Whizlabs - ML Specialty Exam Course - Data Engineering - part 2

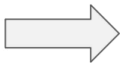
Text Feature Engineering

- ❏ Transform text within data so machine learning algorithms can better analyze it
- ❏ Splitting text into smaller pieces
- ❏ Used for text analysis of documents, streamed dialog, etc.
- ❏ Can use in a pipeline as steps in a machine learning analysis

Bag-of-Words with CountVectorizer()

- ❏ Bag-of-Words
  - ❏ Tokenizes raw text and creates a statistical representation of the text
  - ❏ Breaks up text by whitespace into single words

```
from sklearn.feature_extraction.text import CountVectorizer
# Some text
short_text = ["The quick brown fox jumped over the lazy dog."]
# Create the vectorizer transformer
vectorizer = CountVectorizer()
# Tokenize and create the vocabulary
vectorizer.fit(short_text)
# Show the vectorized vocabulary
print(vectorizer.vocabulary_)
# Encode the text
vector = vectorizer.transform(short_text)
```



```
# Show the encoded vector
print(vector.shape)
print(type(vector))
print(vector.toarray())

{'the': 7, 'quick': 6, 'brown': 0, 'fox': 2, 'jumped': 3, 'over': 5, 'lazy': 4, 'dog': 1}
(1, 8)
<class 'scipy.sparse.csr.csr_matrix'>
[[1 1 1 1 1 1 1 2]]
```

N-Gram (extension of Bag-of-Words)

=> breaks up sentences into segments (group of N words)

- ❏ N-Gram
  - ❏ Extension of Bag-of-Words which produces groups of words of n size
  - ❏ Breaks up text by whitespace into groups of words

```
#
# N-Gram example
#
# Create the vectorizer transformer
ngram_vectorizer = CountVectorizer(analyzer='word', ngram_range=(1, 3))
# Tokenize and create the vocabulary
counts = ngram_vectorizer.fit_transform(['The silly puppy ran all around the yard and the bit his owner!'])
```

```
# Check the results
print(ngram_vectorizer.get_feature_names())

['all', 'all around', 'all around the', 'and', 'and the', 'and the bit', 'around', 'around the', 'around the yard', 'bit'
```

OSB - Orthogonal Sparse Bigram

Creates group of words with always the first word or letter in

- Orthogonal Sparse Bigram
  - Creates groups of words of size n, returns every pair of words that includes the first word
  - Creates groups of words that always include the first word

TF-IDF

Helps find important words by filtering out common words

- Term Frequency-Inverse Document Frequency (tf-idf)
  - Shows how important a word or words are to a given set of text by providing appropriate weights to terms that are common and less common
  - Show the popularity of a word or words in text data by making common words like “the” or “and” less important

Use Cases

Use Case	Transformation	Reason
Finding phrases in spam	N-Gram	Compare whole phrases such as "you're a winner!" or "Buy now!"
Finding subject of several PDFs	tf-idf Orthogonal Sparse Bigram	Filter less important words in the documents. Find common word combinations repeated in the documents.

N-Gram allow us to compare whole phrases

TF-IDF => filter of less important words
Then pas results to OSB to find common words combination repeated in the document
=> find subject areas

Text Feature Editing Lab

Bag-of-Worlds example with **CountVectorizer()**

In [1]:
```
from sklearn.feature_extraction.text import CountVectorizer
#
# Bag-of-Words example
#
# Some text
short_text = ["The aggressive brown beast exasperated the pedantic pontificator."]
# Create the vectorizer transformer
vectorizer = CountVectorizer()
# Tokenize and create the vocabulary
vectorizer.fit(short_text)
# Show the vectorized vocabulary
print(vectorizer.vocabulary_)
# Encode the text
vector = vectorizer.transform(short_text)
# Show the encoded vector
print(vector.shape)
print(type(vector))
print(vector.toarray())
```

```
{'the': 6, 'aggressive': 0, 'brown': 2, 'beast': 1, 'exasperated': 3, 'pedantic': 4, 'pontificator': 5}
(1, 7)
<class 'scipy.sparse.csr.csr_matrix'>
[[1 1 1 1 1 1 2]]
```

Sentence broken down into words - each word has a value
Shape = 1 sentence with 7 elements
Encoded vector: shows “the” twice and other words once

N-Gram example with **CountVectorizer()**

In [2]:
```
#
# N-Gram example
#
# Create the vectorizer transformer
ngram_vectorizer = CountVectorizer(analyzer='word', ngram_range=(1, 3))
# Tokenize and create the vocabulary
counts = ngram_vectorizer.fit_transform(['The silly puppy ran all around the yard and the bit his owner!'])
# Check the results
print(ngram_vectorizer.get_feature_names())
```

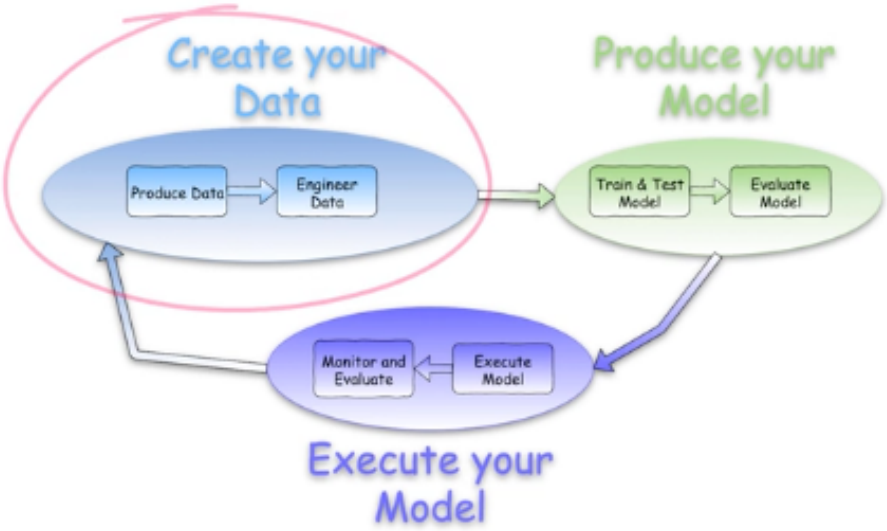
```
['all', 'all around', 'all around the', 'and', 'and the', 'and the bit', 'around', 'around the', 'around the ya
rd', 'bit', 'bit his', 'bit his owner', 'his', 'his owner', 'owner', 'puppy', 'puppy ran', 'puppy ran all', 'ra
n', 'ran all', 'ran all around', 'silly', 'silly puppy', 'silly puppy ran', 'the', 'the bit', 'the bit his', 't
he silly', 'the silly puppy', 'the yard', 'the yard and', 'yard', 'yard and', 'yard and the']
```

N-gram broke up our text by words, with combination of 1 to 3 words

AWS Migration services and tools

AWS Machine Learning - Migration Services

The Machine Learning Cycle - Gathering Data



AWS Machine Learning - Migration Services

AWS Migration Services

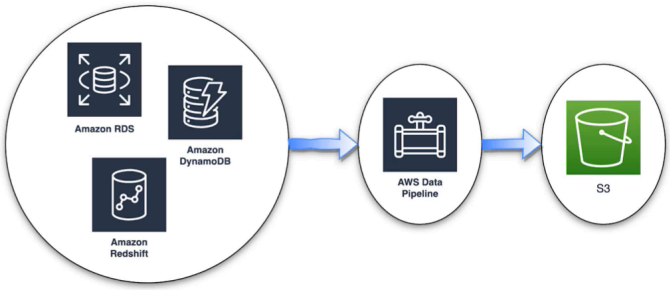
- ❏ Migrate data from source to machine learning repository
- ❏ Several AWS services to help move data
  - ❏ Amazon Data Pipeline
  - ❏ AWS Database Migration Service (DMS)
  - ❏ AWS Glue
  - ❏ Amazon SageMaker
  - ❏ Amazon Athena

Amazon Data Pipeline

- using **Pipeline Activities** we can **COPY** data
- we can **SCHEDULE** it

Can pull data fin RDS, Amazon DB,... into S3

Amazon Data Pipeline

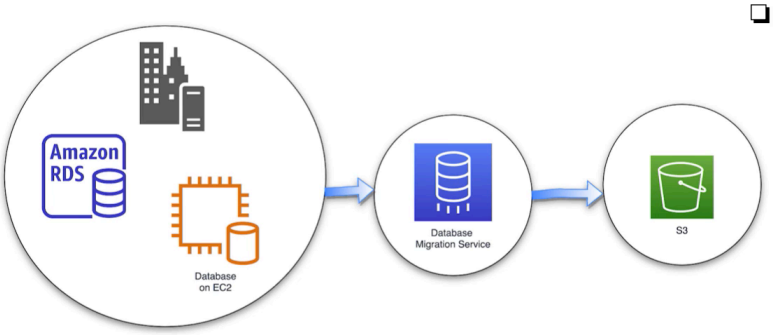


- ❏ Copy data using Pipeline Activities
- ❏ Schedule regular data movement and data processing activities
- ❏ Integrates with on-premise and cloud-based storage systems
- ❏ Use your data where you want it and in the format you choose

DMS - DataBase Migration Service

Migrate DB to DB, or DB to S3

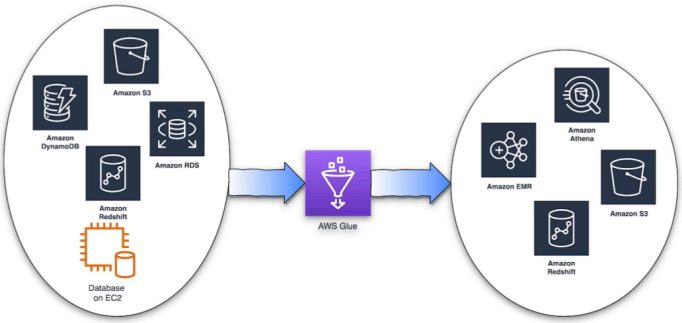
AWS DMS



- ❏ Move data between databases
  - ❏ MySQL to MySQL
  - ❏ Aurora to DynamoDB

AWS Glue

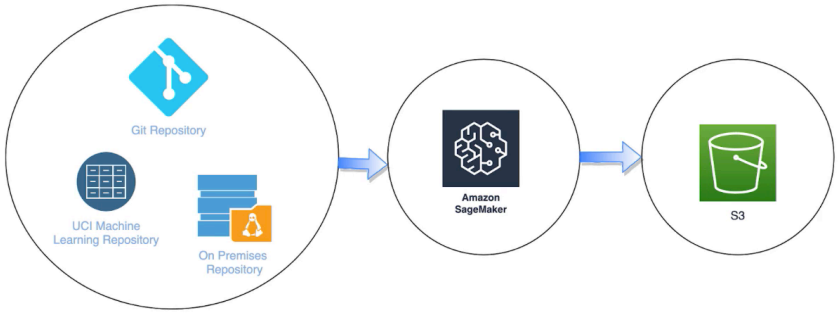
AWS Glue



- ❏ Extract, Transform, and Load (ETL)
  - ❏ Determine data type and schema
- ❏ Can run your data engineering algorithms
  - ❏ Feature Selection
  - ❏ Data Cleansing
- ❏ Can run on demand, on a schedule, or on events

SageMaker

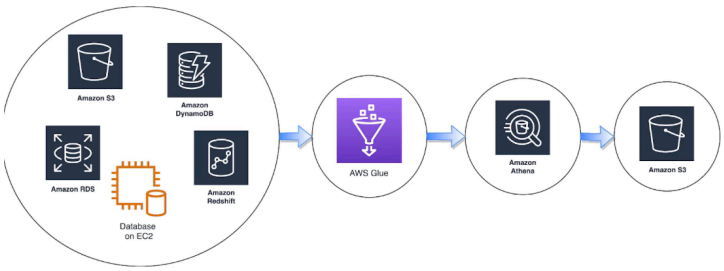
Amazon SageMaker



- ❏ Use jupyter notebooks
  - ❏ Scikit-Learn
  - ❏ Pandas

Athena

Amazon Athena



- ❏ Run SQL queries on S3 data
- ❏ Needs a data catalog such as the one created by Glue
- ❏ SQL transform your data in preparation for use in ML models

Use Cases

All use cases involve is moving data to S3
1/ Move data from **EMR** to S3
=> **Data Pipeline**: has activities to move data from one source to another, like EMR

2/ Move Fata from **DynamoFB** to S3
=> **AWS Glue**: use the BynamoDB **classifier**.

3/ Move data from **Redshift**
=> **DataPipeline**: **UNLOAD** data from Redshift. Or we could use **Glue** to crawl Redshift data, find the schema and move it to S3

4/ Move from **on-prem** DB
=> **DMS**

Use Cases

- ❏ Move data to S3 for your machine learning model
  - ❏ Move data from EMR cluster
  - ❏ Move data from DynamoDB
  - ❏ Move data from Redshift
  - ❏ Move data from on-prem database

Data Engineering Exam Tips

AWS Machine Learning - Data Engineering Exam Tips

Seven steps of data preparation

- Seven steps to prepare you data for use in a machine learning model
  1) Gather your data
  2) Handle missing data
  3) Feature extraction
  4) Decide which features are important
  5) Encode categorical values
  6) Numeric feature engineering
  7) Split your data into training and testing datasets

Gather Data

- Several AWS services to help gather data
  - Amazon Data Pipeline
  - AWS Database Migration Service (DMS)
  - AWS Glue
  - Amazon SageMaker
  - Amazon Athena

Missing Data

- Null value replacement - Several approaches to the problem of handling missing data
  - Do nothing
  - Remove the entire record
  - Mode/median/average value replacement
  - Most frequent value
  - Model-based imputation
    - K-Nearest Neighbors
    - Regression
    - Deep Learning
  - Interpolation / Extrapolation
  - Forward filling / Backward filling
  - Hot deck imputation

Case	Attributes			Decision
	Temperature	Headache	Nausea	Flu
1	high	?	yes	yes
2	very_high	yes	no	yes
3	?	no	no	no
4	normal	yes	?	no
5	?	yes	yes	yes

Feature Extraction

- Creating new features from your existing features, feature extraction creates a new, smaller set of features that stills captures most of the useful information

	sepal length	sepal width	petal length	petal width	target
0	-0.900681	1.032057	-1.341272	-1.312977	Iris-setosa
1	-1.143017	-0.124958	-1.341272	-1.312977	Iris-setosa
2	-1.385353	0.337848	-1.398138	-1.312977	Iris-setosa
3	-1.506521	0.106445	-1.284407	-1.312977	Iris-setosa
4	-1.021849	1.263460	-1.341272	-1.312977	Iris-setosa
...	...	...	...	...	...
145	1.038005	-0.124958	0.819624	1.447956	Iris-virginica
146	0.553333	-1.281972	0.705893	0.922064	Iris-virginica
147	0.795669	-0.124958	0.819624	1.053537	Iris-virginica
148	0.432165	0.800654	0.933356	1.447956	Iris-virginica
149	0.068662	-0.124958	0.762759	0.790591	Iris-virginica

150 rows × 5 columns

	principal component 1	principal component 2	target
0	-2.264542	0.505704	Iris-setosa
1	-2.086426	-0.655405	Iris-setosa
2	-2.367950	-0.318477	Iris-setosa
3	-2.304197	-0.575368	Iris-setosa
4	-2.388777	0.674767	Iris-setosa
...	...	...	...
145	1.870522	0.382822	Iris-virginica
146	1.558492	-0.905314	Iris-virginica
147	1.520845	0.266795	Iris-virginica
148	1.376391	1.016362	Iris-virginica
149	0.959299	-0.022284	Iris-virginica

150 rows × 3 columns

Feature Selection

- Use feature selection to filter irrelevant or redundant features from your dataset
  - Feature selection requires normalization

	Column1	Column2	Column3
0	0	2	0
1	0	1	4
2	0	1	1

	Column1	Column2	Column3
0	0.0	0.554700	0.000000
1	0.0	0.196116	0.784465
2	0.0	0.301511	0.301511

  - Feature selection removes features from your dataset - **Variance Thresholds**

	Column1	Column2	Column3
0	0.0	0.554700	0.000000
1	0.0	0.196116	0.784465
2	0.0	0.301511	0.301511

	Column2	Column3
0	0.554700	0.000000
1	0.196116	0.784465
2	0.301511	0.301511

Encode categorical values

- Several approaches
  - Binarizer Encoding: for features of a binary nature
  - Label Encoding: may imply ordinality, can use Ordinal Encoder
  - One-hot-encoding: Change nominal categorical values such as ‘true’, ‘false’, or ‘rainy’, ‘sunny’ to numerical values

	Weather	...
0	sunny	...
1	cloudy	...
2	cloudy	...
3	sunny	...

one-hot encoding

	is_sunny	is_cloudy	...
0	1	0	...
1	0	1	...
2	0	1	...
3	1	0	...

Numerical feature engineering

- Transform numeric values so machine learning algorithms can better analyze them
- Change numeric values so all values are on the same scale
  - Normalization: rescales the values into a range of [0,1]
  - Standardization: rescales data to have a mean of 0 and a standard deviation of 1 (unit variance)
- Binning

Split data into training and testing datasets

- Split your data with a scikit-learn library
  - Usually approximately an 80/20 split

```
from sklearn.datasets import fetch_mldata
mnist = fetch_mldata('MNIST original')
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.decomposition import PCA
from sklearn.linear_model import LogisticRegression
import time

# split data between train and test
train_img, test_img, train_lbl, test_lbl = train_test_split( mnist.data, mnist.target, test_size=1/7.0, random_state=0)
scaler = StandardScaler()
# Fit training set only.
scaler.fit(train_img)

# Transform training set and test sets
train_img = scaler.transform(train_img)
test_img = scaler.transform(test_img)
```