Sentiment Analysis & Recommender Systems - Part 2

One should look for what is and not what he thinks should be. (Albert Einstein)

Warm up

- Take a few minutes to craft a “tweet” of about 140 characters that either:
  - poses a question about something we’ve discussed
  - sums up something valuable you’ve learned
  - offers a resource you’ve discovered on your own
  - asks for additional information
- When you’re done, post your tweet in the chat so we can pick out a few to discuss in more detail

Welcome back!

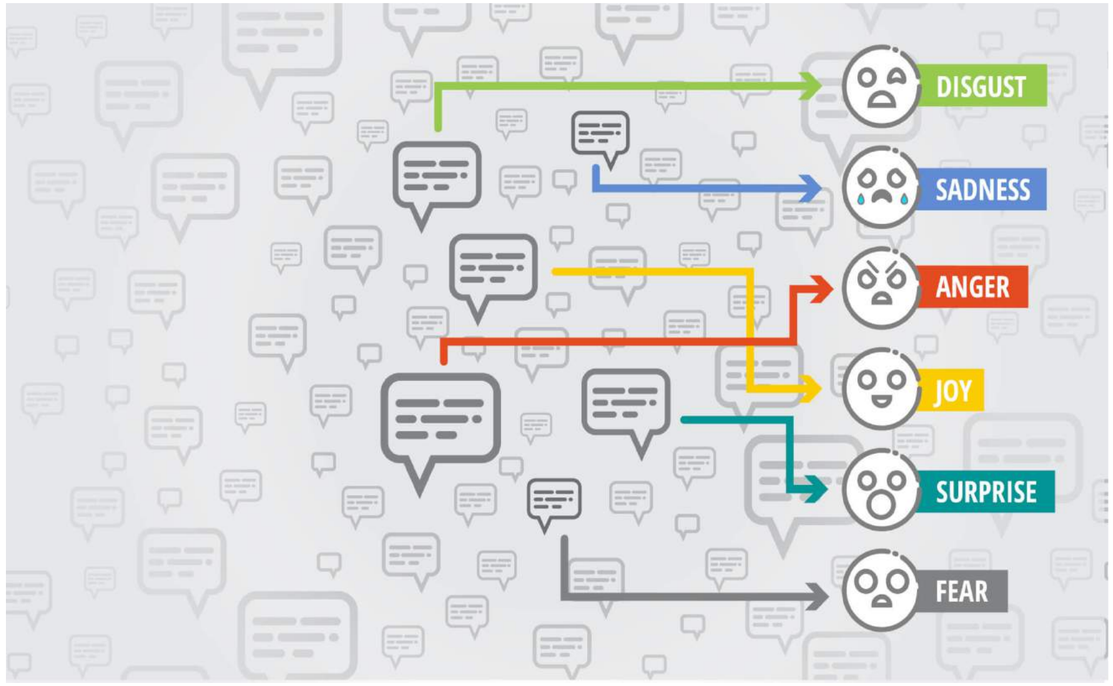
- In last class we learned about:
  - Sentiment analysis and its use cases
  - Classifying sentences and labeling them using vader package
- Today, we will will cover:
  - Logistic regression and how it will be used
  - How to initialize, build, train the logistic regression model
  - Classification performance metrics and methods to optimize the model

Sentiment analysis, everywhere!

- Sentiment mining is used in many areas:
  - opinion mining
  - reputation monitoring
  - business analytics
- It helps businesses understand their customers' experience
- **After our last class—did you happen to notice any other ways in which you may have been subject to sentiment analysis?**
- For example: Did you comment on Amazon about an order?

Sentiment analysis in Python

- **Python is very powerful when it comes to text mining, as you have seen**
- We will use the `scikit-learn` library so that we can build a model that will classify future documents
- **Our final result today will be a logistic regression model that classifies newspaper snippets as either 'negative' or 'positive'**



Module completion checklist

Objective	Complete
Split the data into train and test set for classification	
Explain the concept of logistic regression and how it will be used in this case	
Initialize, build, train the logistic regression model on our training dataset and predict on test	
Summarize classification performance metrics and methods to optimize the model	

Directory settings

- In order to maximize the efficiency of your workflow, you should encode your directory structure into `variables`
- We will use the `pathlib` library
- Let the `main_dir` be the variable corresponding to your `skillsoft-sentiment-analysis-2021` folder
- `data_dir` be the variable corresponding to your `data` folder

```
# Set 'main_dir' to location of the project folder
from pathlib import Path
home_dir = Path(".").resolve()
main_dir = home_dir.parent
```

```
data_dir = str(main_dir) + "/data"
```

- We'll be using this variable to load the data present in the data folder!

Loading packages

```
# Helper packages.
import os
import pandas as pd
import numpy as np
import pickle
import matplotlib.pyplot as plt
```

```
# Packages with tools for text processing.
import nltk
nltk.download('vader_lexicon')
```

```
# Packages for working with text data and analyzing sentiment
from nltk.sentiment.vader import SentimentIntensityAnalyzer
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.linear_model import LogisticRegression
```

```
# Packages to build and measure the performance of a logistic regression model
from sklearn.model_selection import train_test_split
from sklearn import metrics
from sklearn import preprocessing
```

Import data we saved

- Let’s now import the pre-saved pickle to continue with sentiment analysis today!

```
# Load pickled data and models.
score_labels = pickle.load(open(data_dir + "/score_labels.sav","rb"))
DTM_matrix = pickle.load(open(data_dir + '/DTM_matrix.sav',"rb"))
```

Text classification - convert DTM to array

- For our ease of use, we will convert the DTM to an an array

```
DTM_array = DTM_matrix.toarray()
# Let's look at the first few rows of the finalized array.
print(DTM_array[1:4])
```

```
[[0 0 0 ... 0 0 0]
 [0 0 0 ... 0 0 0]
 [0 0 0 ... 0 0 0]]
```

- We see a **sparse matrix with counts for each word in the sentences**
- We still have the same number of rows, but we have a column for each word that appears within any of the sentences
- **Because we have a numeric representation, we can now model the text**

Model building - split the dataset

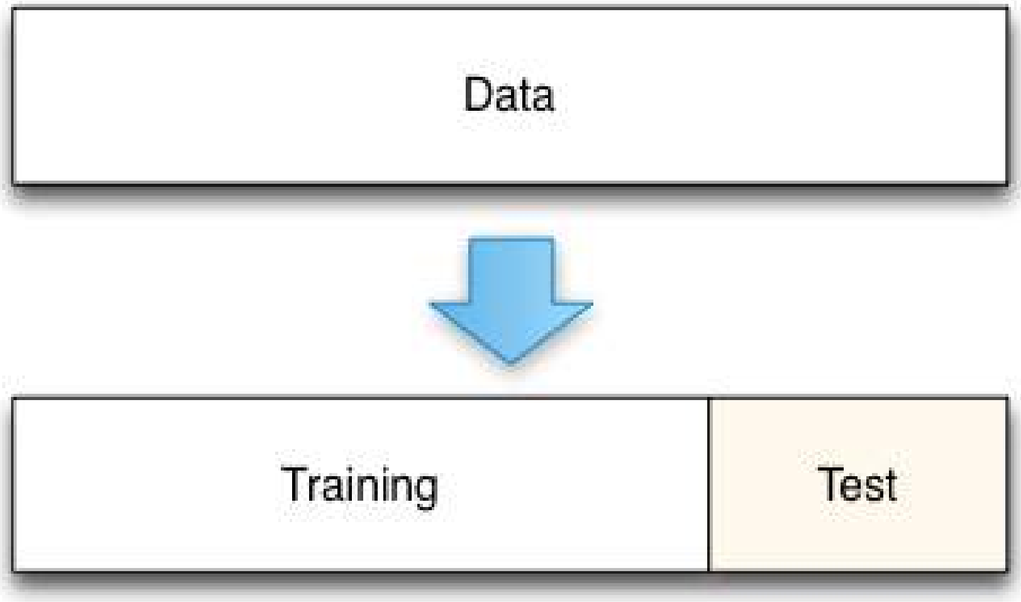
Split the DTM into train and test datasets

- We now start going into the model-building steps
- We will be using a well-known library, `scikit-learn`
- This library contains many well-known **machine learning** packages
- For documentation and more information, you can go to the ***scikit-learn homepage***

Model building - split the dataset

Split the DTM into train and test datasets

- Our next step is to get our dataset ready for the actual **model building**
- Once you have cleaned data in numeric form, you are ready to **split your data into a train and test set**
- You also need to make sure you have your **target variable**
  - In this case, the targets are the **sentiment labels we created, negative / positive**



Model building - split the dataset

Train

- This is the data that you **train your model on**
- Usually about **70% of your dataset**
- Use a larger portion of the data for training, so that the model gets a **large enough sample of the population**

Test

- This is the data that you **test your model on**
- Usually about **30% of your dataset**
- Use a smaller portion to test your trained model on

Model building - split the dataset

Split the DTM into train and test datasets

- We will use `train_test_split` from `scikit-learn` to split our dataset

  The inputs to the function are:

  i. `DTM_array` : the sparse matrix with the word counts for each 'document' (newspaper snippets in our case)
  ii. `score_labels` : the sentiment we calculated for each document, which is our target variable that we want to predict
  iii. `train_size` : the size of the training dataset, here we choose .7 or 70% of the entire dataset
  iv. `random_state`: this randomizes the split for you

```
X_train, X_test, y_train, y_test  =
train_test_split(
        DTM_array,
        score_labels,
        train_size = 0.70,
        random_state = 1234)
```

Model building - split the dataset

Split the DTM into train and test datasets

The four outputs from this function will be:

1. `X_train` : the sparse matrix split into a 70% training sample
2. `X_test` : the remaining 30% of the sample, the 'holdout' set to test the trained model on
3. `y_train` : the corresponding labels to `y_train`
4. `y_test` : the corresponding labels to `y_test`

- Make sure that these variables exist

```
print(len(X_train))
```

```
173
```

```
print(len(X_test))
```

```
75
```

```
print(len(y_train))
```

```
173
```

```
print(len(y_test))
```

```
75
```

Fun quiz

Is logistic regression a supervised machine learning algorithm?

1. YES
2. NO

Fun quiz

Suppose you have been given a fair coin and you want to find out the **probability** and the **odds** of landing on heads. Which of the following pairs of options is true for such a case?

1. **0** and **0.5**
2. **0.5** and **2**
3. **0.5** and **1**
4. None of these

Fun quiz

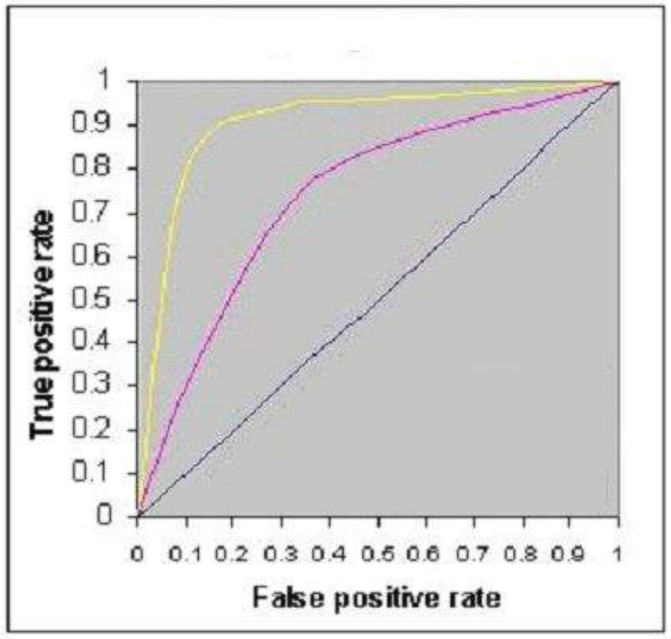
The sigmoid function plays a key role in logistic regression

1. TRUE
2. FALSE

Fun quiz

The figure shows ROC curves for three logistic regression models. Different colors show curves for different hyperparameter values. Which of the following ROC will give best result?

1. Yellow
2. Pink
3. Black
4. All are same



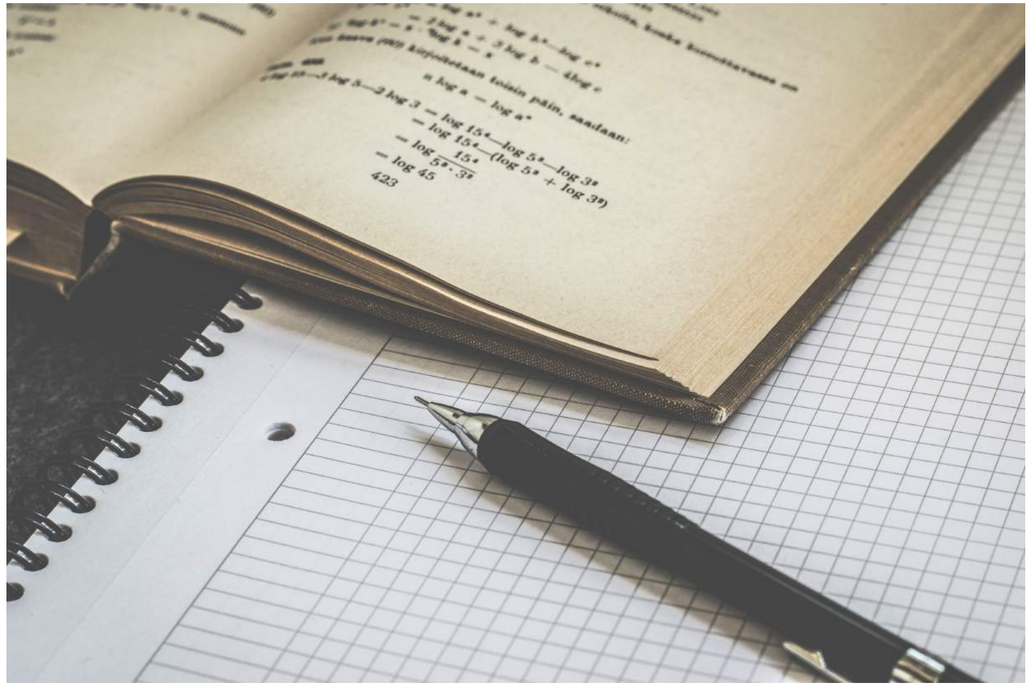
Module completion checklist

Objective	Complete
Split the data into train and test set for classification	✔
Explain the concept of logistic regression and how it will be used in this case	
Initialize, build, train the logistic regression model on our training dataset and predict on test	
Summarize classification performance metrics and methods to optimize the model	

Model building - logistic regression

Build a logistic regression model

- Let's briefly review the concept of logistic regression
  - **Supervised** machine learning method
  - Target/dependent variable is **binary** (one/zero)
  - Outputs the **probability** that an observation will be in the desired class (`y = 1`)
  - Solves for coefficients to create a *curved* function to maximize the likelihood of correct classification
  - `logistic` comes from the `logit` function (*a.k.a. inverse sigmoid function*)



Logistic regression: when to use it?

Build a logistic regression model

Since `logistic` regression is a **supervised machine learning** algorithm, we will use it to:

- **Classify** data into categories

Since `logistic` regression outputs **probabilities** and not actual class labels, it can be used to:

- Easily tweak its performance by adjusting a **cut-off probability** instead of re-running the model with new parameters

Since `logistic` regression is a **well-established algorithm** with multitudes of implementations across many programming languages, it can be used to:

- Create **robust**, **efficient** and **well-optimized models**

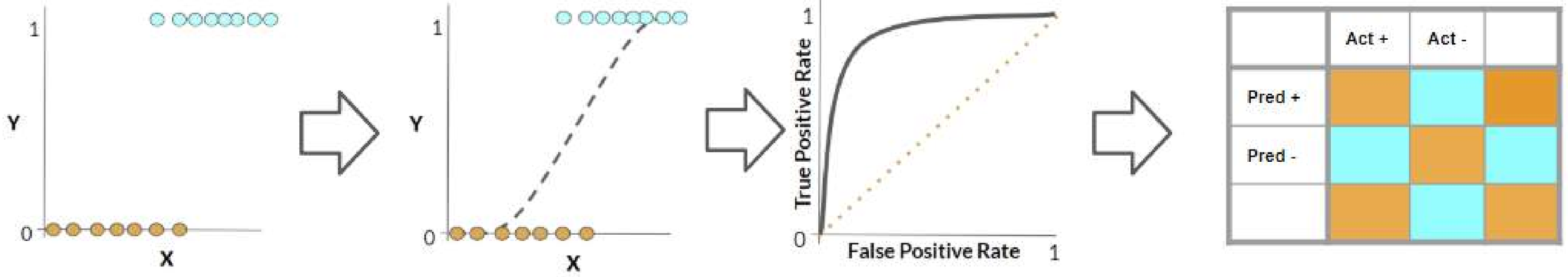
Logistic regression: process

Step 1:
Convert target variable to 1/0

Step 2:
Logistic regression on training data

Step 3:
Use ROC curve & AUC to pick threshold

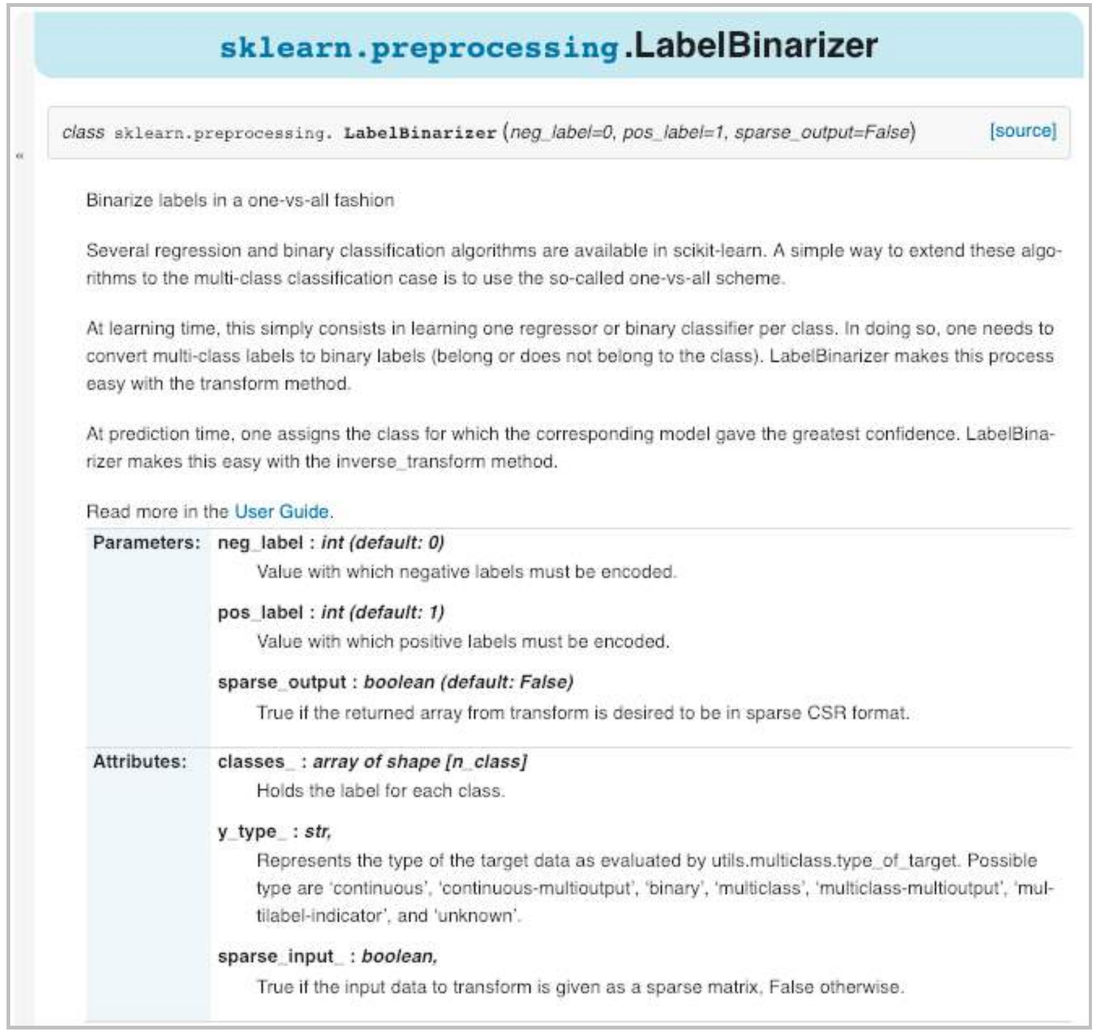
Step 4:
Check performance on test data



Categorical to binary target variable

Build a logistic regression model

- Preparing the target variable: translate an existing binary variable (i.e. any categorical variable with 2 classes) into `1` and `0`
  - we'll convert our labels to a binary variable
  - `scikit-learn` has a package `preprocessing` that can do this for you
  - `LabelBinarizer` will transform a list of strings into 1/0
  - it also works for multi-class problems - for complete documentation, click ***here***



sklearn.preprocessing.LabelBinarizer

class sklearn.preprocessing. **LabelBinarizer** (*neg_label=0, pos_label=1, sparse_output=False*) [source]

Binarize labels in a one-vs-all fashion

Several regression and binary classification algorithms are available in scikit-learn. A simple way to extend these algorithms to the multi-class classification case is to use the so-called one-vs-all scheme.

At learning time, this simply consists in learning one regressor or binary classifier per class. In doing so, one needs to convert multi-class labels to binary labels (belong or does not belong to the class). LabelBinarizer makes this process easy with the transform method.

At prediction time, one assigns the class for which the corresponding model gave the greatest confidence. LabelBinarizer makes this easy with the inverse_transform method.

Read more in the User Guide.

Parameters:

neg_label : *int (default: 0)*
Value with which negative labels must be encoded.

pos_label : *int (default: 1)*
Value with which positive labels must be encoded.

sparse_output : *boolean (default: False)*
True if the returned array from transform is desired to be in sparse CSR format.

Attributes:

classes_ : *array of shape [n_class]*
Holds the label for each class.

y_type_ : *str,*
Represents the type of the target data as evaluated by utils.multiclass.type_of_target. Possible type are 'continuous', 'continuous-multioutput', 'binary', 'multiclass', 'multiclass-multioutput', 'multilabel-indicator', and 'unknown'.

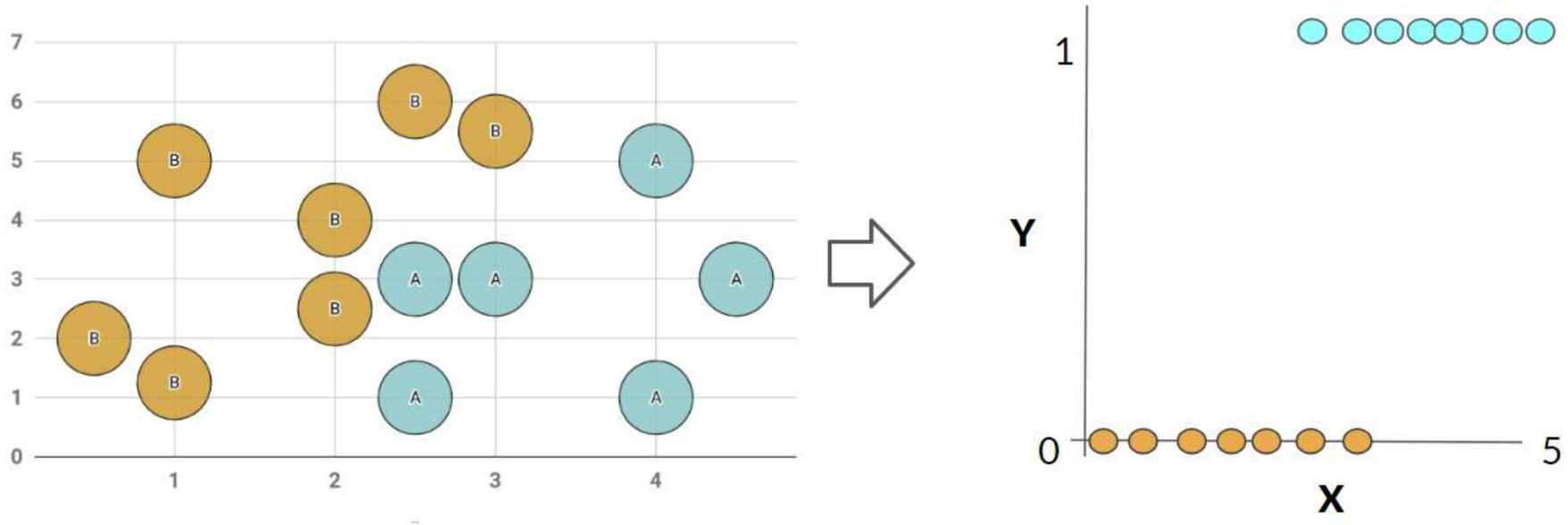
sparse_input_ : *boolean,*
True if the input data to transform is given as a sparse matrix, False otherwise.

Categorical to binary target variable

- Let's convert our labels in our `y_test` list

```
# Initiate the Label Binarizer.
lb = preprocessing.LabelBinarizer()

# Convert y_test to binary integer format.
y_test= lb.fit_transform(y_test)
```



Logistic regression: function

- For every value of x, we find p, i.e. probability of success, or probability that $y = 1$
- To solve for p, logistic regression uses an expression called a **sigmoid function:**

$$p = \frac{exp(ax + b)}{1 + exp(ax + b)}$$

- Although looking pretty involved and scary (nobody likes exponents!), we can see a very **familiar equation inside of the parentheses:** $ax + b$

Logistic regression: a bit more math

Through some algebraic transformations that are beyond the scope of this course,

$$p = \frac{exp(ax + b)}{1 + exp(ax + b)}$$

can become

$$logit(p) = log\left(\frac{p}{1 - p}\right)$$

- Since `p` is the **probability of success**, `1 - p` is the **probability of failure**
- The ratio $\left(\frac{p}{1-p}\right)$ is called the **odds** ratio, it simply tells us the **odds** of having a successful outcome with respect to the opposite
- **Why should we care?** Knowing this provides useful insight into interpreting the coefficients

Logistic regression: coefficients

$$ax + b$$

- In **linear** regression, the coefficients can easily be interpreted
- Increase in x will result in an increase in y and vice versa

BUT

- In **logistic** regression, the simplest way to interpret a positive coefficient is with an increase in likelihood
- Larger x increases the likelihood of $y = 1$

Logistic regression: connection to neural nets

- The curved *sigmoid* function is a widely used function not only in logistic regression

$$p(y = 1) = \frac{exp(ax + b)}{1 + exp(ax + b)}$$

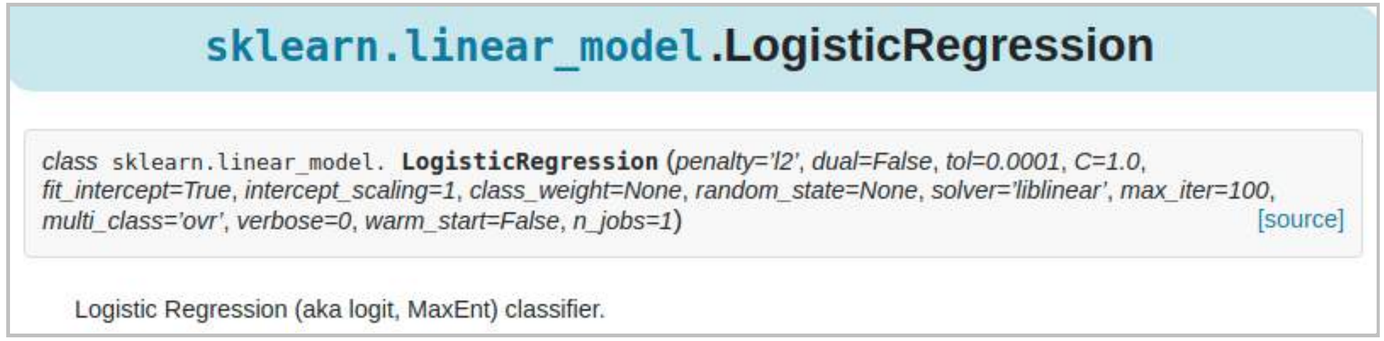
- Due to the properties and range of probabilities it produces, the sigmoid function is often used as an **activation function** in **neural networks** algorithms

Module completion checklist

Objective	Complete
Split the data into train and test set for classification	✔
Explain the concept of logistic regression and how it will be used in this case	✔
Initialize, build, train the logistic regression model on our training dataset and predict on test	
Summarize classification performance metrics and methods to optimize the model	

scikit-learn - logistic regression

- We will be using the `LogisticRegression` library from `scikit-learn.linear_model` package



sklearn.linear_model.LogisticRegression

class sklearn.linear_model. **LogisticRegression** (*penalty='l2', dual=False, tol=0.0001, C=1.0, fit_intercept=True, intercept_scaling=1, class_weight=None, random_state=None, solver='liblinear', max_iter=100, multi_class='ovr', verbose=0, warm_start=False, n_jobs=1*) [source]

Logistic Regression (aka logit, MaxEnt) classifier.

- All inputs are optional arguments
- Right now, we will build a base model
- For all the parameters of the `LogisticRegression` function, visit ***scikit-learn's documentation***

Logistic regression: build

- Let’s build our logistic regression model, we will use all default parameters for now as our baseline model

```
# Set up logistic regression model.
log_model = LogisticRegression()
print(log_model)
```

```
LogisticRegression()
```

Logistic regression: fit

The two main arguments are :

1. `X_train`: a `pandas` dataframe or a `numpy` array of train data predictors
2. `y_train`: a `pandas` series or an a `numpy` array of train labels

```
# Fit the model.
log_model = log_model.fit(X = X_train, y =
y_train)
```

fit (*X*, *y*, *sample_weight=None*) [source]

Fit the model according to the given training data.

Parameters:

X : {array-like, sparse matrix}, shape (n_samples, n_features)

Training vector, where n_samples is the number of samples and n_features is the number of features.

y : array-like, shape (n_samples,)

Target vector relative to X.

sample_weight : array-like, shape (n_samples,) optional

Array of weights that are assigned to individual samples. If not provided, then each sample is given unit weight.

New in version 0.17: sample_weight support to LogisticRegression.

Returns:

self : object

Returns self.

Logistic regression: predict

The main argument is :

1. `X_test`: a `pandas` dataframe or a `numpy` array of test data predictors

predict (*X*) [source]

Predict class labels for samples in X.

Parameters:	**X** : {array-like, sparse matrix}, shape = [n_samples, n_features] Samples.
Returns:	**C** : array, shape = [n_samples] Predicted class label per sample.

Logistic regression: predict (cont'd)

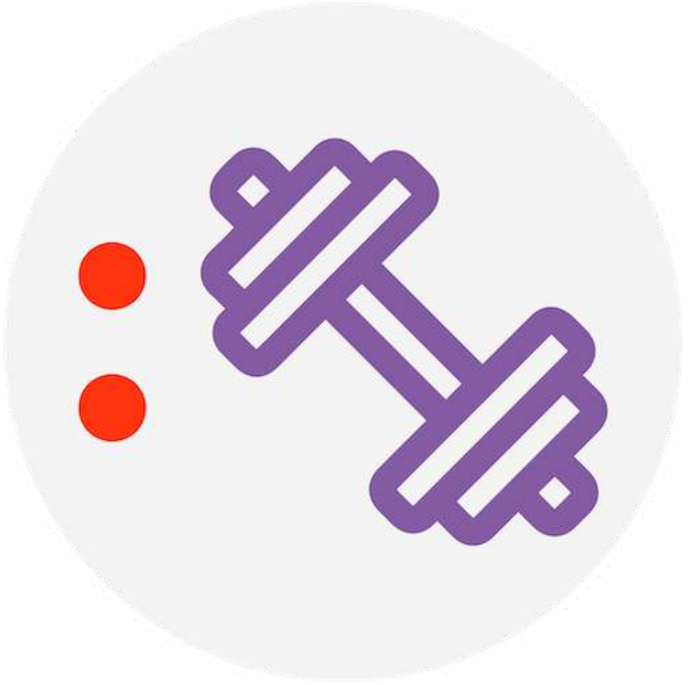
```
# Predict on test data.
y_pred = log_model.predict(X_test)
print(y_pred)
```

```
['positive' 'positive' 'positive' 'positive' 'positive' 'positive'
 'positive' 'positive' 'positive' 'negative' 'negative' 'negative'
 'positive' 'positive' 'positive' 'positive' 'positive' 'positive'
 'negative' 'positive' 'negative' 'positive' 'positive' 'positive'
 'positive' 'positive' 'positive' 'positive' 'positive' 'positive'
 'negative' 'positive' 'positive' 'positive' 'positive' 'negative'
 'positive' 'negative' 'positive' 'positive' 'positive' 'positive'
 'positive' 'positive' 'positive' 'positive' 'positive' 'positive'
 'positive' 'positive' 'positive' 'positive' 'positive' 'positive'
 'positive' 'positive' 'positive' 'positive' 'positive' 'positive'
 'negative' 'positive' 'positive' 'positive' 'positive' 'positive'
 'positive' 'positive' 'positive' 'positive' 'positive' 'positive'
 'positive' 'positive' 'positive']
```

```
# Convert y_pred to binary integer format.
y_pred= lb.fit_transform(y_pred)
```

Knowledge check 1

Exercise 1



Module completion checklist

Objective	Complete
Split the data into train and test set for classification	✔
Explain the concept of logistic regression and how it will be used in this case	✔
Initialize, build, train the logistic regression model on our training dataset and predict on test	✔
Summarize classification performance metrics and methods to optimize the model	

Model building - analyze results

Analyze results by predicting on the test set and on new data

- **We now have**:
  - model that predicts sentiment
  - predictions from the model for the test dataset
- **We want**:
  - accuracy of our model
  - methods to tune and optimize our model

To review the performance of our model, we look at:

- Confusion matrix
- ROC curve
- AUC

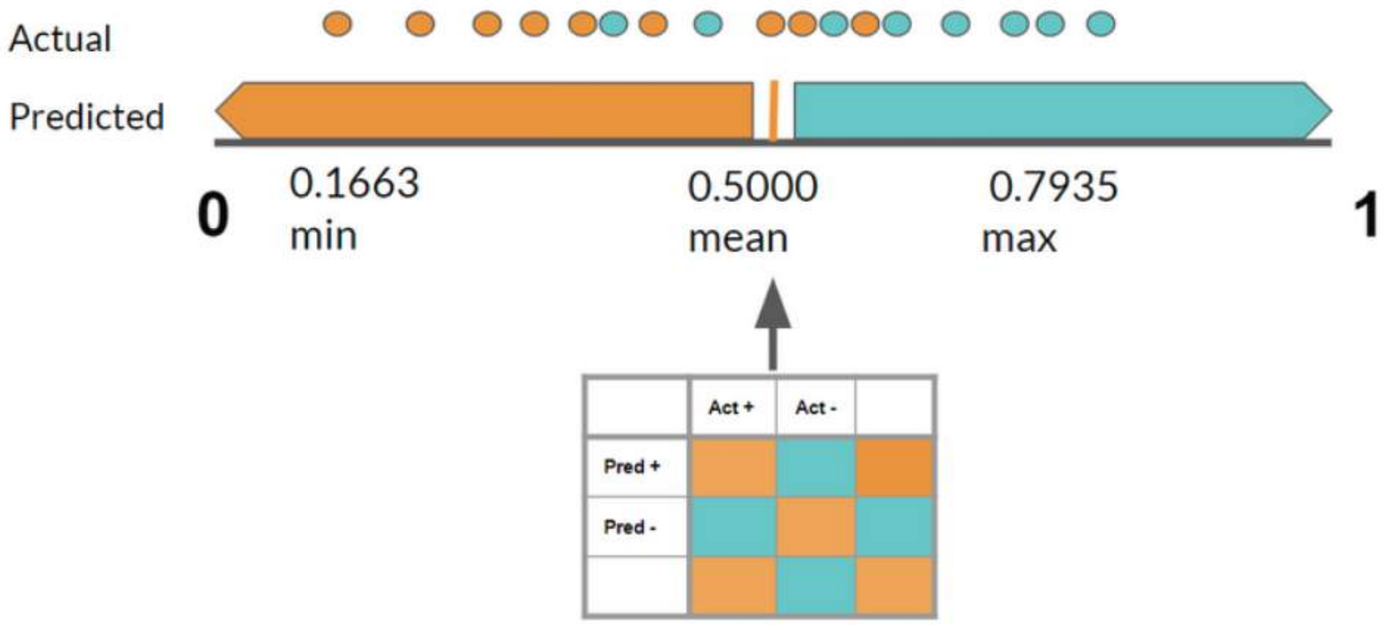
Confusion matrix

	Low yield	High yield	Predicted totals
Predicted low yield	**True positive (TP)**	**False positive (FP)**	Total predicted positive
Predicted high yield	**False negative (FN)**	**True negative (TN)**	Total predicted negative
Actual totals	Total positives	Total negatives	**Total**

- **True positive rate (TPR)** (a.k.a *Sensitivity*, *Recall*) = **TP** / Total positives
- **True negative rate (TNR)** (a.k.a *Specificity*) = **TN** / Total negatives
- **False positive rate (FPR)** (a.k.a *Fall-out*, *Type I Error*) = **FP** / Total negatives
- **False negative rate (FNR)** (a.k.a *Type II Error*) = **FN** / Total positives
- **Accuracy** = **TP + TN / Total**
- **Misclassification rate** = **FP + FN / Total**

From threshold to metrics

- In logistic regression, the output is a range of probabilities from `0` to `1`
- But how do you interpret that as a `1`/`0` or `Positive`/`Negative` label?
- You set a **threshold** where everything above is predicted as `1` and everything below is predicted `0`
- A typical threshold for logistic regression is `0.5`



From metrics to a point

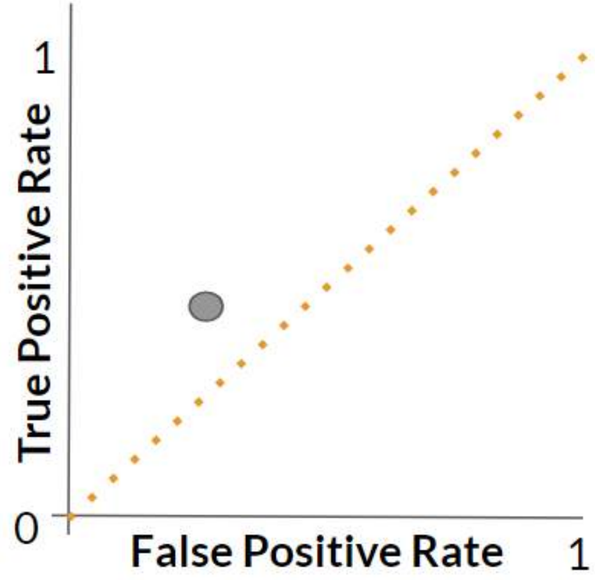
Each threshold can create a confusion matrix, which can be used to calculate a point in space defined by:

- **True positive rate (TPR)** on the `y-axis`
- **False positive rate (FPR)** on the `x-axis`

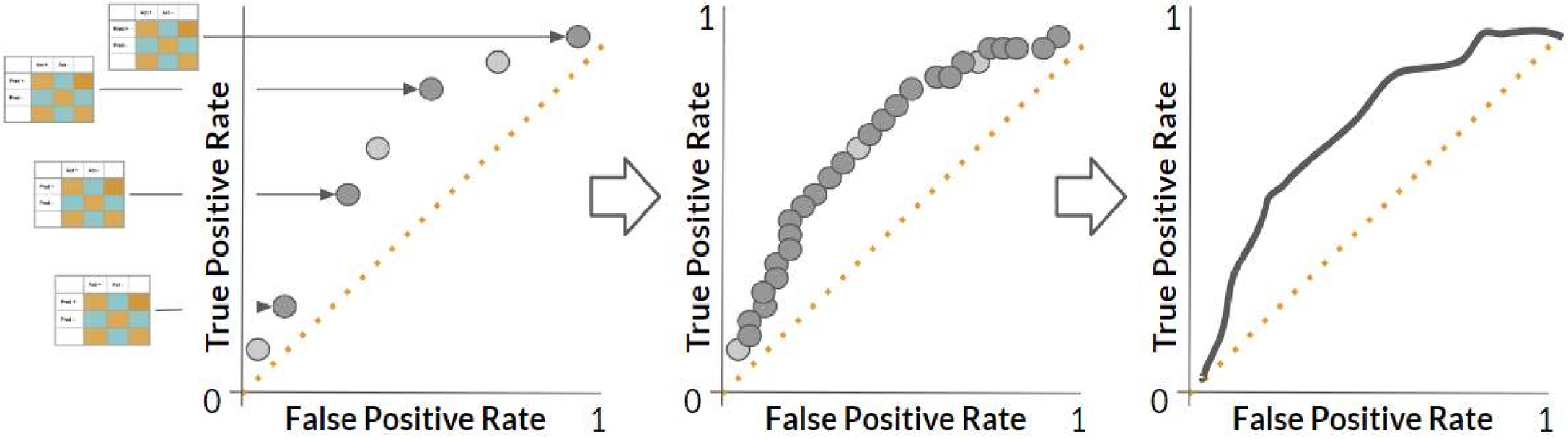
Threshold = 0.50

	Act +	Act -	
Pred +			
Pred -			

TPR = 0.42
FPR = 0.32



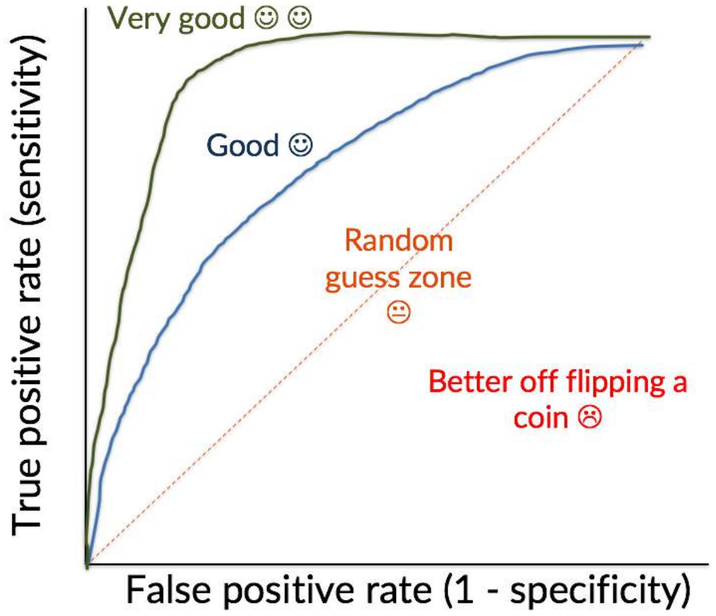
From points to a curve



- When we move thresholds, we re-calculate our metrics and create confusion matrices for every threshold
- Every time, we plot a new point in the **TPR** vs **FPR** space

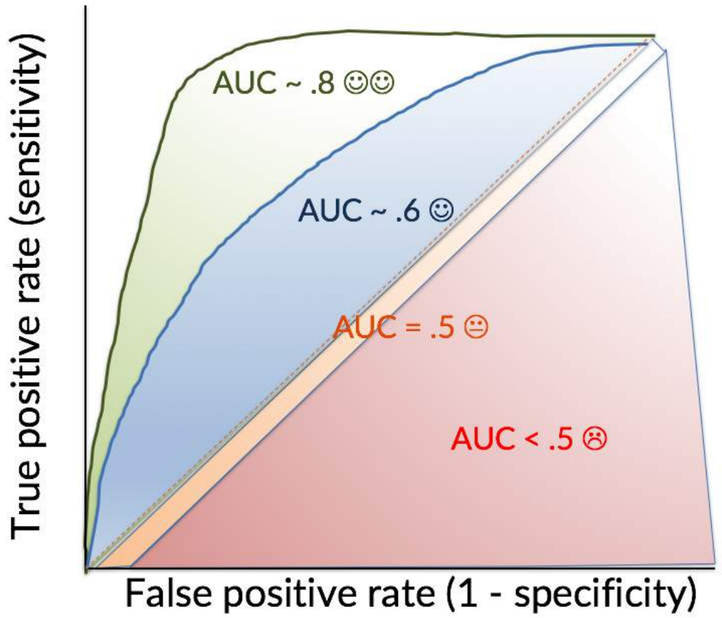
ROC: receiver operator characteristic

- The result of plotting points in **TPR** vs **FPR** space is a curve called **R**eceiver **O**perator **C**haracteristic (**ROC**)
- It shows a trade-off between the two rates
- It is a common and one of the best ways to assess performance of classification models

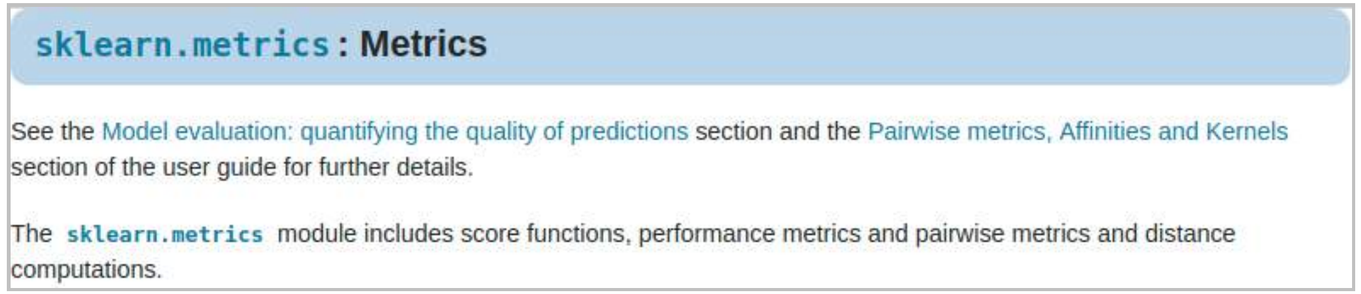


AUC: area under the curve

- It is a **performance metric** used to compare classification models to measure **predictive accuracy**
- The **AUC** should be **above .5** to say the model is better than a random guess
- The perfect **AUC** = 1 (you will never see this number working with real world data!)



scikit-learn: metrics package



sklearn.metrics : Metrics

See the Model evaluation: quantifying the quality of predictions section and the Pairwise metrics, Affinities and Kernels section of the user guide for further details.

The `sklearn.metrics` module includes score functions, performance metrics and pairwise metrics and distance computations.

- We will use the following methods from this library:
  - `confusion_matrix`
  - `accuracy_score`
  - `classification_report`
  - `roc_curve`
  - `auc`
- For all the methods and parameters of the `metrics` package, visit ***scikit-learn's documentation***

Confusion matrix and accuracy

Both `confusion_matrix` and `accuracy_score` take 2 arguments:

1. Original data labels
2. Predicted labels

```
# Take a look at test data confusion matrix.
conf_matrix_test = metrics.confusion_matrix(y_test, y_pred)
print(conf_matrix_test)
```

```
[[ 5 20]
 [ 4 46]]
```

```
# Compute test model accuracy score.
test_accuracy_score = metrics.accuracy_score(y_test, y_pred)
print("Accuracy on test data: ", test_accuracy_score)
```

```
Accuracy on test data:  0.68
```

Classification report

- To make interpretation of `classification_report` easier, in addition to the 2 arguments that `confusion_matrix` takes, we can add the actual class names for our target variable

```
# Create a list of target names to interpret class assignments.
target_names = ['Negative', 'Positive']
```

```
# Print an entire classification report.
class_report = metrics.classification_report(y_test,
                                             y_pred,
                                             target_names = target_names)
print(class_report)
```

```
              precision    recall  f1-score   support

    Negative       0.56      0.20      0.29        25
    Positive       0.70      0.92      0.79        50

    accuracy                           0.68        75
   macro avg       0.63      0.56      0.54        75
weighted avg       0.65      0.68      0.63        75
```

Classification report (cont'd)

```
print(class_report)
```

```
              precision    recall  f1-score   support

    Negative       0.56      0.20      0.29        25
    Positive       0.70      0.92      0.79        50

    accuracy                           0.68        75
   macro avg       0.63      0.56      0.54        75
weighted avg       0.65      0.68      0.63        75
```

- `precision` is **Positive Predictive Value** = **TP** / (**TP** + **FP**)
- `recall` is **TPR** = **TP** / Total positives
- `f1-score` is a weighted harmonic mean of `precision` and `recall`, where it reaches its best value at `1` and worst score at `0`
- `support` is actual number of occurrences of each class in `y_test`

Getting probabilities instead of class labels

```
# Get probabilities instead of predicted values.
test_probabilities = log_model.predict_proba(X_test)
print(test_probabilities[0:5, :])
```

```
[[0.20311326 0.79688674]
 [0.2984757  0.7015243 ]
 [0.25027588 0.74972412]
 [0.18437749 0.81562251]
 [0.4233091  0.5766909 ]]
```

```
# Get probabilities of test predictions only.
test_predictions = test_probabilities[: , 1]
print(test_probabilities[0:5])
```

```
[[0.20311326 0.79688674]
 [0.2984757  0.7015243 ]
 [0.25027588 0.74972412]
 [0.18437749 0.81562251]
 [0.4233091  0.5766909 ]]
```

Computing FPR, TPR and threshold

```
# Get FPR, TPR and threshold values.
fpr, tpr, threshold = metrics.roc_curve(y_test,           #<- test data labels
                                        test_predictions) #<- predicted probabilities
print("False positive: ", fpr)
```

```
False positive:  [0.   0.   0.   0.04 0.04 0.04 0.16 0.16 0.24 0.24 0.28 0.28 0.32 0.32
 0.36 0.36 0.6  0.6  0.64 0.64 0.72 0.72 0.72 0.8  0.8  1.  ]
```

```
print("True positive: ", tpr)
```

```
True positive:  [0.   0.02 0.28 0.28 0.32 0.36 0.36 0.54 0.54 0.6  0.6  0.68 0.68 0.72
 0.72 0.74 0.74 0.78 0.78 0.86 0.86 0.9  0.92 0.92 1.   1.  ]
```

```
print("Threshold: ", threshold)
```

```
Threshold:  [1.94647208 0.94647208 0.86101811 0.85414363 0.8446247  0.83469912
 0.82959738 0.80236104 0.79366218 0.78892501 0.78785443 0.77215489
 0.76599919 0.76333981 0.76315689 0.74972412 0.7015243  0.68955934
 0.67333992 0.61488059 0.58148866 0.5766909  0.57394379 0.51601976
 0.41688963 0.11792183]
```

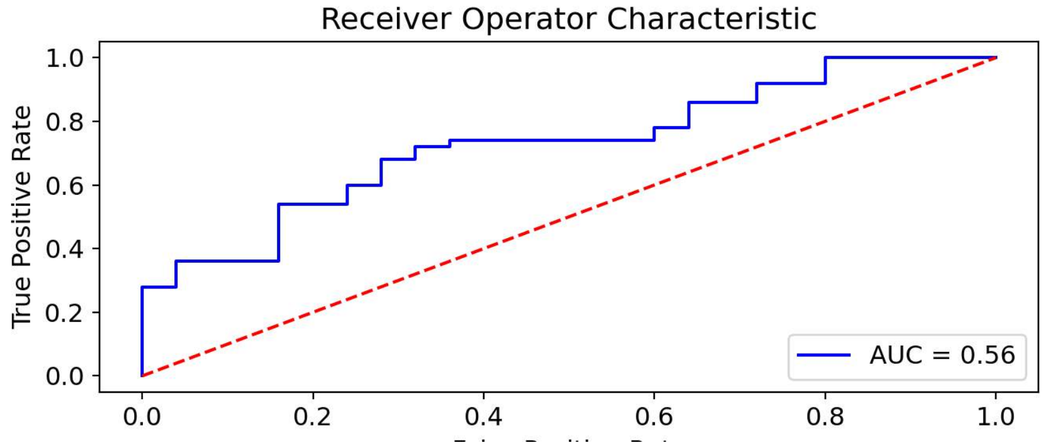
Computing AUC

```
# Get AUC value
auc = metrics.roc_auc_score(y_test,y_pred)
print("Area under the ROC curve: ", auc)
```

```
Area under the ROC curve:  0.56
```

Putting it all together: ROC plot

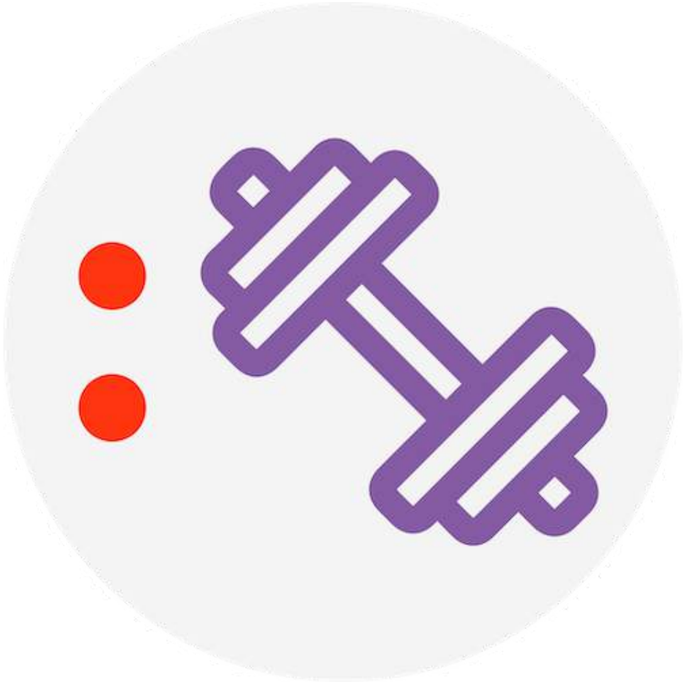
```
# Make an ROC curve plot.
plt.title('Receiver Operator Characteristic')
plt.plot(fpr, tpr, 'b', label = 'AUC = %0.2f' % auc)
plt.legend(loc = 'lower right')
plt.plot([0, 1], [0, 1],'r--')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.show()
```



- Our model has an accuracy of about `0.68`
- it's good, but might be based on a biased target
- We also have an AUC of about `0.56`, which is not all that great
- You may want to look at these aspects to improve the model:
  - balancing the target variable and methods to do so
  - using the TF-IDF matrix for a more weighted and normalized base

Knowledge check 2

Exercise 2



Module completion checklist

Objective	Complete
Split the data into train and test set for classification	✔
Explain the concept of logistic regression and how it will be used in this case	✔
Initialize, build, train the logistic regression model on our training dataset and predict on test	✔
Summarize classification performance metrics and methods to optimize the model	✔

Summary

- Today, we covered:
  - Logistic regression and how it will be used
  - How to initialize, build, train the logistic regression model
  - Classification performance metrics and methods to optimize the model
- In the next class, we will start talking about recommender systems.

This completes our module