 🡪 can lead to overfitting

Diving deeper into model complexity and over/underfitting:

train\_accuracies = {}

test\_accuracies = {}

neighbors = np.arange(1, 26)

for neighbor in neighbors:

knn = KNeighborsClassifier(n\_neighbors=neighbor)

knn.fit(X\_train, y\_train)

train\_accuracies[neighbor] = knn.score(X\_train, y\_train)

teest\_accuracies[neighbor] = knn.score(X\_test, y\_test)

plt.figure(figsize=(8, 6))

plt.title(“KNN: Varying Number of Neighbors”)

plt.plot(neighbors, train\_accuracies.values(), label=”Training Accuracy”)

plt.plot(neighbors, test\_accuracies.values(), label=”Testing Accuracy”)

plt.legend()

plt.xlabel(“Number of Neighbors”)

plt.ylabel(“Accuracy”)

plt.show()

Interpreting model complexity is a great way to evaluate supervised learning performance. Your aim is to produce a model that can interpret the relationship between features and the target variable, as well as generalize well when exposed to new observations.

**Regression**

The target variable has continuous values.

X = diabetes\_df.drop(“glucose”, axis=1).values

Y = diabetes\_df[“glucose”].values

X\_bmi = X[:, 3]

X\_bmi = X\_bmi.reshape(-1, 1)

print(X\_bmi.shape, y.shape)

import matplotlib.pyplot as plt

plt.scatter(X\_bmi, y, color=”blue”)

plt.ylabel(“Blood Glucose (mg/dl)”)

plt.xlabel(“Body Mass Index”)

plt.show()

from sklearn.linear\_model import LinearRegression

reg = LinearRegression()

reg.fit(X\_bmi, y)

predictions = reg.predict(X\_bmi)

plt.scatter(X\_bmi, y)

plt.plot(X\_bmi, preditions)

plt.ylabel(“Blood Glucose (mg/dl)”)

plt.xlabel(“Body Mass Index”)

plt.show()

**Fine-Tuning Your Model**

**Preprocessing and Pipelines**