21.1 Lesson Summary - Introduction to Machine Learning

Machine Learning processes data with algorithms and uses the results to identify patterns, predict future behavior, enhance computing processes, and more.

Concept: **Intelligent algorithms** utilize data to alter their behavior. Three main kinds of intelligent algorithms are Machine Learning, Predictive Analysis, and Artificial Intelligence. **Machine Learning** is the ability of software to learn and improve from experience. **Predictive Analysis** is the ability of software to predict future outcomes based on historical data. **Artificial Intelligence** is the simulation of human intelligence by machines.

Machine Learning can be achieved through both Unsupervised and Supervised learning. **Unsupervised Learning** allows the software to discover patterns in data without human input. **Supervised Learning** requires human input to describe the data.

**Classification** is an application of supervised machine learning where data is classified by a set of human provided labels. **Regression** is a technique of supervised machine learning that generates a model to describe existing data and predict new data. **Clustering** is an unsupervised learning technique where data is organized into categories based on machine generated labels.

The Machine Learning paradigm includes three steps, Model, Fit and Predict. **Models** are generated to describe the data. Data is used to **fit** or train the models. The trained models are then tested in their ability to **predict** future data or data unused in the fitting process.

Concept: A **Simple Linear Regression** describes how to derive a dependent variable given an independent variable by using a linear equation **y = mx + b** where **y** is the dependent variable, **x** is the independent variable, **m** is the slope of the line and **b** is the **y** intercept. Linear Regressions are fast to perform and are a good test before investigating more complex models.

The Sklearn Python library offers a LinearRegression object to derive a linear equation from your data. If you have a list of x and y coordinates you could get the linear equation for the data using the following code:

*from sklearn.linear\_model import LinearRegression*

*model = LinearRegression()*

*model.fit(x\_coordinates, y\_coordinates)*

*print(f"y = {model.coef\_}x + {model.intercept\_}")*

Once you have created and fit (or trained) your model you can use it to predict y values by passing in a list of x values. For example:

*model.predict(X)*

* Activity: 01-Ins\_Univariate\_Linear\_Regression\_Sklearn, 02-Stu\_LSD\_Regression

Concept: How predictive a model is can be quantified by the **Mean Squared Error** (**MSE**) and **R Squared** (**R2**) values. A good MSE score will be close to zero and a good R2 Score will be close to 1. If you have a series of *X* and *y* values you can find the MSE and R2 using the following code:

*from sklearn.linear\_model import LinearRegression*

*from sklearn.metrics import mean\_squared\_error, r2\_score*

*# Create a linear model*

*model = LinearRegression()*

*# Fit (Train) our model to the data*

*model.fit(X, y)*

*# Use our model to predict a value*

*predicted = model.predict(X)*

*# Score the prediction with mse and r2*

*mse = mean\_squared\_error(y, predicted)*

*r2 = r2\_score(y, predicted)*

*print(f"Mean Squared Error (MSE): {mse}")*

*print(f"R-squared (R2 ): {r2}")*

R2 score is the default for many Sklearn models including LinearRegression. The following code also provides the R2 value for *X* and *y* data points:

*model.score(X,y)*

Another way to test your model is by breaking up your data set into data to train your model and data to test the model after it has been trained. To do this you can use the train\_test\_split function from the Sklearn library. If you started with arrays of data *X* and *y* you could use the following code to get the R2 value of your model:

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

*X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, random\_state=10)*

*model.fit(X\_train, y\_train)*

*model.score(X\_test, y\_test)*

* Activity: 03-Ins\_Quantifying\_Regression, 04-Stu\_Brains\_Regression

Concept: Linear regressions can use **multiple input features**. Input features can be thought of as dimensions on a graph that influence a y value. A two-dimensional plot has one feature, x that influences y. A three-dimensional plot has 2 features, x and y, that influence z. If you have multiple features in your *X* array you can utilize the same code to generate a linear regression model and calculate it's R2 score:

*from sklearn.linear\_model import LinearRegression*

*model = LinearRegression()*

*model.fit(X, y)*

*score = model.score(X, y)*

*print(f"R2 Score: {score}")*

It is difficult to visualize data in N dimensional space so you can use a **Residual** plot to visualize the veracity of a model's prediction. For example:

*predictions = model.predict(X)*

*plt.scatter(predictions, predictions - y)*

*plt.hlines(y=0, xmin=predictions.min(), xmax=predictions.max())*

*plt.show()*

* Activity: 05-Ins\_Multiple\_Linear\_Regression\_Sklearn, 06-Stu\_Beer\_Foam\_MultipleRegression
* Suppl link: ??????????

Concept: Most machine learning algorithms rely on numerical data. Text and categorical data must be converted to numeric values in order to function. If a row in a dataset is the member of a category this should be indicated by having a column of 1's and 0's where 1 indicates that the row is a member of the column's category. Pandas' *get\_dummies* method will create these algorithm ready columns for you. The *get\_dummies* method can be applied to an entire DataFrame to automatically identify categorical data or it can be applied to one column, for example:

*binary\_encoded\_data\_frame = pd.get\_dummies(my\_data\_frame, columns=["original\_category\_column"])*

Machine learning algorithms often perform best when non-categorical numerical values are normalized. Normalization shifts all of the numbers in a dataset to the same scale. To normalize a dataset you can use sklearn's StandardScalar function. If we wanted to apply the StandardScalar function to a dataset of *X* and *y* values we could use the following code:

*from sklearn.preprocessing import StandardScaler*

*X\_scaler = StandardScaler().fit(X)*

*y\_scaler = StandardScaler().fit(y)*

*X\_scaled = X\_scaler.transform(X)*

*y\_scaled = y\_scaler.transform(y)*

* Activity: 07-Ins\_Data\_Preprocessing, 08-Stu\_Respiratory\_Preprocessing