Key Aspects:

Create DataFrame – spark.createDataFrame

( [ (tuple\_key1, “tuple\_value1”), (tuple\_key2, “tuple\_value2”), (tuple\_key3, “tuple\_value3”) ], [“column\_name1”, “column\_name2”] )

dataframe.show( )

Read data and create dataframe from S3 Bucket in AWS

spark.sparkContent.addFile(url\_for\_CSV\_file)

df = spark.read.csv(SparkFiles.get(“CSV\_File\_Name”), sep=”,”, header=True)

Key Module Commands

- dataframe\_name.printSchema( ) – Provides the schema for the dataframe

- dataframe\_name.columns

- dataframe\_name.describe( )

- dataframe\_name.show( ) – shows the column names and a few rows 🡪 similar to PANDAS head( )

- dataframe\_name.select(“column\_name”) – Select the column in a dataframe

- dataframe\_name.filter(“price<20”) – Filters a dataframe by values in “points” column greater than 20

Can also link .select after filter and grab the specific columns needed from the filtered data

Can also filter with the format:

df.filter(df[“column\_name”] == “exact\_value\_entry”).show( )

Manipulate Columns in Spark:

- Duplicate Column Into New Column 🡪

dataframe\_name.withColumn(‘column\_name’, dataframe\_name[‘new\_column\_name’] ).show( )

- Rename Column 🡪

dataframe\_name.withColumnRenamed(‘column\_name’, ‘updated\_name’).show( )

- Multiply All Column Values 🡪

dataframe\_name.withColumn(‘multiplied\_column’, dataframe\_name[‘column\_name’]\*random\_number\_#).show( )

Transformation – Listing and reading the instructions of Spark programs

Order a Dataframe by Ascending Values: The “points” column, in this case

df.orderBy(df[“points”].desc( ) )

Calculate the average of column values

df.select(avg(“points”))

Action – the directive to complete the instructions

Show the ordered Dataframe

df.orderBy(df[“points”].desc( ) ).show(5)

Language Processing:

The NLP Pipeline Sequence:

1. Raw text – Input text data
2. Tokenization – Separate and filter paragraphs or sentences into individual words
3. Stop Words Filtering – Remove common words that add no real value to the data
4. TF-IDF (Term Frequency-Inverse Document Frequency) – Statistically rank the words by importance compared to the rest of the words in the text, also includes converting text to numbers for machine recognition

TF-IDF = TF \* IDF

1. Machine Learning – Put everything together and run through machine learning model to produce an output analysis

**Tokenization** – Machines understanding natural language by breaking up sentences into strings and provided texts as arrays of strings to represent sentences in a text.

Tools to be used: NLTK, spaCy

from pyspark.ml.feature import Tokenizer

tokenizer = Tokenizer(inputCol=”sentence”, output=”words”)

tokenizer

tokenized\_df = tokenizer.transform(dataframe)

tokenized\_df.show(truncate=False)

**Normalization** – Fixing misspellings and converting them into their original form. Two methods exist:

***Stemming*** – Removes the suffix and reduces the suffix to a rough version of the word, where the “stem” of the word is taken and considered the base word. Can be an issue with language when the word “ponies” does not have the original word “pony” come from removing the plural form. Instead it yields “poni”, which isn’t a word on its own.

***Lemmatization*** – Removes the suffix and reduces it to the original form, uses a lexicon reference to reduce a word like “am” to the simplified dictionary concept “to be / be”. Less abstract and less sophisticated, but more effective at consistently yielding real understandable words.

Natural Language Generation – Used to generate new text (application: chatbots, automated custom reports, custom webpage content)

Bag-of-Words – A model using the most frequently used words for reference in the natural language generation job in question. (sources to draw on: a document of words to be read in but not weighted by order and only referenced by the count of each word)

N-gram – Creating groupings of items from text, particularly cases where a series of words appear often in a consecutive order. The frequency of which, is used to gather the meaning of a text’s line with greater accuracy. Length of words per grouping defines the “n” aspect, and common terms exist such as: unigram – 1-gram, bigram – 2-gram, trigram – 3-gram. Example: “This”, “This is”, “This is because”, being common groupings in a text such as a report explaining some sort of conclusions drawn.

Stop Words Remover

from pyspark.ml.feature import StopWordsRemover

remover = StopWordsRemover(inputCol=”raw”, outputCol=”filtered”)

remover\_frame = remover.transform(dataframe\_name)

TF-IDF - Convert Text to Numerical Format

***CountVectorizer*** – Index words across all documents and return a vector of word counts in descending format

***HashingTF*** – Converts words to numeric IDs, identical words have the same ID and the IDs are mapped to an index and counted and returned in a vector.

**HashingTF Format** - HashingTF function with parameters: input column, output column, numered features (number of buckets for split words) 🡪

hashing = HashingTF(inputCol=”filtered”, outputCol=”hashedValues”, numFeatures=pow(2,18) )

hashed\_df = hashing.transform(removed\_frame)

hashed\_df.show(truncate=False)

Overview of Key Steps: Create all features for a data set

🡪 pos\_neg\_to\_num = StringIndexer(inputCol=”class”, outputCol=”label”)

🡪 tokenizer = Tokenizer(inputCol=”text”, outputCol=”token\_text”)

🡪 stopremove = StopWordsRemover(inputCol=”token\_text”, outputCol = “stop\_tokens”)

🡪 hashingTF = HashingTF(inputCol = “stop\_tokens”, outputCol = “hash\_token”)

🡪 idf = IDF(inputCol=”hash\_token”, outputCol = “idf\_token”)

Setting Up the Machine Learning Section

*Run The Pipeline*

from pyspark.ml import Pipeline

data\_prep\_pipeline = Pipeline(stages=[pos\_neg\_to\_num, tokenizer, stopremove, hashingTF, idf, clean\_up] )

*Fit and Transform the Pipeline*

cleaner = data\_prep\_pipeline.fit(data\_df)

cleaned = cleaner.transform(data\_df)

*Create a Training Set and Testing Set* – Creating a split so 70% will be used for training and 30% will be used for testing. The final parameter, 21, is the seed and is arbitrary - only needing to be the same value each time this particular train and test instance is used and needed for reproducible results

training, testing = cleaned.randomSplit([0.7, 0.3], 21)

Naïve Bayes Model for Machine Learning – Runs the data and provides output columns all the way to the right in the dataframe, with “prediction” indicating positive to negative reviews.

**Note**: 1.0 = negative and 0.0 = positive

from pyspark.ml.classification import NaiveBayes

nb = NaiveBayes( )

predictor = nb.fit(training

test\_results = predictor.transform(testing)

test\_results.show(5)

Binary Classification Evaluator – Used for gathering the accuracy of a binary determination, for instance: the bias conclusion relating to reviews and existence of a negative or positive trend

from pyspark.ml.evaluation import BinaryClassificationEvaluator

acc\_eval = BinaryClassificationEvaluator(labelCol = “label”, rawPredictionCol = ‘prediction’)

acc = acc\_eval.evaluate(test\_results)

print(“Accuracy of model at predicting reviews was: %f” % acc)