DATA SOCIETY®

Advanced text mining - part 2

"One should look for what is and not what he thinks should be."
-Albert Einstein.

Welcome back!

- In this module, we'll walkthrough using a Document Term Matrix (DTM) to classify the sentiment of each newspaper snippet in our NY Times file as either positive or negative
- You'll then have the opportunity to practice analyzing the sentiment of movie reviews from a popular review website as part of the exercises
- First, let's refresh ourselves on **bag-of-words** analysis and the steps to get to a finalized matrix

Module completion checklist

Objective	Complete
Review and define the outcome of bag-of-words analysis	
Summarize the concept of sentiment analysis	
Classify each cleaned sentence from the cleaned text file as positive, negative, or neutral and store labels	
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	

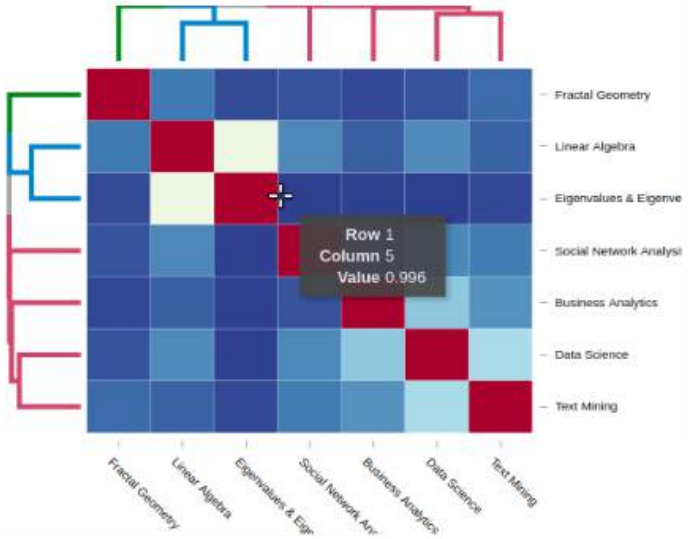
"Bag-of-words" analysis

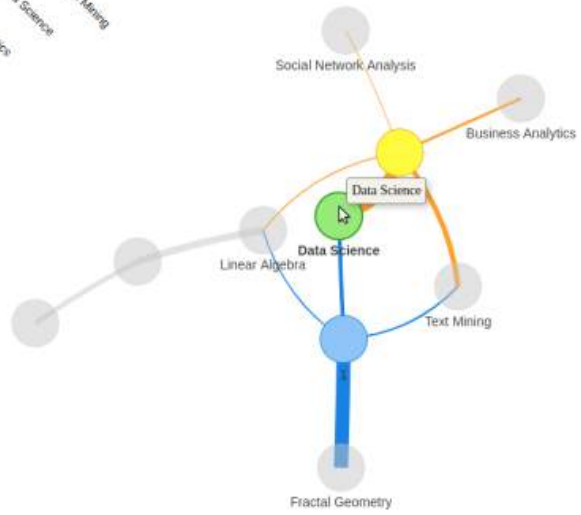
- How do we quantify and compute on text data? - *We need to translate words into numbers*
- How do we translate words into numbers? - *The simplest solution is to **count** them*
- The analysis of text data based on word counts (a.k.a. frequencies) in documents is called **bag-of-words**

Index	Word	Freq	%
A	Apple	5	20
B	Book	7	28
C	Cat	13	52

"Bag-of-words" analysis: use cases

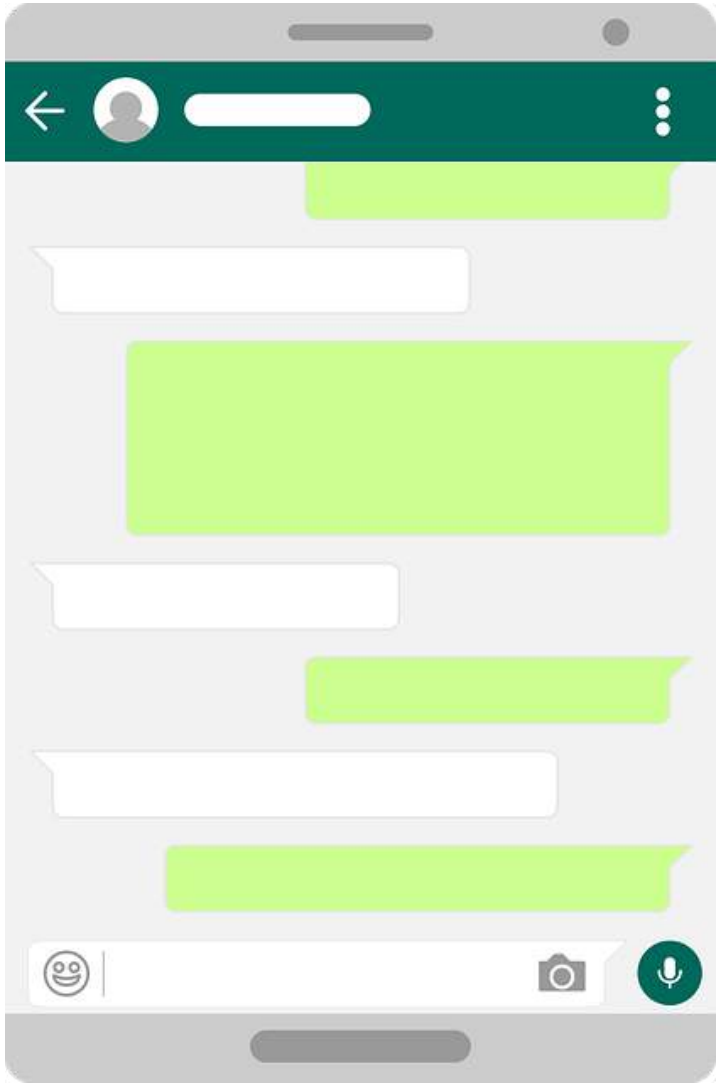
- While a seemingly *crude* approach, it's useful for quite a few things:
  - Word and document similarity query processing
  - Word and document clustering
  - Topic modeling
  - Sentiment analysis
  - Automated document summarization





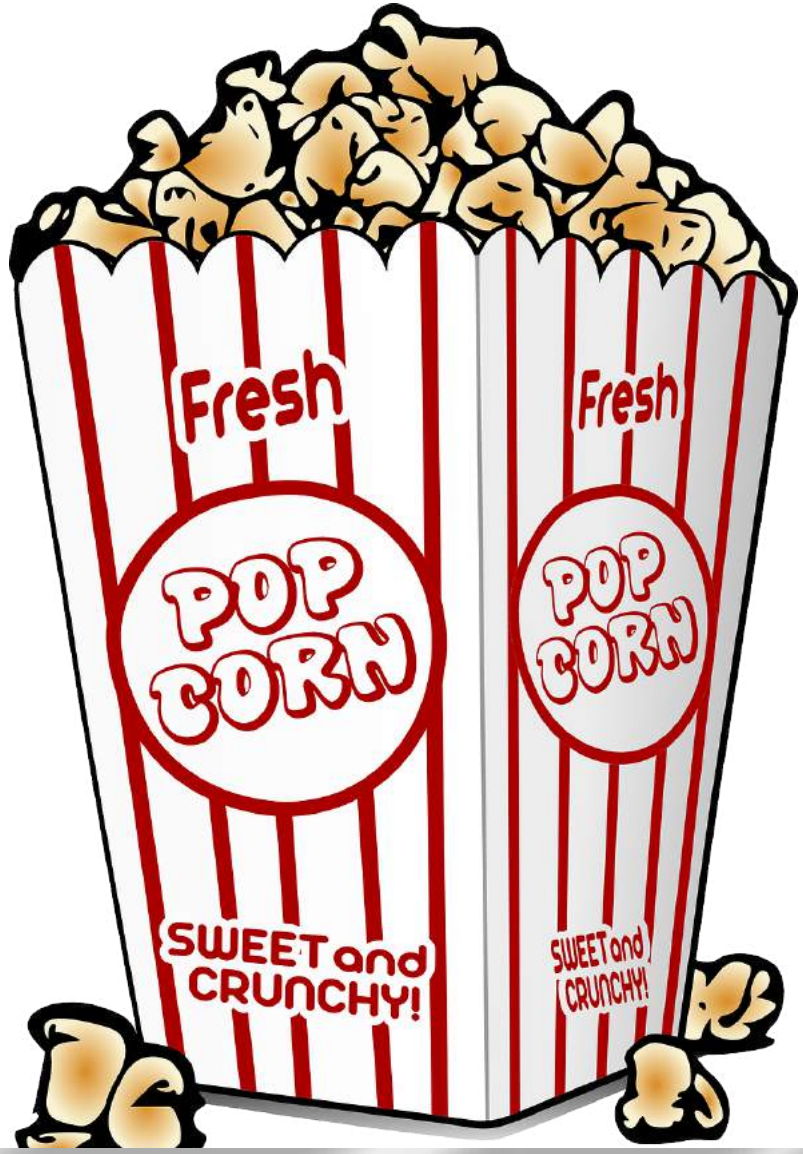
Use cases in customer support

- Working with **customer support data**
  - Analyzing positive and negative sentiment, trying to improve customer service
- Working with **chat messages**
  - Analyzing positive and negative sentiment, trying to detect and improve the biggest triggers for unhappy customers



Use cases in sales and product development

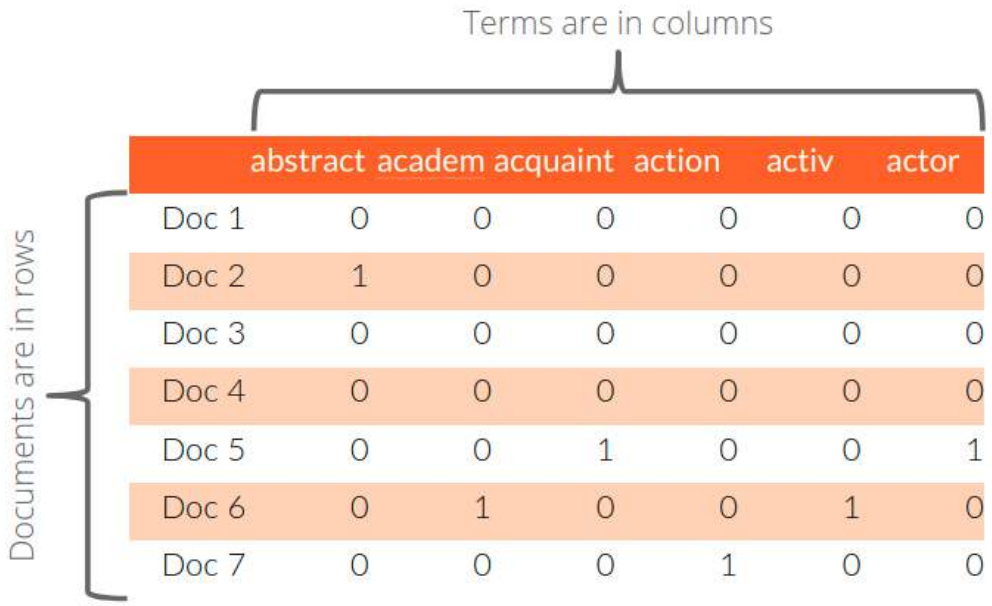
- Working with **consumer reviews**
  - Analyzing positive and negative sentiment, trying to improve product offering
- Working with **movie reviews**
  - Analyzing positive and negative sentiment, trying to improve movie selection offered with subscription service



"Bag-of-words" analysis: key elements

What we have
A corpus of documents cleaned and processed in a certain way • All words a converted to lower case • All punctuation, numbers, and special characters are removed • Stopwords are removed • Words are stemmed to their root form (and sometimes lemmatized)
A Document-Term Matrix (DTM): with counts of each word recorded for each document
A transformed representation of a Document-Term Matrix (i.e. weighted with TF-IDF weights)

Refresher: what is a DTM?



	abstract	academ	acquaint	action	activ	actor
Doc 1	0	0	0	0	0	0
Doc 2	1	0	0	0	0	0
Doc 3	0	0	0	0	0	0
Doc 4	0	0	0	0	0	0
Doc 5	0	0	1	0	0	1
Doc 6	0	1	0	0	1	0
Doc 7	0	0	0	1	0	0

- **Document-term matrix** is simply a matrix of unique words counted in each document:
  - Documents are arranged in the rows
  - Unique terms are arranged in columns
- The corpus **vocabulary** consists of all of the unique terms (i.e. column names of DTM) and their total counts across all documents (i.e. column sums)

Module completion checklist

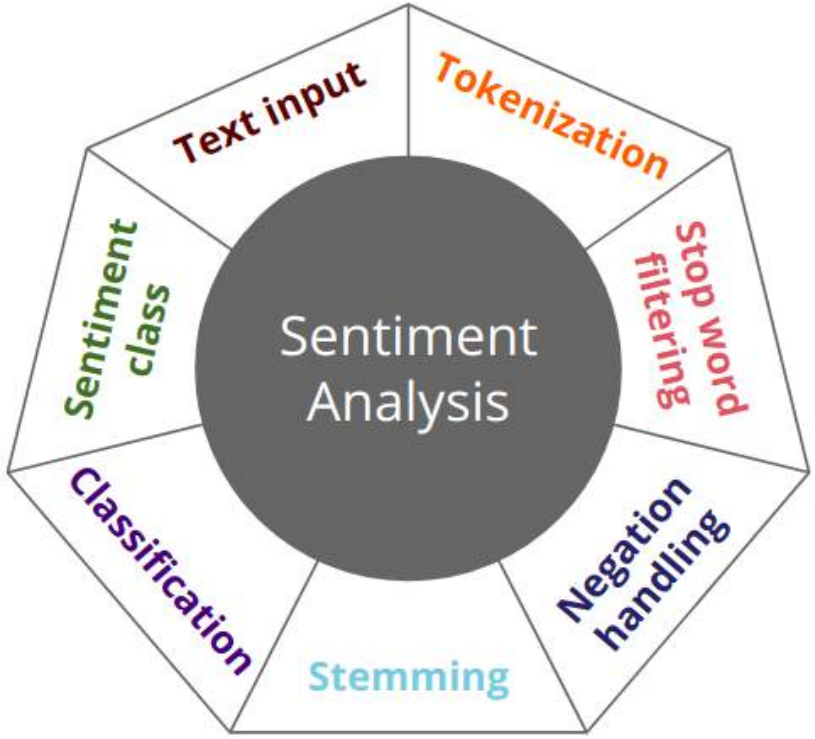
Objective	Complete
Review and define the outcome of bag-of-words analysis	✔
Summarize the concept of sentiment analysis	
Classify each cleaned sentence from the cleaned text file as positive, negative, or neutral and store labels	
Split the data into train and test set for classification	
Explain the 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	

Sentiment analysis - use cases

- Sentiment mining is used in many areas:
  - opinion mining
  - reputation monitoring
  - business analytics
- It helps businesses understand their customers' experience
- **How do you see sentiment analysis as a helpful tool within your job/field?**

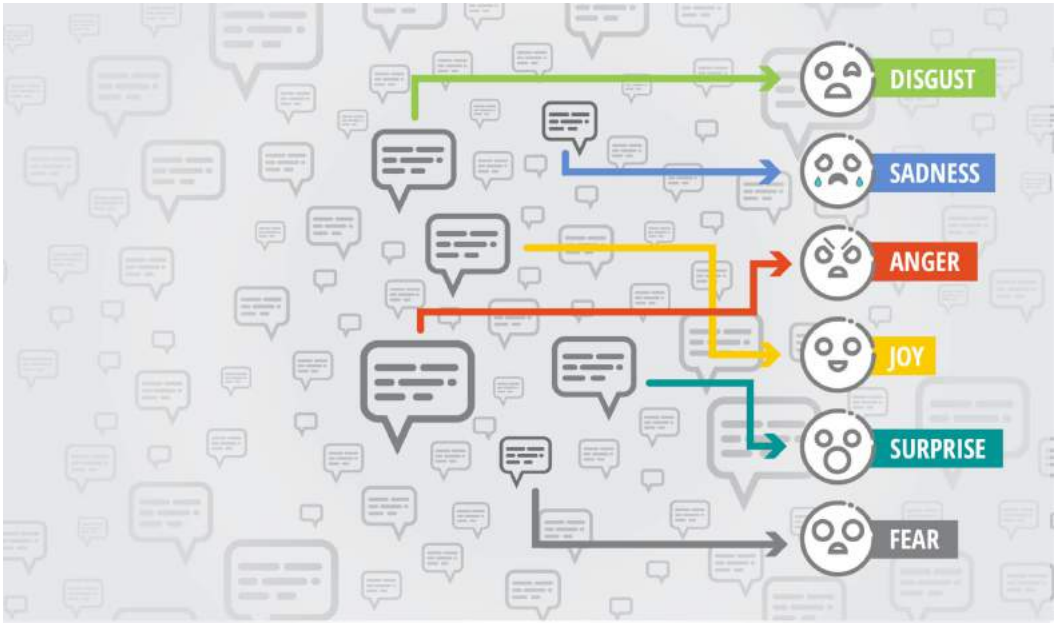
Sentiment analysis - steps

- There are **many ways to build out sentiment analysis**, including what we will use today
- Some of the processes will use more complex methodology / algorithms, but **today, we want to understand the basis of sentiment analysis**
- **Using these base steps, you will be able to build upon it** on your own as you continue using sentiment analysis
- *Today, you will have some time to tweak the base model that we build in a workshop at the end of this class*



Sentiment analysis in Python

- **Python is very powerful when it comes to text mining, as you have seen**
- We will continue to use the `NLTK` package
- We will also use the `scikit-learn` library so that we can build a model that will classify future documents
- We will split the process we will be using today into two parts for simplicity - text classification and model building
- **Our final result in this module will be a logistic regression model that classifies newspaper snippets as either 'negative' or 'positive'**



Sentiment analysis in Python

Text classification

- **Classify** articles using NLTK's `SentimentIntensityAnalyzer`
- **Load the DTM** `pickle` (if we did not have one, we would build a DTM at this step)

Model building

- **Split the DTM into train and test datasets**, including the sentiment labels for each article
- **Build a logistic regression model** on the train dataset that classifies texts as one of the two categories, "negative" or "positive"
- **Analyze results** by predicting on the test set and on new data
- **Optimize the model** after inspecting results
- **Update the model** when new data comes in to have a continuously updated sentiment model

Knowledge check 1

Module completion checklist

Objective	Complete
Review and define the outcome of bag-of-words analysis	✔
Summarize the concept of sentiment analysis	✔
Classify each cleaned sentence from the cleaned text file as positive, negative, or neutral and store labels	
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`
- Let the `main_dir` be the variable corresponding to your `booz-allen-hamilton` folder

```
from pathlib import Path
# Set `home_dir` to the root directory of your computer.
home_dir = Path.home()

# Set `main_dir` to the location of your `booz-allen-hamilton` folder.
main_dir = home_dir / "Desktop" / "booz-allen-hamilton"

# Make `data_dir` from the `main_dir` and remainder of the path to data directory.
data_dir = main_dir / "data"
```

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
```

```
# 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
```

- **Note:** If you have `scikit-learn v0.19` or lower, substitute `sklearn.model_selection` with `sklearn.cross_validation` to import `train_test_split` package!

Working directory

- Set the working directory to `data_dir`

```
# Set working directory.
os.chdir(data_dir)
```

```
# Check working directory.
print(os.getcwd())
```

```
/home/[user-name]/Desktop/booz-allen-hamilton/data
```

Text classification - classify snippets

Classify

- Today, we are going to classify each of our newspaper snippets using the `SentimentIntensityAnalyzer` function from the `vader` package
- They will be classified by document and classified as either:
  - "negative"
  - "positive"

Text classification - classify snippets

Classify

- We already have cleaned the snippets and saved them as a `pickle`
- Let's load the snippets to see what we are working with

```
cleaned_txt = pickle.load(open((data_dir + '/NYT_clean_list.sav'),"rb"))
```

Text classification - classify snippets

Classify

```
print(cleaned_txt[0:10])
```

```
['pakistan struggl batsmen must find way handl south africa potent pace attack claw way back seri
second test start like live newland wicket thursday', 'nation footbal leagu microscop lack minor head
coach recent slew fire leagu', 'hit hot streak right time goal golf top male profession year new
calendar cram major championship super busi stretch', 'pope franci usher new year ode motherhood
tuesday remind faith mother exampl embrac best antidot today disjoint world solitud miseri', 'chri
froom defend giro titl year choos focu win fifth tour de franc crown instead team sky announc tuesday',
'pakistan former prime minist nawaz sharif appeal convict prison sentenc hand islamabad tribun last
week', 'thousand demonstr march hong kong tuesday demand full democraci fundament right even independ
china face mani see mark clampdown communist parti local freedom', 'nick kyrgio start brisban open titl
defens battl victori american ryan harrison open round tuesday', 'british polic confirm tuesday treat
stab attack injur three peopl manchest victoria train station terrorist investig search address
cheetham hill area citi', 'marcellu wiley still fenc let young son play footbal former nfl defens end
fox sport person tell podcaston sport like nfl tri make footbal safer game de']
```

- You can see we have a list of each snippet as a string
- Each snippet has been cleaned and stemmed, ready for scoring

Text classification - introducing vader

Classify

- As mentioned earlier, we will be using the `SentimentIntensityAnalyzer` function from the `vader` package, which is in the `NLTK` library
- We loaded this function earlier, let's take a quick look at what it does:
  - **VADER** = **V**alence **A**ware **D**ictionary for s**E**ntiment **R**easoning
  - `vader` sentiment analysis relies on a dictionary which maps lexical features to emotion intensities called *sentiment scores*
  - `vader.SentimentIntensityAnalyzer` will return a score in the range -1 to 1 from most negative to positive
  - The sentiment score is calculated by summing up the sentiment scores of each `vader` dictionary listed word in the sentence
  - `vader` best works on short documents, like this example, and tweets and other types of messages
  - You can go to the ***nltk.sentiment.vader module documentation*** page for the source code

Text classification - classify

Classify

- We will now use `vader.SentimentIntensityAnalyzer` to classify our loaded and cleaned snippets
- Within this function, we will be specifically using `SentimentIntensityAnalyzer.polarity_scores`
- This will give us a score between -1 and 1 and then compound the negative, neutral, and positive valences to give us a combined score, which is the score we will use today

Text classification - classify (cont'd)

class
nltk.sentiment.vader.**SentimentIntensityAnalyzer**(*lexicon_file='sentiment/vader_lexicon.zip/vader_lexicon/vader_lexicon.txt'*) [source]

Bases: `object`

Give a sentiment intensity score to sentences.

make_lex_dict() [source]

Convert lexicon file to a dictionary

polarity_scores(*text*) [source]

Return a float for sentiment strength based on the input text. Positive values are positive valence, negative value are negative valence.

score_valence(*sentiments*, *text*) [source]

sentiment_valence(*valence*, *sentitext*, *item*, *i*, *sentiments*) [source]

nltk.sentiment.vader.**allcap_differential**(*words*) [source]

Check whether just some words in the input are ALL CAPS

Parameters: **words** (*list*) – The words to inspect
Returns: *True* if some but not all items in *words* are ALL CAPS

nltk.sentiment.vader.**negated**(*input_words*, *include_nt=True*) [source]

Determine if input contains negation words

nltk.sentiment.vader.**normalize**(*score*, *alpha=15*) [source]

Normalize the score to be between -1 and 1 using an alpha that approximates the max expected value

nltk.sentiment.vader.**scalar_inc_dec**(*word*, *valence*, *is_cap_diff*) [source]

Check if the preceding words increase, decrease, or negate/nullify the valence

Text classification - classify (cont'd)

Classify

- Let's take a look at what the function outputs when iterating through each cleaned sentence

```
# Initialize the `SentimentIntensityAnalyzer().`
sid = SentimentIntensityAnalyzer()

# Iterate through each sentence printing out the scores for each.
for sentence in cleaned_txt:
     print(sentence)
     ss = sid.polarity_scores(sentence)
     for k in ss:
         print('{0}: {1}, '.format(k, ss[k]), end='')
     print()
```

```
pakistan struggl batsmen must find way handl south africa potent pace attack claw way back seri second
test start like live newland wicket thursday
neg: 0.112, neu: 0.797, pos: 0.091, compound: -0.1531,
nation footbal leagu microscop lack minor head coach recent slew fire leagu
neg: 0.32, neu: 0.68, pos: 0.0, compound: -0.5719,
hit hot streak right time goal golf top male profession year new calendar cram major championship super
busi stretch
neg: 0.0, neu: 0.65, pos: 0.35, compound: 0.8225,
pope franci usher new year ode motherhood tuesday remind faith mother exampl embrac best antidot today
disjoint world solitud miseri
neg: 0.0, neu: 0.72, pos: 0.28, compound: 0.7906,
chri froom defend giro titl year choos focu win fifth tour de franc crown instead team sky announc
tuesday
neg: 0.0, neu: 0.826, pos: 0.174, compound: 0.5859,
```

Text classification - classify (cont'd)

Classify

- We use `SentimentIntensityAnalyzer` to write a function that:
  - takes the cleaned text, runs the `SentimentIntensityAnalyzer` on each snippet
  - outputs results in four scores: **“neg”,“neu”,“pos”,“compound”**
  - takes the “compound” score and uses that to analyze if the snippet is negative or positive
  - Returns the labels for each snippet in a list

Text classification - classify (cont'd)

Classify

```
# This function outputs a list of labels for snippet:
def sentiment_analysis(texts):
        list_of_scores = []
        for text in texts:
            sid = SentimentIntensityAnalyzer()
            compound = sid.polarity_scores(text)["compound"]
            if compound >= 0:
                list_of_scores.append("positive")
            else:
                list_of_scores.append("negative")
        return(list_of_scores)
```

```
score_labels =
sentiment_analysis(cleaned_txt)
```

```
print(score_labels[1:5])
```

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

- You now have a list `score_labels` that contains a sentiment classification for each snippet
- These will be used as your `y` variable when you build your train and test datasets

Text classification - Load the DTM

Load the DTM

- Let's load the `DTM pickle`
- We will convert it to an array for easier usage and call the array `DTM_array`
- If we had not built it, we would have to transform our `.txt` file to `DTM` now

```
DTM_matrix = pickle.load(open((data_dir + '/DTM_matrix.sav'),"rb"))
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**

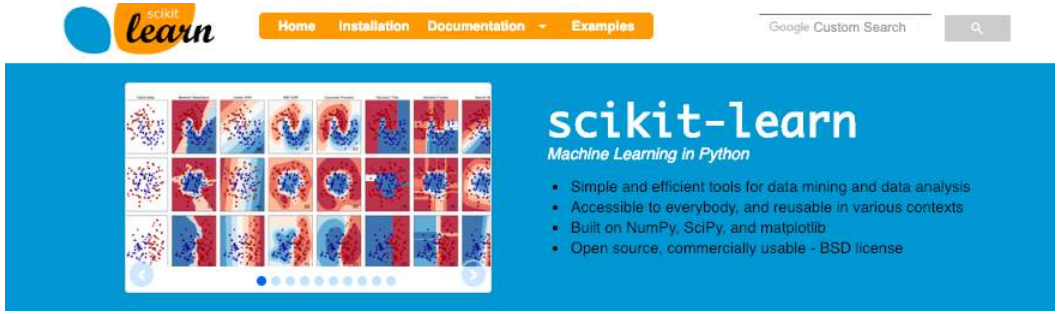
Module completion checklist

Objective	Complete
Review and define the outcome of bag-of-words analysis	✔
Summarize the concept of sentiment analysis	✔
Classify each cleaned sentence from the cleaned text file as positive, negative, or neutral and store labels	✔
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 - split the dataset

Split the DTM into train and test datasets

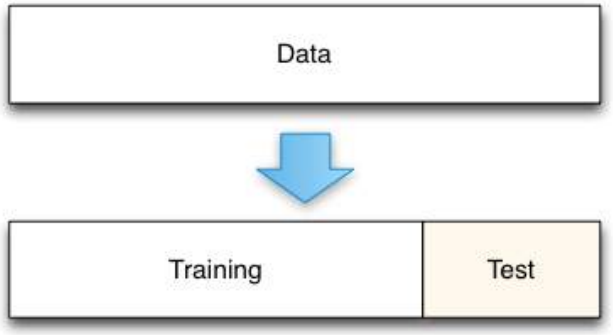
- We now start going into the model-building steps
- We will be using a well-known library, `scikit-learn`
- This library contains many well-known **machine learning** packages
- For documentation and more information, you can go to the ***scikit-learn homepage***



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:

1. `DTM_array` : the sparse matrix with the word counts for each 'document' (newspaper snippets in our case)
2. `score_labels` : the sentiment we calculated for each document, which is our target variable that we want to predict
3. `train_size` : the size of the training dataset, here we choose .7 or 70% of the entire dataset
4. `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
```

Knowledge check 2

Exercise 1

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

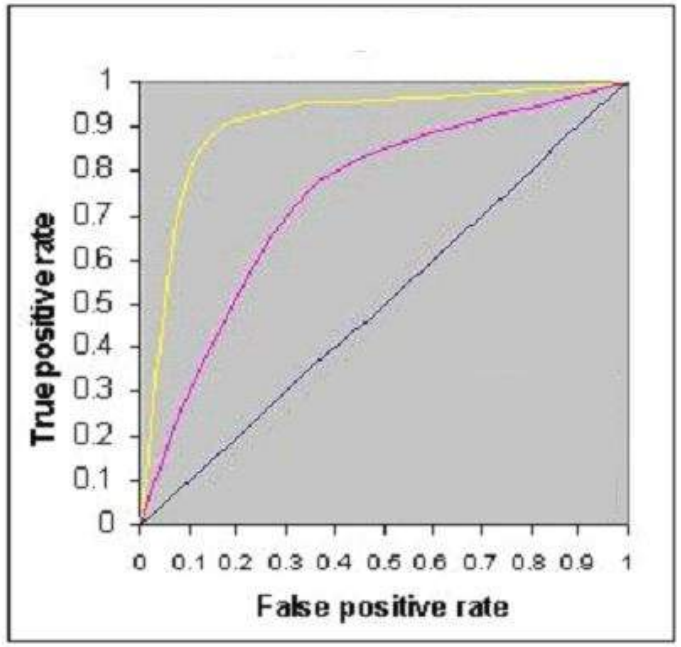
The sigmoid function plays a key role in logistic regression

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
Review and define the outcome of bag-of-words analysis	✔
Summarize the concept of sentiment analysis	✔
Classify each cleaned sentence from the cleaned text file as positive, negative, or neutral and store labels	✔
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:

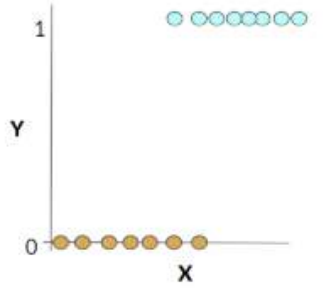
- Easily tweak its performance by adjusting a **cut-off probability** instead of re-running the model with new parameters

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

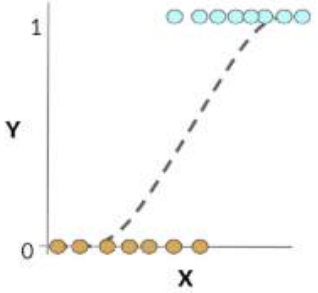
- Create **robust**, **efficient** and **well-optimized models**

Logistic regression: process

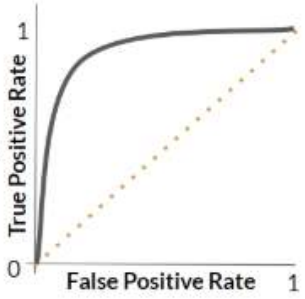
Step 1:
Convert target variable to 1/0



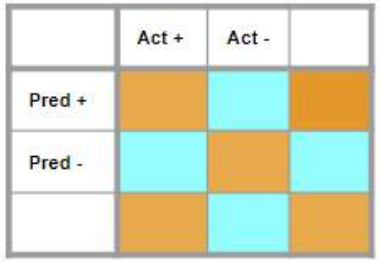
Step 2:
Logistic regression on training data



Step 3:
Use ROC curve & AUC to pick threshold



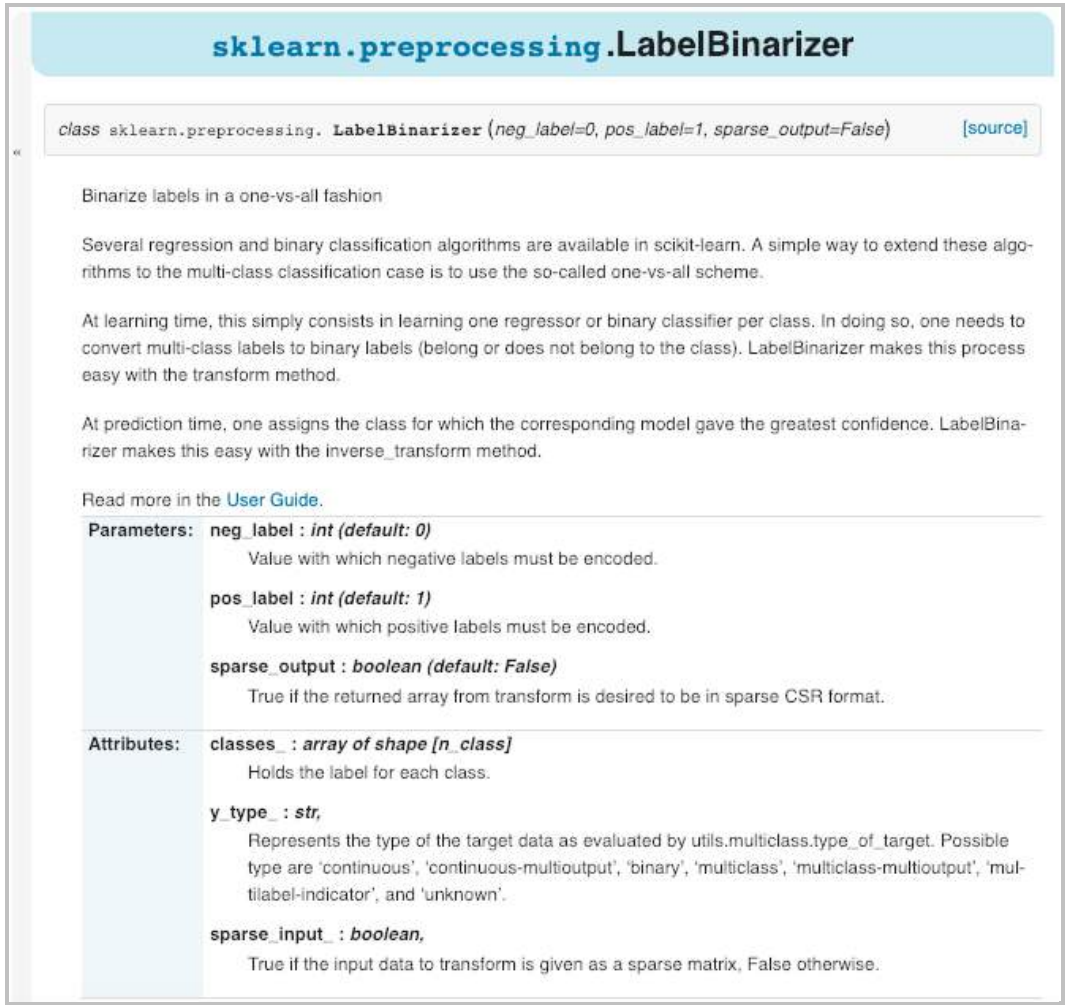
Step 4:
Check performance on test data



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'.

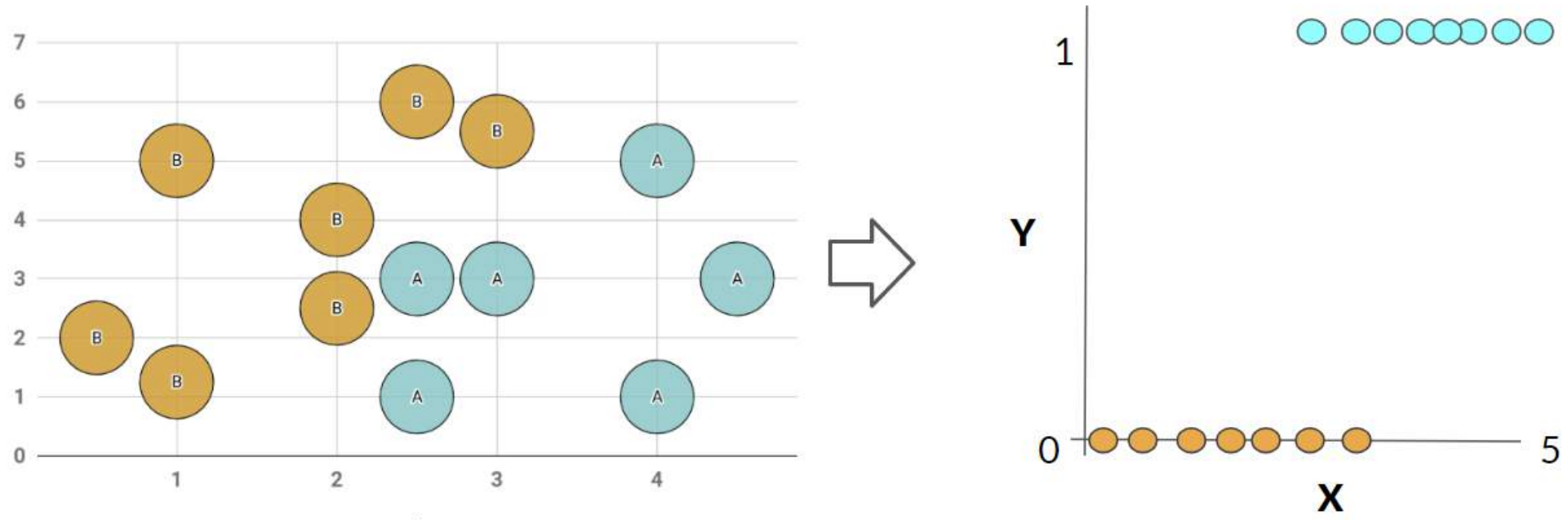
sparse_input_ : *boolean,*
True if the input data to transform is given as a sparse matrix, False otherwise.

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

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

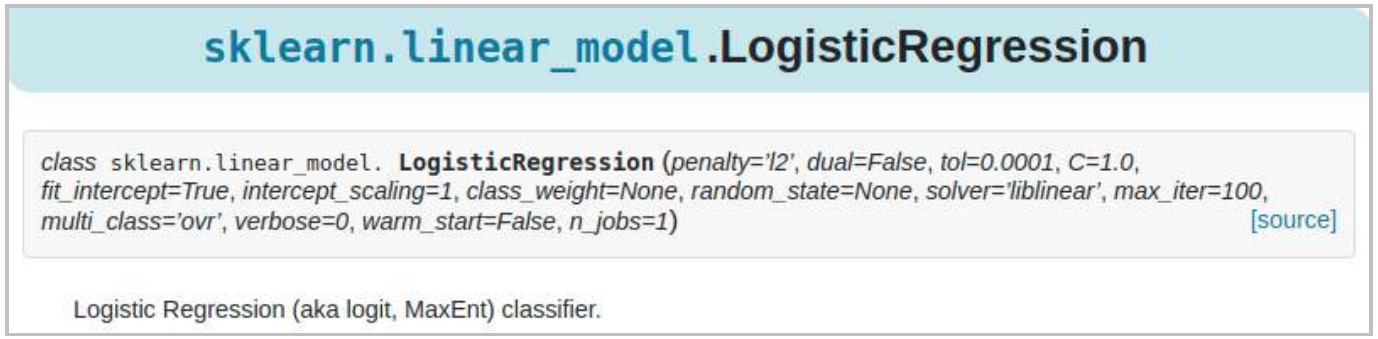
- Due to the properties and range of probabilities it produces, the sigmoid function is often used as an **activation function** in **neural networks** algorithms

Module completion checklist

Objective	Complete
Review and define the outcome of bag-of-words analysis	✔
Summarize the concept of sentiment analysis	✔
Classify each cleaned sentence from the cleaned text file as positive, negative, or neutral and store labels	✔
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)
```

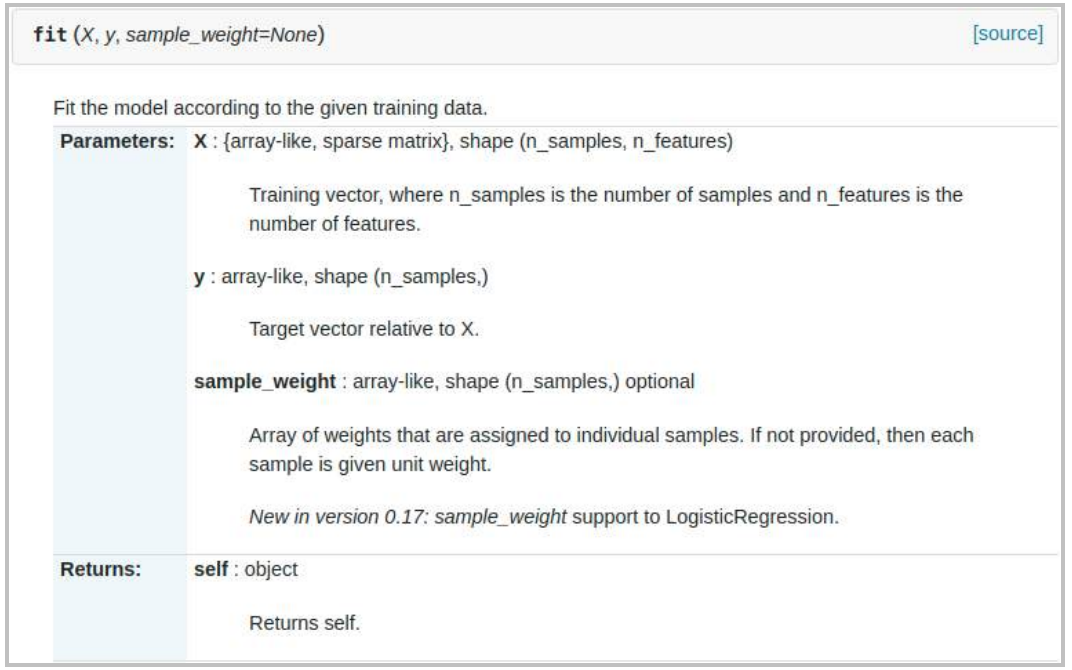
```
LogisticRegression()
```

Logistic regression: fit

The two main arguments are the same as with most classifiers in `scikit-learn`:

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

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.

The main argument is the same as with most classifiers in `scikit-learn`:

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

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 3

Exercise 2

Module completion checklist

Objective	Complete
Review and define the outcome of bag-of-words analysis	✔
Summarize the concept of sentiment analysis	✔
Classify each cleaned sentence from the cleaned text file as positive, negative, or neutral and store labels	✔
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

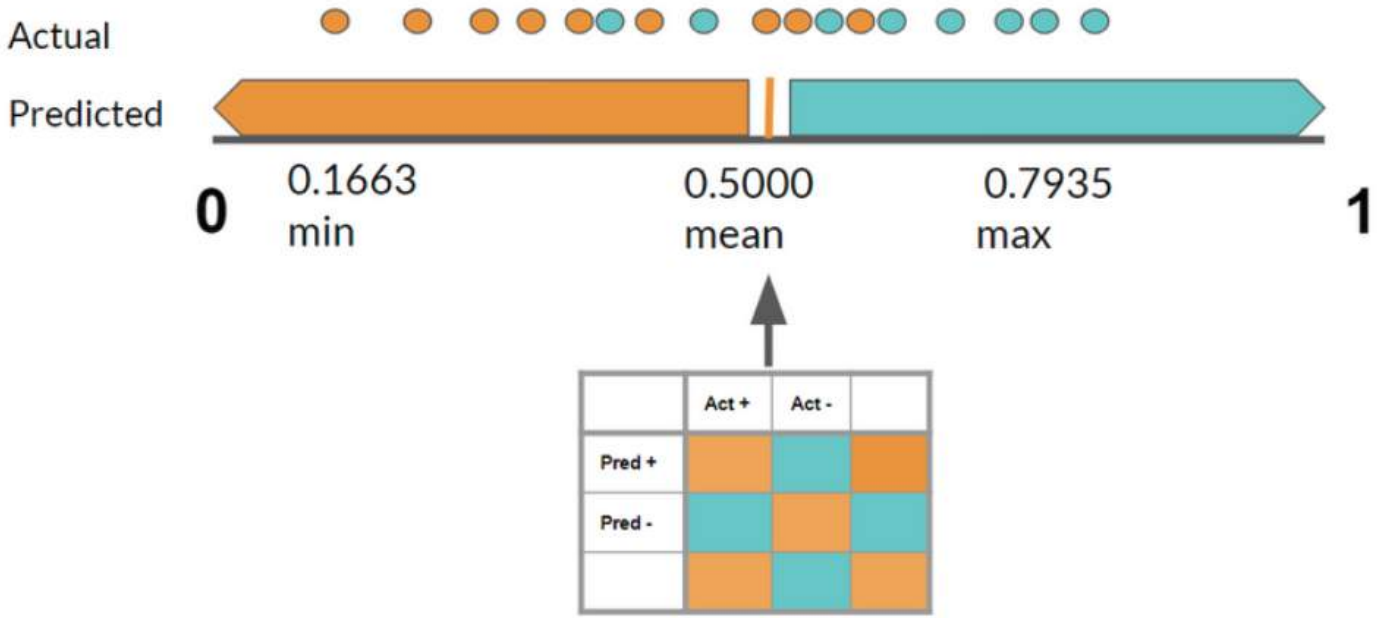
Confusion matrix recap

	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 negatives
- **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 `High yield/Low yield` 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

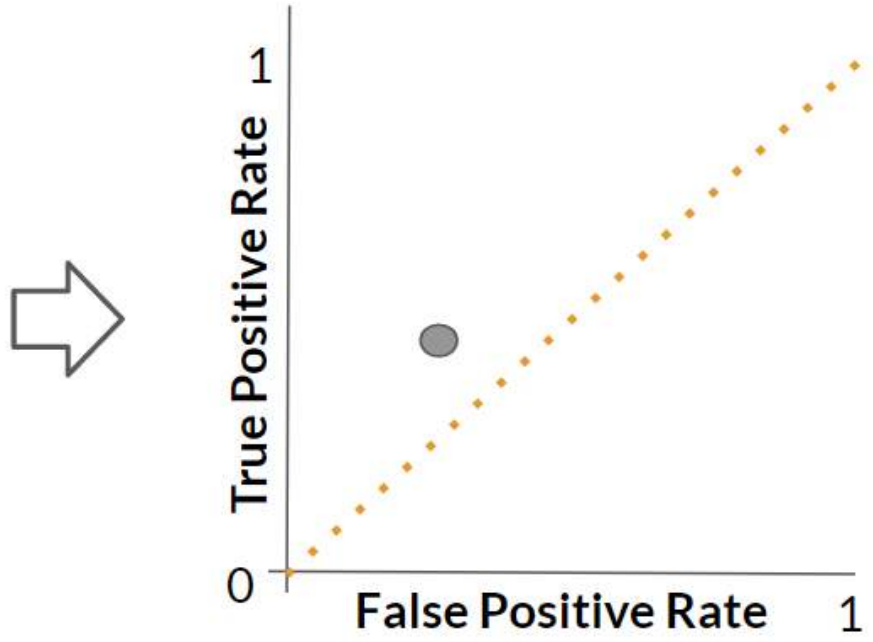
Each threshold can create a confusion matrix, which can be used to calculate a point in space defined by:

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

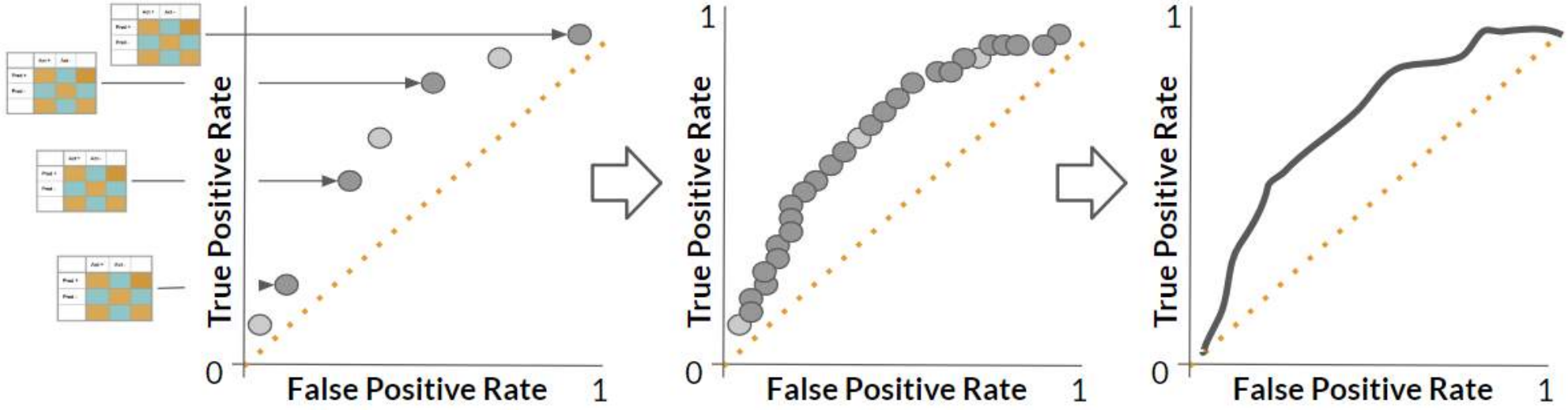
Threshold = 0.50

	Act +	Act -	
Pred +			
Pred -			

TPR = 0.42
FPR = 0.32



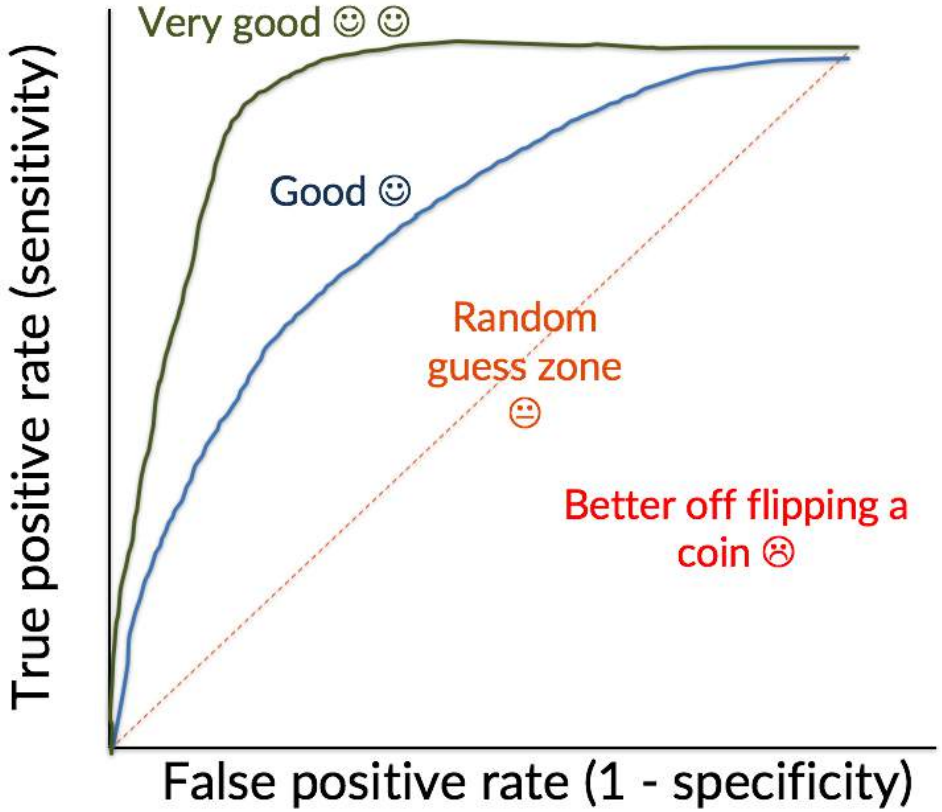
From points to a curve



- 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

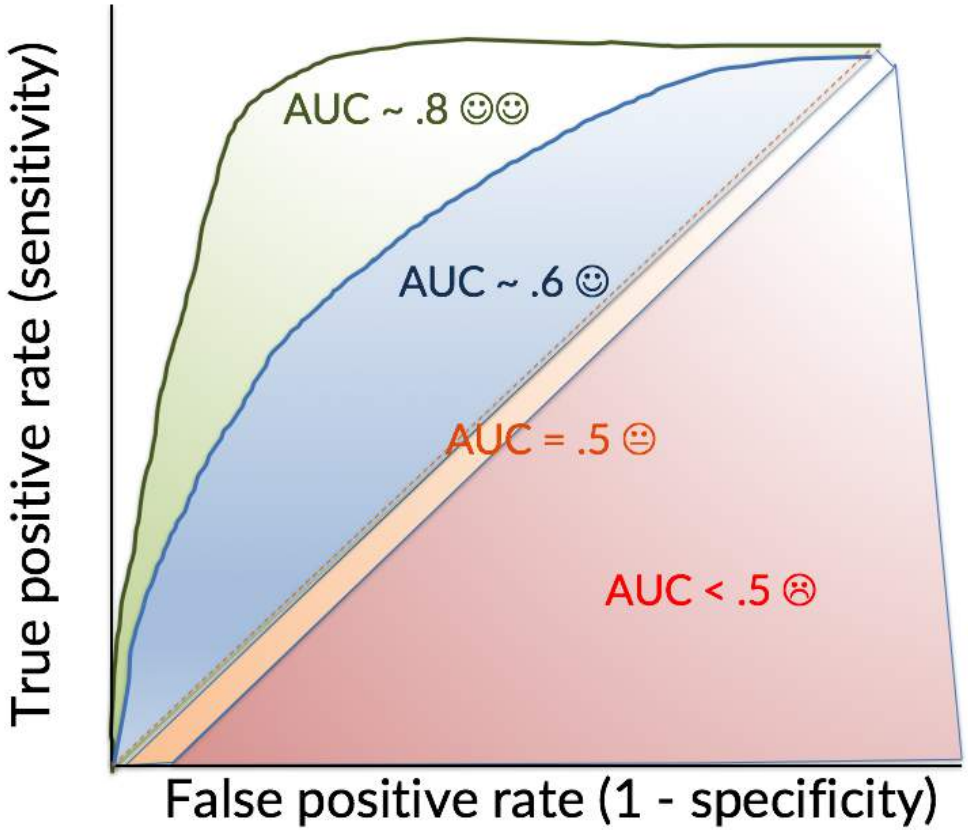
ROC: receiver operator characteristic

- 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

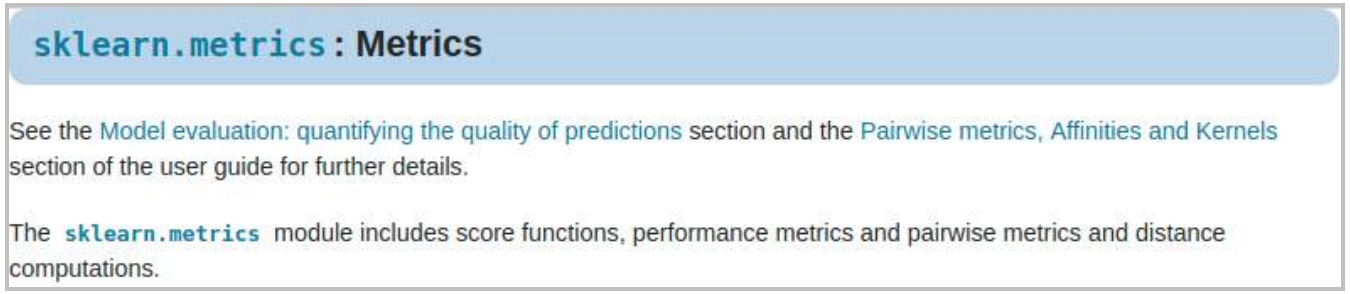


AUC: area under the curve

- 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]
```

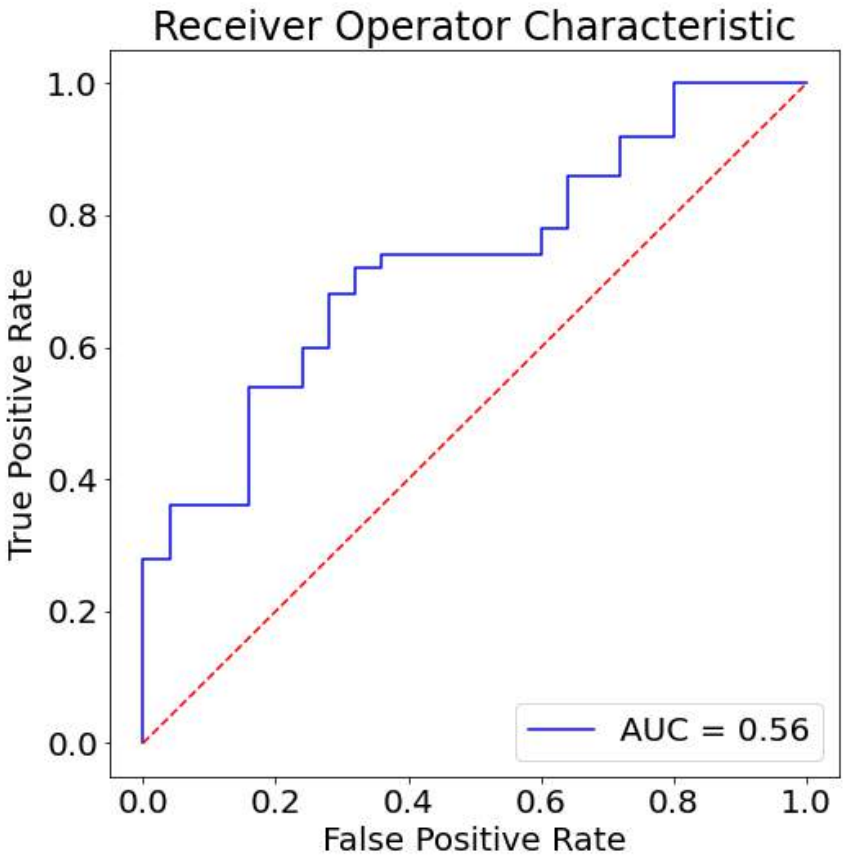
Computing AUC

```
# Get AUC by providing the FPR and TPR.
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 a great AUC of about `0.56`, which is not all that great
- You may want to look at these aspects during the workshop 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 4

Exercise 3

Module completion checklist

Objective	Complete
Review and define the outcome of bag-of-words analysis	✔
Summarize the concept of sentiment analysis	✔
Classify each cleaned sentence from the cleaned text file as positive, negative, or neutral and store labels	✔
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	✔

This completes our module

Congratulations!