Understanding Basic of Machine Learning

Linear Regression
Logistic Regression

scikit-learn, also known as sklearn

- open-source machine learning library for Python
- It provides a wide range of machine learning algorithms and tools for data preprocessing, model selection, evaluation, and visualization
  - linear regression, logistic regression, decision trees, random forests, support vector machines, and k-nearest neighbors.
- provides tools for data preprocessing, such as scaling, normalization, and feature selection.

Linear regression

- popular machine learning algorithm used for predicting a **continuous outcome variable** (dependent variable) based on one or more predictor variables (independent variables).
- It assumes a linear relationship between the **dependent variable and the independent variables**, and tries to fit a straight line that best represents the relationship between them.
- The most common type of linear regression is simple linear regression, which involves only one independent variable. The equation for a simple linear regression is:
- $y = b0 + b1*x + e$
- where y is the dependent variable, x is the independent variable, b0 is the y-intercept, b1 is the slope of the line, and e is the error term.
- The goal of linear regression is to estimate the values of the parameters b0 and b1 that best fit the data, such that the sum of the squared errors (SSE) between the predicted values and the actual values is minimized.
- Linear regression can be used for a variety of applications, such as predicting the price of a house based on its size and location, or predicting the amount of rainfall based on the temperature and humidity.
- However, it is important to note that linear regression assumes a linear relationship between the dependent and independent variables, and may not be appropriate for non-linear relationships.

Example of how linear regression

- Suppose we want to predict the weight of a person based on their height. We have a dataset that contains the height (in inches) and weight (in pounds) of 10 people:
- We can use linear regression to predict the weight of a person based on their height. In this case, the height is the independent variable (x), and the weight is the dependent variable (y).
- We can fit a simple linear regression model to this data using scikit-learn:

Height (in)	Weight (lbs)
60	110
62	120
64	130
66	140
68	150
70	160
72	170
74	180
76	190
78	200

Lab 6-1

```
import numpy as np
from sklearn.linear_model import LinearRegression

# Create numpy arrays for height and weight
X = np.array([60, 62, 64, 66, 68, 70, 72, 74, 76, 78]).reshape(-1, 1)
y = np.array([110, 120, 130, 140, 150, 160, 170, 180, 190, 200])

# Create a Linear Regression model
model = LinearRegression()

# Fit the model to the data
model.fit(X, y)

# Predict the weight of a person who is 68 inches tall
height = np.array([[68]])
weight = model.predict(height)

print(f"A person who is 68 inches tall is predicted to weight
{weight[0]:.2f} pounds.")
```

Logistic regression

- machine learning algorithm used for **binary classification** tasks, where the goal is to predict the probability that an input belongs to a particular class (usually labeled as 0 or 1). It is a type of **generalized linear model that uses a logistic function** (also known as the **sigmoid function**) to transform the output of a linear regression model into a probability value between 0 and 1.
- The logistic function is defined as: $p = 1 / (1 + exp(-z))$
- where p is the predicted probability, z is the output of the linear regression model (also known as the logit), and $exp()$ is the exponential function.
- The logistic regression model tries to find the values of the coefficients (parameters) that best fit the training data, by maximizing the likelihood of the observed data given the model.
- Logistic regression can be used for a variety of applications, such as
  - predicting whether a customer will buy a product based on their demographic and purchase history, or
  - predicting whether a patient will develop a disease based on their symptoms and medical history.

Lab 6-2 How logistic regression can be used:

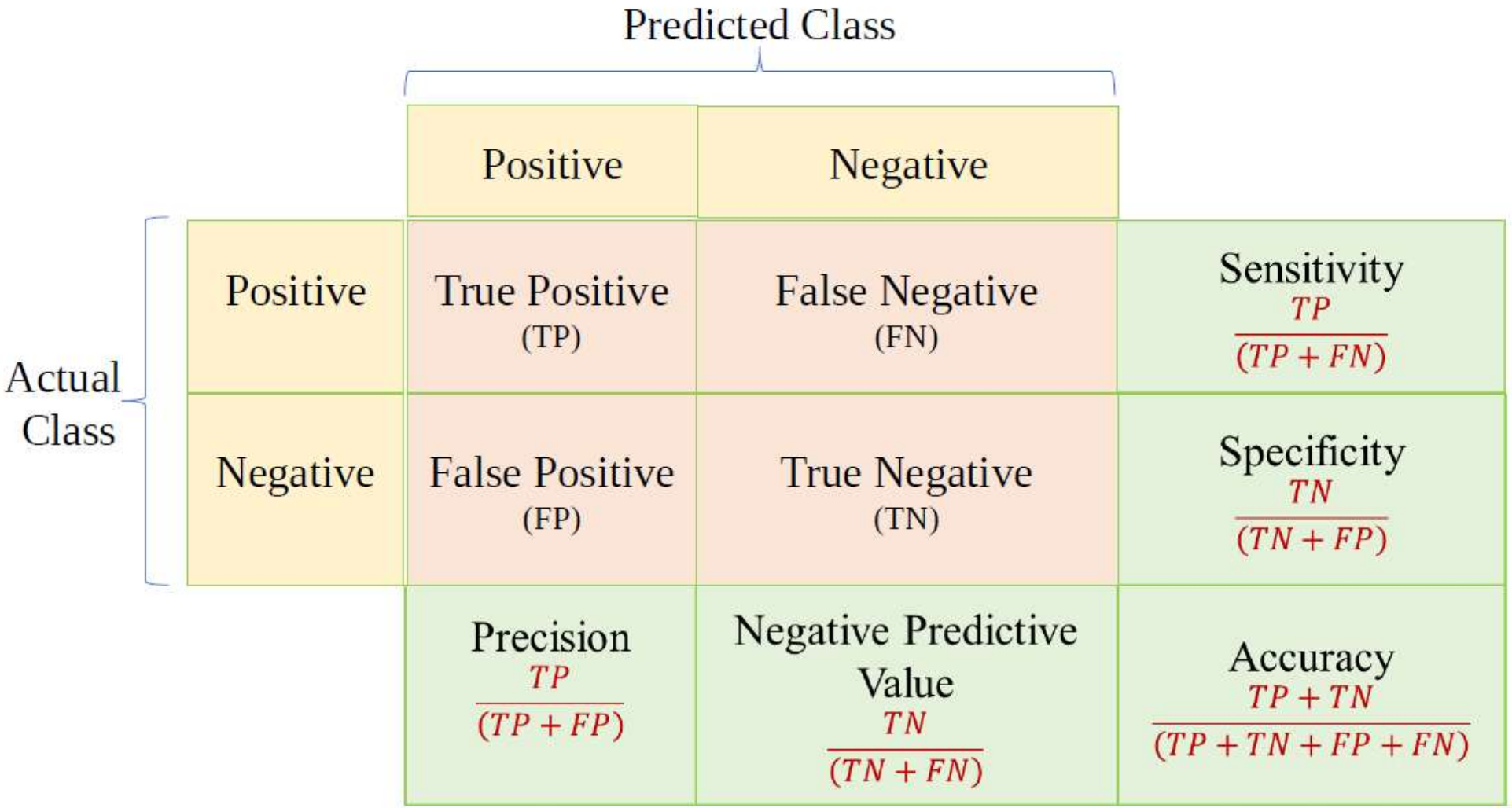
- Suppose we have a dataset that contains information about whether a person has diabetes or not, based on their age and BMI. We want to use logistic regression to predict whether a new patient with a given age and BMI has diabetes or not.

- In this example, the logistic regression model predicted that a patient who is 50 years old and has a BMI of 32 has a probability of 0.89 of having diabetes.

```
import numpy as np
from sklearn.linear_model import LogisticRegression

# Create numpy arrays for age, BMI, and diabetes status
age = np.array([25, 30, 35, 40, 45, 50, 55, 60, 65,
70])
bmi = np.array([20, 22, 25, 27, 30, 32, 35, 37, 40,
42])
diabetes = np.array([0, 0, 0, 0, 1, 1, 1, 1, 1, 1])

# Create a Logistic Regression model
model = LogisticRegression()

# Fit the model to the data
X = np.column_stack((age, bmi))
model.fit(X, diabetes)

# Predict the probability of diabetes for a patient who
is 50 years old and has a BMI of 32
patient = np.array([[50, 32]])
prob = model.predict_proba(patient)

print(f"The probability of diabetes for a patient who
is 50 years old and has a BMI of 32 is
{prob[0][1]:.2f}.")
```

Confusion Matrix usage

- to evaluate the quality of the output of a classifier
- a table that is often used to evaluate the performance of a machine learning algorithm
- used to show the **number of correctly and incorrectly classified examples**, by comparing the predicted labels with the true labels.
- displays the following four metrics:
  - **True Positive (TP):** the number of actual positives that were correctly identified by the model.
  - **False Positive (FP):** the number of actual negatives that were incorrectly identified as positives by the model.
  - **False Negative (FN):** the number of actual positives that were incorrectly identified as negatives by the model.
  - **True Negative (TN):** the number of actual negatives that were correctly identified by the model.


Predicted Class
Positive
Negative
Actual Class
Positive
True Positive (TP)
False Negative (FN)
Sensitivity $\frac{TP}{(TP+FN)}$
Negative
False Positive (FP)
True Negative (TN)
Specificity $\frac{TN}{(TN+FP)}$
Precision $\frac{TP}{(TP+FP)}$
Negative Predictive Value $\frac{TN}{(TN+FN)}$
Accuracy $\frac{TP+TN}{(TP+TN+FP+FN)}$

- Let's say we have a binary classification problem where we want to predict whether an email is spam or not. We have a dataset of 100 emails, where 60 are non-spam (ham) and 40 are spam. We train a machine learning model on this data and use it to make predictions on a test set of 20 emails. Here's an example confusion matrix based on the model's predictions:

	Actual spam (60)	Actual spam (40)
Predicted spam	12	3
Predicted spam	2	3

- In this confusion matrix:
- 12 emails were correctly predicted as ham (true negatives).
- 3 emails were incorrectly predicted as spam (false positives).
- 2 emails were incorrectly predicted as ham (false negatives).
- 3 emails were correctly predicted as spam (true positives).

Accuracy score

- Suppose we have a test set of 20 emails and our machine learning model correctly predicts 15 of them. Then, the accuracy score of the classifier can be calculated as follows:

Accuracy = (Number of correct predictions) / (Total number of predictions)

Accuracy = (True Positives + True Negatives) / (True Positives + False Positives + False Negatives + True Negatives)

Accuracy = (12 + 3) / (12 + 3 + 2 + 3) = 15 / 20 = 0.75 or 75%

- Therefore, the accuracy score of the classifier is 0.75 or 75%. This means that the model correctly classified 75% of the test set emails. However, it's important to note that accuracy can sometimes be misleading, especially if the classes are imbalanced or if certain types of errors are more important than others.

RECALL score of classifier

- Recall (also known as sensitivity or true positive rate) is a metric that measures the proportion of actual positive cases that were correctly identified by the model. In the context of our example, recall can be calculated as follows:

```
Recall = True Positives / (True Positives + False Negatives)

Recall = 3 / (3 + 2) = 0.6 or 60%
```

- Therefore, the recall score of the classifier is 0.6 or 60%. This means that the model correctly identified 60% of the actual spam emails in the test set. Recall is a useful metric when the cost of false negatives (i.e., incorrectly classifying a spam email as non-spam) is high, such as in cases where a spam email could be potentially harmful.

Precision score

- A metric that measures the proportion of predicted positive cases that are actually positive. In the context of our example, precision can be calculated as follows:

Precision = True Positives / (True Positives + False Positives)

Precision = 3 / (3 + 3) = 0.5 or 50%

- Therefore, the precision score of the classifier is 0.5 or 50%. This means that out of all the emails the model predicted as spam, only 50% were actually spam. Precision is a useful metric when the cost of false positives (i.e., incorrectly classifying a non-spam email as spam) is high, such as in cases where important non-spam emails could be mistakenly marked as spam.

F1 score

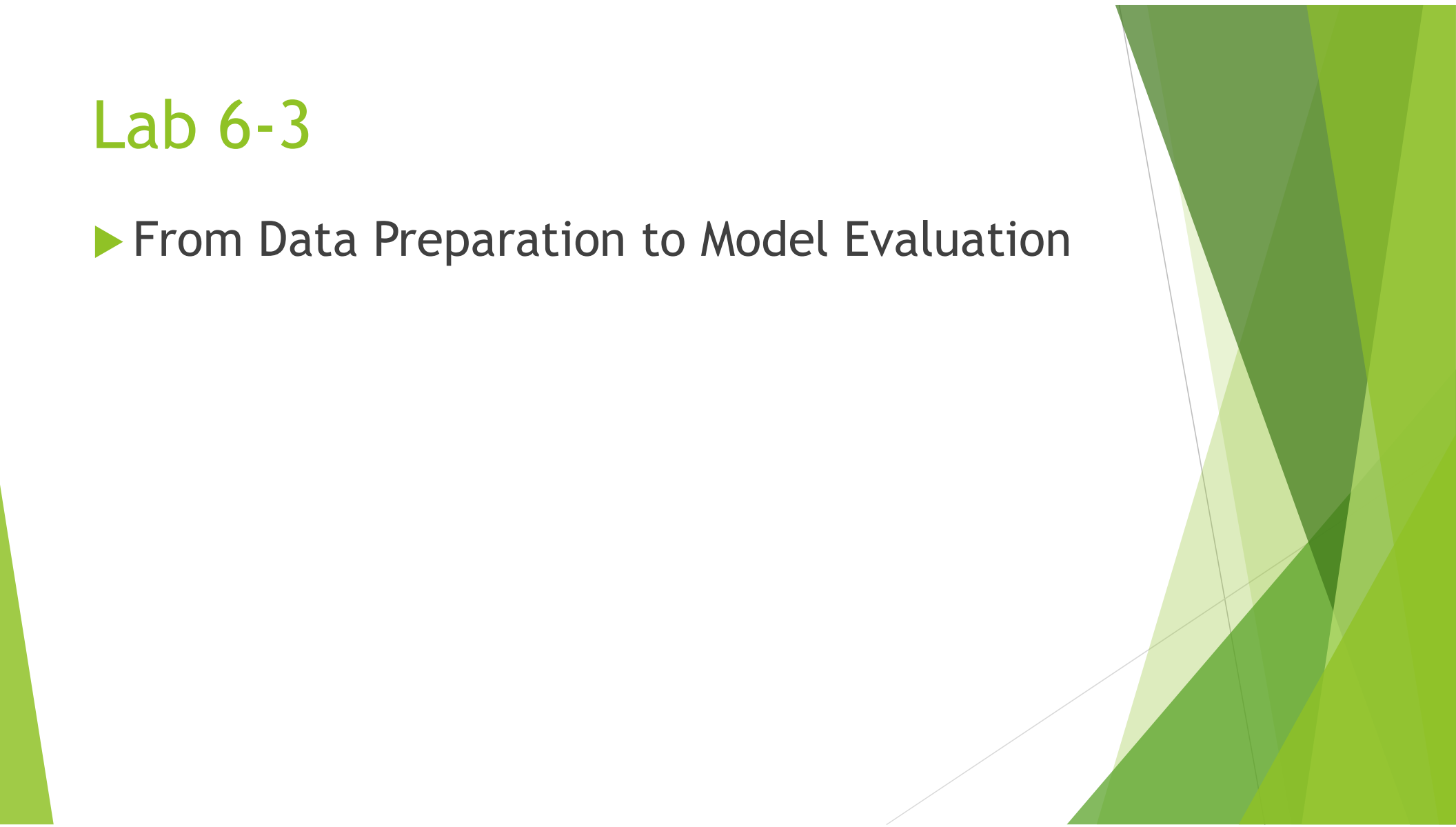
- A metric that combines precision and recall into a single score that balances both metrics. In the context of our example, F1 score can be calculated as follows:

```
Precision = True Positives / (True Positives + False Positives)
Precision = 3 / (3 + 3) = 0.5

Recall = True Positives / (True Positives + False Negatives)
Recall = 3 / (3 + 2) = 0.6

F1 Score = 2 * (Precision * Recall) / (Precision + Recall)
F1 Score = 2 * (0.5 * 0.6) / (0.5 + 0.6) = 0.545 or 54.5%
```

Therefore, the F1 score of the classifier is 0.545 or 54.5%. This means that the model achieved a balance between precision and recall, taking into account both false positives and false negatives. F1 score is a useful metric when we want to optimize both precision and recall simultaneously.

Lab 6-3

- From Data Preparation to Model Evaluation



Your turn now…

Generate random classification data using the following syntax

```
x, y = make_classification(n_samples=1000, n_classes=2, random_state=42)
```

Split data into train and test sets (70:30)

Train the data using a logistic regression model.

Predict probabilities for the test set.

Calculate the ROC curve and AUC

Plot the ROC curve.

More reading and try

- **Polynomial Regression URL:** https://sparkbyexamples.com/machine-learning/polynomial-regression-with-examples/