Microsoft

3 praktische Nutzungsszenarien mit Azure Databricks

Meistern Sie Ihre Big Data- und KI-Herausforderungen

Azure Databricks

Was Sie in diesem E-Book lernen und weshalb

Azure Databricks ist eine schnelle, einfache und kollaborative Apache® Spark™-basierte Analyseplattform. Sie bietet Einrichtung mit einem Mausklick, optimierte Workflows und die Skalierbarkeit und Sicherheit von Microsoft Azure.

In diesem E-Book soll nicht einfach nur die Funktionsweise von Azure Databricks erklärt werden. Sie finden hier vielmehr drei Szenarien, in denen Azure Databricks Datenanalysten bei bestimmen Aufgaben unterstützt, und eine Beschreibung der Ergebnisse. Wir behandeln die folgenden Themen:

- Ein Modell zur Fluktuationsanalyse
- Ein Modul für Filmempfehlungen
- Eine Demo zur Angriffserkennung

Erklärungen zu Notebooks

Notebooks in Azure Databricks sind interaktive Arbeitsbereiche, die von Benutzern gemeinsam in mehreren Bereichen zum Erkunden und Visualisieren genutzt werden können. Mit Notebooks können Sie Daten in großem Maßstab untersuchen, Modelle erstellen und trainieren und ihre Ergebnisse freigeben. Dabei erzielen Sie von Tests bis hin zur Produktion eine schnelle und effektive Iteration und Zusammenarbeit. In diesem E-Book zeigen wir Ihnen anhand von Beispielcode und Ergebnissen, wie Sie Notebooks nutzen können und wie Azure Databricks funktioniert.

Für wen ist dieses Buch gedacht?

Dieses E-Book ist in erster Linie für Datenanalysten gedacht, es enthält aber auch hilfreiche Informationen für Datentechniker und Business-Anwender mit Interesse an der Erstellung, Bereitstellung und Visualisierung von Datenmodellen.

Inhalt

Einstieg

Die Demos in diesem E-Book zeigen, wie Azure Databricks-Notebooks Teams bei der Analyse und Lösung von Problemen unterstützen. Lesen Sie die detaillierten Demos durch, oder testen Sie Azure Databricks selbst, indem Sie sich **für ein kostenloses Konto anmelden**.

Wenn Sie die Notebooks testen möchten, nachdem Sie Ihr kostenloses Konto eingerichtet haben, verwenden Sie für jedes Notebook die folgenden Anweisungen für die erste Einrichtung.

Nachdem Sie Azure Databricks im Azure-Portal ausgewählt haben, können Sie es ausführen, indem Sie einen Cluster erstellen. Zum Ausführen dieser Notebooks können Sie alle Standardeinstellungen in Azure Databricks für die Clustererstellung übernehmen. Führen Sie dafür die folgenden Schritte aus:

1. Klicken Sie auf das Clustersymbol in der linken Leiste.
2. Wählen Sie „Cluster erstellen“.
3. Geben Sie einen Clusternamen ein.
4. Klicken Sie auf die Schaltfläche „Cluster erstellen“.

Nun ist die Einrichtung abgeschlossen, und Sie können die Azure Databricks-Notebooks importieren.
Gehen Sie zum Importieren der Notebooks folgendermaßen vor:

1. Klicken Sie auf das Arbeitsbereichssymbol.
2. Wählen Sie in der Benutzerspalte Ihr Verzeichnis aus.
3. Klicken Sie auf das Dropdownmenü für den Import. Ziehen Sie Ihre Notebooks-Dateien in dieses Dialogfeld.
4. Klicken Sie im Notebook in der Dropdownliste auf „Getrennt“.
5. Wählen Sie den Cluster aus, den Sie im vorherigen Schritt erstellt haben.

Notebook 1

Demo einer Fluktuationsanalyse

Demo einer Fluktuationsanalyse

Kundenfluktuation bezeichnet den Verlust von Kunden. Für viele Unternehmen ist es unerlässlich, Fluktuation vorherzusehen und zu vermeiden.

In diesem Notebook verwenden wir ein vorgefertigtes Modell in Azure Databricks, um die Kundenfluktuation zu analysieren. Mit diesem Modell lässt sich mit einer Genauigkeit von 90 % vorhersagen, wann ein Kunde abwandert. So können wir einen Bericht erstellen, der Kunden anzeigt, die im Begriff sind, abzuwandern. Davon ausgehend können wir eine Strategie entwickeln, etwa in Form von Sonderangeboten, um diese Fluktuation zu verhindern. Dieses Beispiel befasst sich mit Mobilfunkanbietern. Das Ziel besteht darin, Kunden von einem Anbieterwechsel abzuhalten. Dieses Notebook:

- Enthält Funktionen, die für Datenanalysten, Datentechniker und Business-Anwender relevant sind.
- Eignet sich für einen datengesteuerten Storytelling-Ansatz, der zeigt, wie Notebooks in Azure Databricks genutzt werden können.
- Verwendet einen Machine-Learning-Algorithmus mit Gradient Boosting zur Analyse von Fluktuationsdatensätzen.
- Veranschaulicht einen einfachen Workflow für die Fluktuationsanalyse. Wir verwenden einen Fluktuationsdatensatz aus dem UCI Machine Learning Repository.

Schritt 1: Erfassen von Fluktuationsdaten in einem Notebook

Wir laden den UCI-Datensatz herunter, der auf der UCI-Site gehostet wird.

In der Metadatendatei „churn.names“ sehen wir die Bedeutung der Datenspalten:

- state: discrete.
- account length: continuous.
- area code: continuous.
- phone number: discrete.
- international plan: discrete.
- voice mail plan: discrete.
- number vmail messages: continuous.
- total day minutes: continuous.
- total day calls: continuous.
- total day charge: continuous.
- total eve minutes: continuous.
- total eve calls: continuous.
- total eve charge: continuous.
- total night minutes: continuous.
- total night calls: continuous.
- total night charge: continuous.
- total intl minutes: continuous.
- total intl calls: continuous.
- total intl charge: continuous.
- number customer service calls: continuous.
- churned: discrete <- Dies ist die Klassifizierung, die prognostiziert werden soll und die angibt, ob der Kunde abwandert.

```
%sh
mkdir /tmp/churn
wget http://www.sgi.com/tech/mlc/db/churn.data -O /tmp/churn/churn.data
wget http://www.sgi.com/tech/mlc/db/churn.test -O /tmp/churn/churn.test
--2017-08-25 19:52:36--  http://www.sgi.com/tech/mlc/db/churn.data
Resolving www.sgi.com (www.sgi.com)... 192.48.178.134
Connecting to www.sgi.com (www.sgi.com)|192.48.178.134|:80... connected.
HTTP request sent, awaiting response... 200 OK
Length: 376493 (368K) [text/plain]
Saving to: ‘/tmp/churn/churn.data’

     0K ......... ......... ......... ......... ......... 13%  131K 2s
    50K ......... ......... ......... ......... ......... 27%  650K 1s
   100K ......... ......... ......... ......... ......... 40%  336K 1s
   150K ......... ......... ......... ......... ......... 54%  672K 1s
   200K ......... ......... ......... ......... ......... 67% 57.9M 0s
   250K ......... ......... ......... ......... ......... 81%  667K 0s
   300K ......... ......... ......... ......... ......... 95% 69.0M 0s
   350K ......... .......                                  100%  166M=0.8s

2017-08-25 19:52:37 (485 KB/s) - ‘/tmp/churn/churn.data’ saved
[376493/376493]

--2017-08-25 19:52:37--  http://www.sgi.com/tech/mlc/db/churn.test
Resolving www.sgi.com (www.sgi.com)... 192.48.178.134
Connecting to www.sgi.com (www.sgi.com)|192.48.178.134|:80... connected.
HTTP request sent, awaiting response... 200 OK
Length: 188074 (184K) [text/plain]
Saving to: ‘/tmp/churn/churn.test’

     0K ......... ......... ......... ......... ......... 27%  160K 1s
    50K ......... ......... ......... ......... ......... 54%  329K 0s
   100K ......... ......... ......... ......... ......... 81%  665K 0s
   150K ......... ......... ......... ...                 100% 23.2M=0.5s

2017-08-25 19:52:37 (340 KB/s) - ‘/tmp/churn/churn.test’ saved
[188074/188074]
```

Stellen Sie die Daten lokal bereit.

```
%py
dbutils.fs.mkdirs("/mnt/churn")
dbutils.fs.mv("file:///tmp/churn/churn.data", "/mnt/churn/churn.data")
dbutils.fs.mv("file:///tmp/churn/churn.test", "/mnt/churn/churn.test")

Out[2]: True
```

```
%fs ls /mnt/churn
```

Pfad	Name	Größe
dbfs:/mnt/churn/churn.data	churn.data	376493
dbfs:/mnt/churn/churn.test	churn.test	188074

Der zweite Schritt besteht darin, das Schema im Datenrahmen zu erstellen.

```
from pyspark.sql.types import *

# Der zweite Schritt besteht darin, das Schema zu erstellen.
schema =StructType([
    StructField("state",StringType(), False),
    StructField("account_length",DoubleType(), False),
    StructField("area_code",DoubleType(), False),
    StructField("phone_number",StringType(), False),
    StructField("international_plan",StringType(), False),
    StructField("voice_mail_plan",StringType(), False),
    StructField("number_vmail_messages",DoubleType(), False),
    StructField("total_day_minutes",DoubleType(), False),
    StructField("total_day_calls",DoubleType(), False),
    StructField("total_day_charge",DoubleType(), False),
    StructField("total_eve_minutes",DoubleType(), False),
    StructField("total_eve_calls",DoubleType(), False),
    StructField("total_eve_charge",DoubleType(), False),
    StructField("total_night_minutes",DoubleType(), False),
    StructField("total_night_calls",DoubleType(), False),
    StructField("total_night_charge",DoubleType(), False),
    StructField("total_intl_minutes",DoubleType(), False),
    StructField("total_intl_calls",DoubleType(), False),
    StructField("total_intl_charge",DoubleType(), False),
    StructField("number_customer_service_calls",DoubleType(), False),
    StructField("churned",StringType(), False)
])

df = (spark.read.option("delimiter", ",")
  .option("inferSchema", "true")
  .schema(schema)
  .csv("dbfs:/mnt/churn/churn.data"))
```

```
display(df)
```

state	account_length	area_code	phone_number	international_plan	voice_mail_plan	number_vmail_messages	total_day_minutes	total_day_calls	total_day_charge	tot
KS	128	415	382-4657	no	yes	25	265.1	110	45.07	197
OH	107	415	371-7191	no	yes	26	161.6	123	27.47	195
NJ	137	415	358-1921	no	no	0	243.4	114	41.38	121
OH	84	408	375-999	yes	no	0	299.4	71	50.9	61.9
OK	75	415	330-6626	yes	no	0	166.7	113	28.34	148
AL	118	510	391-8027	yes	no	0	223.4	98	37.98	220
MA	121	510	355-9993	no	yes	24	218.2	88	37.09	348
MO	147	415	329-9001	yes	no	0	157	79	26.69	103
LA	117	408	335-4719	no	no	0	184.5	97	31.37	351
WV	141	415	330-8173	yes	yes	37	258.6	84	43.96	222
IN	65	415	329-6603	no	no	0	129.1	137	21.95	228
RI	74	415	344-9403	no	no	0	187.7	127	31.91	5

Schritt 2: Ergänzen Sie die Daten, um zusätzliche Einblicke in den Fluktuationsdatensatz zu erhalten.

Wir zählen die Anzahl von Datenpunkten und trennen abgewanderte und nicht abgewanderte Kunden.

```
# Da wir es später benötigen werden …
from pyspark.sql.functions import *
from pyspark.sql.types import *
```

Wir führen einen Filter-und Zählvorgang durch, um die Anzahl von Kunden zu ermitteln, die abgewandert sind.

```
numCases = df.count()
numChurned = df.filter(col("churned") == ' True.').count()
```

```
numCases = numCases
numChurned = numChurned
numUnchurned = numCases - numChurned
print("Total Number of cases: {0:,}".format( numCases ))
print("Total Number of cases churned: {0:,}".format( numChurned ))
print("Total Number of cases unchurned: {0:,}".format( numUnchurned ))
Total Number of cases: 3,333
Total Number of cases churned: 483
Total Number of cases unchurned: 2,850
```

Die Daten werden in eine PARQUET-Datei umgewandelt. Dieses Datenformat eignet sich gut für die Analyse großer Datenmengen.

```
df.repartition(1).write.parquet('/mnt/databricks-wesley/demo-data/
insurance/churndata')
```

Schritt 3: Durchsuchen von Fluktuationsdaten

Wir erstellen aus den PARQUET-Daten eine Tabelle, damit wir diese mit Spark SQL im Maßstab analysieren können.

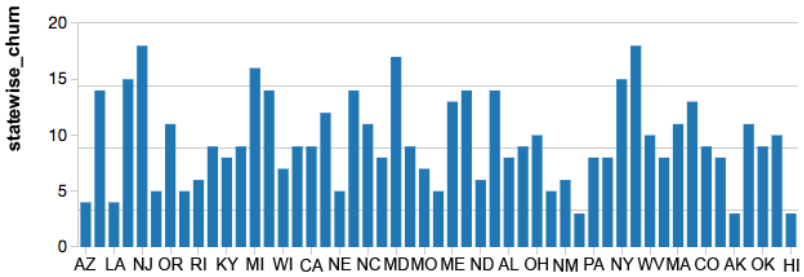
```
%sql

    Drop table temp_idsdata;

    CREATE TEMPORARY TABLE temp_idsdata
    USING parquet
    OPTIONS (
      path "/mnt/databricks-wesley/demo-data/insurance/churndata"
    )
OK
```

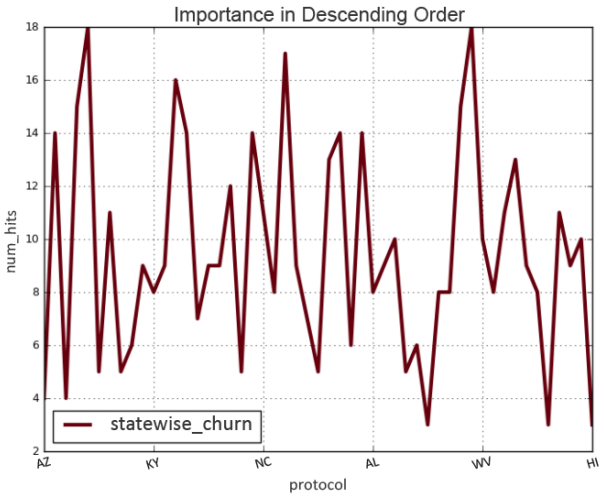
Fluktuation nach Bundesstaat mithilfe von databricks graph

```
%sql
SELECT state, count(*) as statewise_churn FROM temp_idsdata where
churned= " True." group by state
```



Fluktuation nach Bundesstaat mithilfe von python matplotlib

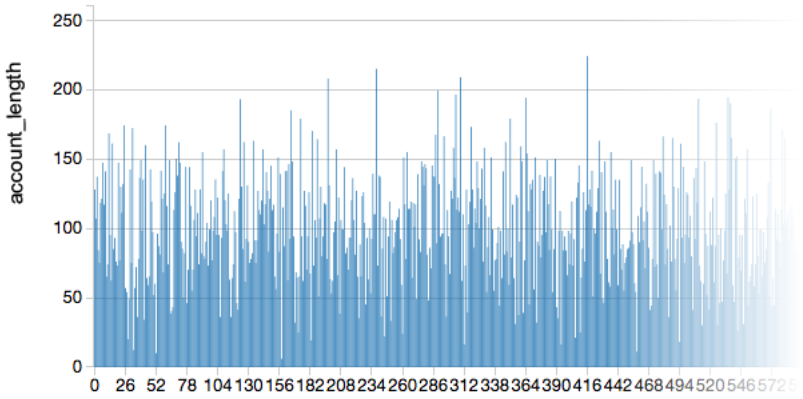
```
import matplotlib.pyplot as plt
importance = sqlContext.sql("SELECT state, count(*) as statewise_churn
FROM temp_idsdata where churned= ' True.' group by state")
importanceDF = importance.toPandas()
ax = importanceDF.plot(x="state", y="statewise_
churn",lw=3,colormap='Reds_r',title='Importance in Descending Order',
fontsize=9)
ax.set_xlabel("protocol")
ax.set_ylabel("num_hits")
plt.xticks(rotation=12)
plt.grid(True)
plt.show()
display()
```



Schritt 4: Visualisierung

Zeigt die Verteilung der Kontolänge an.

```
display(df.select("account_length").orderBy(id))
```




```
df.printSchema()

root
 |-- state: string (nullable = true)
 |-- account_length: double (nullable = true)
 |-- area_code: double (nullable = true)
 |-- phone_number: string (nullable = true)
 |-- international_plan: string (nullable = true)
 |-- voice_mail_plan: string (nullable = true)
 |-- number_vmail_messages: double (nullable = true)
 |-- total_day_minutes: double (nullable = true)
 |-- total_day_calls: double (nullable = true)
 |-- total_day_charge: double (nullable = true)
 |-- total_eve_minutes: double (nullable = true)
 |-- total_eve_calls: double (nullable = true)
 |-- total_eve_charge: double (nullable = true)
 |-- total_night_minutes: double (nullable = true)
 |-- total_night_calls: double (nullable = true)
 |-- total_night_charge: double (nullable = true)
 |-- total_intl_minutes: double (nullable = true)
 |-- total_intl_calls: double (nullable = true)
 |-- total_intl_charge: double (nullable = true)
 |-- number_customer_service_calls: double (nullable = true)
 |-- churned: string (nullable = true)
```

Schritt 5: Modellerstellung

Erstellen Sie eine Tabelle.

Modellanpassung und Zusammenfassung

```
from  pyspark.ml.feature import StringIndexer

indexer1 = (StringIndexer()
                    .setInputCol("churned")
                    .setOutputCol("churnedIndex")
                    .fit(df))
```

Erstellen Sie ein Array der Daten.

```
indexed1 = indexer1.transform(df)
finaldf = indexed1.withColumn("censor", lit(1))

from pyspark.ml.feature import VectorAssembler
vecAssembler = VectorAssembler()
vecAssembler.setInputCols(["account_length", "total_day_calls",  "total_eve_calls", "total_night_calls", "total_intl_calls",  "number_customer_service_calls"])
vecAssembler.setOutputCol("features")
print vecAssembler.explainParams()

from pyspark.ml.classification import GBTClassifier

aft = GBTClassifier()
aft.setLabelCol("churnedIndex")

print aft.explainParams()
inputCols: input column names. (current: ['account_length', 'total_day_calls', 'total_eve_calls', 'total_night_calls', 'total_intl_calls', 'number_customer_service_calls'])
outputCol: output column name. (default: VectorAssembler_402dae9a2a13c5e1ea7f__output, current: features)
cacheNodeIds: If false, the algorithm will pass trees to executors to match instances with nodes. If true, the algorithm will cache node IDs for each instance.
Caching can speed up training of deeper trees. Users can set how often should the cache be checkpointed or disable it by setting checkpointInterval. (default: False)
checkpointInterval: set checkpoint interval (>= 1) or disable checkpoint (-1). E.g. 10 means that the cache will get checkpointed every 10 iterations. (default: 10)
```

```
featuresCol: features column name. (default: features)
labelCol: label column name. (default: label, current: churnedIndex)
lossType: Loss function which GBT tries to minimize (case-insensitive). Supported options: logistic (default: logistic)
maxBins: Max number of bins for discretizing continuous features.  Must be >=2 and >= number of categories for any categorical feature. (default: 32)
maxDepth: Maximum depth of the tree. (>= 0) E.g., depth 0 means 1 leaf node; depth 1 means 1 internal node + 2 leaf nodes. (default: 5)
maxIter: max number of iterations (>= 0). (default: 20)
maxMemoryInMB: Maximum memory in MB allocated to histogram aggregation. If too small, then 1 node will be split per iteration, and its aggregates may
exceed this size. (default: 256)
minInfoGain: Minimum information gain for a split to be considered at a tree node. (default: 0.0)
minInstancesPerNode: Minimum number of instances each child must have after split. If a split causes the left or right child to have fewer than
minInstancesPerNode, the split will be discarded as invalid. Should be >= 1. (default: 1)
predictionCol: prediction column name. (default: prediction)
seed: random seed. (default: 2857134701650851239)
stepSize: Step size to be used for each iteration of optimization (>= 0). (default: 0.1)
subsamplingRate: Fraction of the training data used for learning each decision tree, in range (0, 1]. (default: 1.0)
```

Erstellen eines Modells aus Trainingsdaten

```
from pyspark.ml import Pipeline

# Wir verwenden die neue spark.ml pipeline-API. Wenn Sie bereits mit scikit-learn gearbeitet haben, wird Ihnen das bekannt vorkommen.
lrPipeline = Pipeline()

# Jetzt weisen wir die Pipeline an, zuerst den Funktionsvektor zu erstellen und dann die lineare Regression auszuführen.
lrPipeline.setStages([vecAssembler, aft])

# Pipelines sind Kalkulatoren, und wir rufen „fit“ auf, um sie zu verwenden:
lrPipelineModel = lrPipeline.fit(finaldf)
```

Verwenden eines Modells für die Datenprognose

```
predictionsAndLabelsDF = lrPipelineModel.transform(finaldf)
confusionMatrix = predictionsAndLabelsDF.select('churnedIndex', 'prediction')
```

Wahrheitsmatrix für das Fluktuationsmodell

```
from pyspark.mllib.evaluation import MulticlassMetrics
metrics = MulticlassMetrics(confusionMatrix.rdd)
cm = metrics.confusionMatrix().toArray()
```

Leistungskennzahlen des Modells

```
print metrics.falsePositiveRate(0.0)
print metrics.accuracy
```

```
0.0514705882353
0.891689168917
```

Interpretation der Ergebnisse

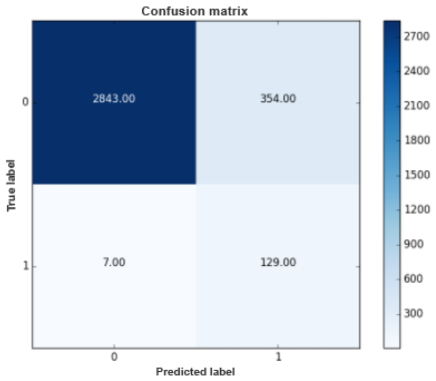
Die Abbildung auf der rechten Seite zeigt den Index, der zum Messen der einzelnen Fluktuationstypen verwendet wird.

Der Fluktuationsindex, bei dem der Gradient-Boosting-Algorithmus zum Einsatz kommt, hat eine Genauigkeit von fast 89 %.

Wahrheitsmatrix in matplotlib

```
%python
import matplotlib.pyplot as plt
import numpy as np
import itertools
plt.figure()
classes=list([0,1])
plt.imshow(cm, interpolation='nearest', cmap=plt.cm.Blues)
plt.title('Confusion matrix')
plt.colorbar()
tick_marks = np.arange(len(classes))
plt.xticks(tick_marks, classes, rotation=0)
plt.yticks(tick_marks, classes)

fmt =  '.2f'
thresh = cm.max() / 2.
for i, j in itertools.product(range(cm.shape[0]), range(cm.shape[1])):
    plt.text(j, i, format(cm[i, j], fmt),
             horizontalalignment="center",
             color="white" if cm[i, j] > thresh else "black")
plt.tight_layout()
plt.ylabel('True label')
plt.xlabel('Predicted label')
plt.show()
display()
```



Notebook 2

Modul für Filmempfehlungen

Modul für Filmempfehlungen

Empfehlungsmodule kommen in vielen Branchen zum Einsatz, extern auf Einzelhandelsseiten oder auch intern auf Mitarbeiterseiten. Ein Empfehlungsmodul stellt Pläne für Anwender bereit, die auf Punkten basieren, die für diese von Bedeutung sind.

Diese Demonstration ist ein einfaches Beispiel eines Verbrauchers, der eine Filmwebsite verwendet, um einen Film auszuwählen. Empfehlungsmodule untersuchen historische Daten dazu, was Nutzer ausgewählt haben, und erstellen anschließend eine Prognose der Auswahl, die der Benutzer wahrscheinlich trifft. Dieses Notebook für ein Filmempfehlungsmodul:

- Basiert auf der Azure Databricks-Plattform und verwendet einen **Machine-Learning-ALS-Empfehlungsalgorithmus** zum Erstellen von Empfehlungen für Filme.
- Demonstriert einen Analyseworkflow für Filmempfehlungen mithilfe von Filmdaten aus dem Kaggle-Datensatz.
- Stellt einen Ort zum Erstellen der gesamten Analyseanwendung bereit, damit Benutzer mit anderen Teilnehmern zusammenarbeiten können.
- Ermöglicht es Benutzern, die fortlaufende Genauigkeit anzuzeigen, damit Verbesserungen möglich sind.

Schritt 1: Erfassen von Filmdaten in einem Notebook

Wir extrahieren den Filmdatensatz, der auf Kaggle gehostet wird.

Wählen Sie aus den am besten bewerteten Filmen zehn aus, da diese wahrscheinlich am bekanntesten sind. Erstellen Sie Azure Databricks-Widgets, damit Benutzer Bewertungen für diese Filme abgeben können.

```
sqlContext.sql("""
    select
      movie_id, movies.name, count(*) as times_rated
    from
      ratings
    join
      movies on ratings.movie_id = movies.id
    group by
      movie_id, movies.name, movies.year
    order by
      times_rated desc
    limit
      200
    """
).registerTempTable("most_rated_movies")
```

```
if not "most_rated_movies" in vars():
  most_rated_movies = sqlContext.table("most_rated_movies").rdd.
takeSample(True, 10)
  for i in range(0, len(most_rated_movies)):
    dbutils.widgets.dropdown("movie_%i" % i, "5", ["1", "2", "3", "4", "5"],
most_rated_movies[i].name)
```

Ändern Sie die Werte oben in Ihre persönlichen Bewertungen, bevor Sie fortfahren.

```
from datetime import datetime
from pyspark.sql import Row
ratings = []
for i in range(0, len(most_rated_movies)):
  ratings.append(
    Row(user_id = 0,
        movie_id = most_rated_movies[i].movie_id,
        rating = float(dbutils.widgets.get("movie_%i" %i))))
myRatingsDF = sqlContext.createDataFrame(ratings)
```

Schritt 2: Ergänzen der Daten und Vorbereiten der Modellierung

```
%sql select min(user_id) from ratings
```

min(user_id)
1

```
from pyspark.sql import functions

ratings = sqlContext.table("ratings")
ratings = ratings.withColumn("rating", ratings.rating.cast("float"))
```

```
(training, test) = ratings.randomSplit([0.8, 0.2])
```

Schritt 3: Modellerstellung

Passen Sie ein ALS-Modell an die Bewertungstabelle an.

```
from pyspark.ml.recommendation import ALS

als = ALS(maxIter=5, regParam=0.01, userCol="user_id", itemCol="movie_id",
ratingCol="rating")
model = als.fit(training.unionAll(myRatingsDF))
```

Schritt 4: Modellauswertung

Werten Sie das Modell aus, indem Sie den Root Mean Square-Fehler auf dem Testsatz berechnen.

```
predictions = model.transform(test).dropna()
predictions.registerTempTable("predictions")
```

```
%sql select user_id, movie_id, rating, prediction from predictions
```

user_id	movie_id	rating	prediction
4227	148	2	2.6070688
1242	148	3	2.5327067
1069	148	2	3.8583977
2507	148	4	3.8449543
53	148	5	3.940087
216	148	2	2.2447278
2456	148	2	3.5586698
4169	463	2	2.7173512
4040	463	1	2.1891994

```
from pyspark.ml.evaluation import RegressionEvaluator

evaluator = RegressionEvaluator(metricName="rmse", labelCol="rating",
predictionCol="prediction")
```

```
rmse = evaluator.evaluate(predictions)
```

```
displayHTML("<h4>The Root-mean-square error is %s</h4>" % str(rmse))
```

The Root-mean-square error is 0.897631907219

Schritt 5: Modelltest

Testen Sie die Erstellung von Prognosen durch das Modell.

```
mySampledMovies = model.transform(myRatingsDF)
mySampledMovies.registerTempTable("mySampledMovies")
```

```
display(sqlContext.sql("select user_id, movie_id, rating, prediction
from mySampledMovies"))
```

user_id	movie_id	rating	prediction
0		5	NaN
0		5	NaN
0		4	NaN
0		5	NaN
0		4	NaN
0		5	NaN
0		3	NaN

```
my_rmse = evaluator.evaluate(mySampledMovies)
```

```
displayHTML("<h4>My Root-mean-square error is %s</h4>" % str(my_rmse))
```

My Root-mean-square error is 0.418569215012

```
from pyspark.sql import functions
df = sqlContext.table("movies")
myGeneratedPredictions = model.transform(df.select(df.id.alias(
"movie_id")).withColumn("user_id", functions.expr("int('0')")))
myGeneratedPredictions.dropna().registerTempTable("myPredictions")
```

```
%sql
SELECT
  name, prediction
from
  myPredictions
join
  most_rated_movies on myPredictions.movie_id = most_rated_movies.movie_id
order by
  prediction desc
LIMIT
  10
```

name	prediction
Star Trek: The Wrath of Khan	6.1421475
Star Wars: Episode IV - A New Hope	5.3213224
Raiders of the Lost Ark	5.295201
Casablanca	5.278496
Star Trek IV: The Voyage Home	5.251287

Interpretation der Ergebnisse

Die angezeigte Tabelle enthält die zehn wichtigsten Filmempfehlungen für den Benutzer. Diese Prognosen basieren auf demografischen Daten zu Filmen und den Bewertungen des Benutzers.

Notebook 3

Demo eines Angriffserkennungssystems

Demo eines Angriffserkennungssystems

Angriffserkennungssysteme sind Geräte oder Softwareanwendungen, die Netzwerke oder Systeme im Hinblick auf schädliche Aktivitäten oder Richtlinienverstöße überwachen.

Dieses Notebook zeigt, wie Benutzer Bedrohungen im Internet besser erkennen können. Wir zeigen Ihnen, wie Sie Netzwerkaktivitätsprotokolle in Echtzeit überwachen, um Alarme zu verdächtigen Aktivitäten zu erstellen, Sicherheitszentren beim Untersuchen verdächtiger Aktivitäten zu unterstützen und Netzwerkverteilungsmodelle zum Zuordnen der Netzwerkoberfläche und der Bewegung von Entitäten zu entwickeln und so Penetrationspunkte zu identifizieren. Dieses Notebook:

- Ist eine vorgefertigte Lösung zusätzlich zu Apache® Spark™, erstellt in Scala innerhalb der Azure Databricks-Plattform.
- Verwendet eine **logistische Regression**, um Eindringversuche zu identifizieren, indem nach Abweichungen im Verhalten gesucht wird, um neue Angriffe zu identifizieren.
- Enthält einen Datensatz mit einer Teilmenge simulierter Netzwerkverkehrsbeispiele.
- Zeigt, wie die ersten drei Erkenntnisse durch Visualisierung gewonnen werden.
- Ermöglicht es Datenanalysten und Datentechnikern, die Genauigkeit zu verbessern, indem sie mehr Daten erhalten oder das Modell verbessern.

Schritt 1: Erfassen von IDS-Daten in einem Notebook

Die Datei „CIDDS-001 data set.zip“ kann von der CIDDS-Seite heruntergeladen werden.

Entpacken Sie die Daten, und laden Sie die Datei „CIDDS-001>traffic>ExternalServer>*.csv“ aus dem entpackten Ordner in die Databricks-Notebooks hoch.

```
val idsdata = sqlContext.read.format(“csv”)
  .option(“header”, “true”)
  .option(“inferSchema”, “true”)
  .load(“/FileStore/tables/ctrdsk051502399641231/”)

display(idsdata)
```

Date first seen	Duration	Proto	Src IP Addr	Src Pt	Dst IP Addr	Dst Pt	Packets	Bytes	Flows	Flags	Tos	class	attackType	attackID	attackDescription
2017-03-15T00:01:16.632+0000	0	TCP	192.168.100.5	445	192.168.220.16	58844	1	108	1	.AP...	0	normal	---	---	---
2017-03-15T00:01:16.552+0000	0	TCP	192.168.100.5	445	192.168.220.15	48888	1	108	1	.AP...	0	normal	---	---	---
2017-03-15T00:01:16.551+0000	0.004	TCP	192.168.220.15	48888	192.168.100.5	445	2	174	1	.AP...	0	normal	---	---	---
2017-03-15T00:01:16.631+0000	0.004	TCP	192.168.220.16	58844	192.168.100.5	445	2	174	1	.AP...	0	normal	---	---	---
2017-03-15T00:01:16.552+0000	0	TCP	192.168.100.5	445	192.168.220.15	48888	1	108	1	.AP...	0	normal	---	---	---
2017-03-15T00:01:16.631+0000	0.004	TCP	192.168.220.16	58844	192.168.100.5	445	2	174	1	.AP...	0	normal	---	---	---

```
val newNames = Seq(“datefirstseen”, “duration”, “proto”, “srcip”,“srcpt”,“dstip”,“dstpt”,“packets”,“bytes”,“flows”,“flags”,“tos”,“transtype”,“label”,“attackid”,“attackdescription”)
val dfRenamed = idsdata.toDF(newNames: _*)
val dfReformat = dfRenamed.select(“label”,“datefirstseen”, “duration”, “proto”,
“srcip”,“srcpt”,“dstip”,“dstpt”,“packets”,“bytes”,“flows”,“flags”,“tos”,“transtype”,“attackid”,“attackdescription”)
newNames: Seq[String] = List(datefirstseen, duration, proto, srcip, srcpt, dstip, dstpt, packets, bytes, flows, flags, tos, transtype, label, attackid, attackdescription)
dfRenamed: org.apache.spark.sql.DataFrame = [datefirstseen: timestamp, duration: double ... 14 more fields]
dfReformat: org.apache.spark.sql.DataFrame = [label: string, datefirstseen: timestamp ... 14 more fields]
```

Schritt 2: Ergänzen Sie die Daten, um zusätzliche Einblicke in den IDS-Datensatz zu erhalten.

Wir erstellen eine TEMPORAY-Tabelle aus dem Dateispeicherort „/tmp/wesParquet“ im PARQUET-Dateiformat.

Das PARQUET-Dateiformat ist das bevorzugte Dateiformat, da es für Notebooks in der Azure Databricks-Plattform optimiert ist.

```
%sql

    CREATE TEMPORARY TABLE temp_idsdata
    USING parquet
    OPTIONS (
      path “/tmp/wesParquet”
    )
Error in SQL statement: TempTableAlreadyExistsException:
Temporary table ‘temp_idsdata’ already exists;
```

Berechnen Sie Statistiken über die zurückgegebenen Inhaltsgrößen.

```
%sql
select min(trim(bytes)) as min_bytes,max(trim(bytes)) as max_
bytes,avg(trim(bytes)) as avg_bytes from temp_idsdata
```

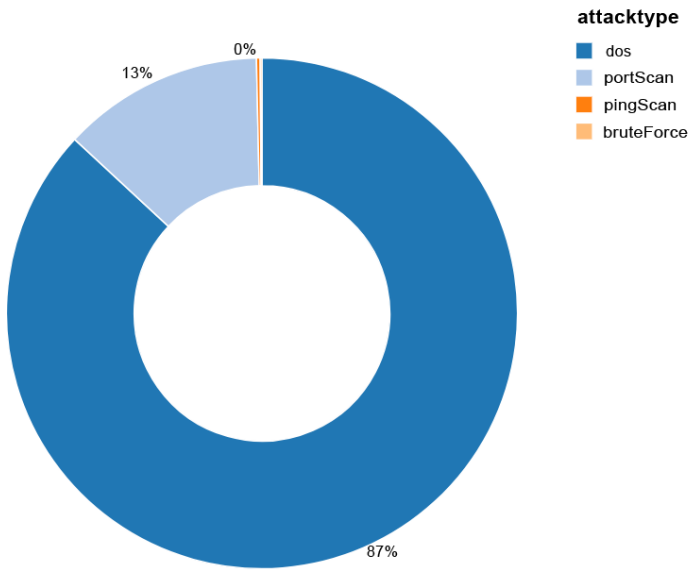
min_bytes	max_bytes	avg_bytes
1.0 M	99995	1980.1018585682032

Schritt 3: Durchsuchen der IDS-Daten durch Erfassen der Art von Angriffen im Netzwerk

Analyse der erfassten Angriffsarten

```
%sql

select attacktype, count(*) as the_count from temp_idsdata where
attacktype <> ‘---’ group by attacktype order by the_count desc
```



Schritt 4: Visualisierung

Finden und visualisieren Sie Sonderfälle.

Zeigen Sie eine Liste von IP-Adressen an, die mehr als N-mal auf den Server zugegriffen haben.

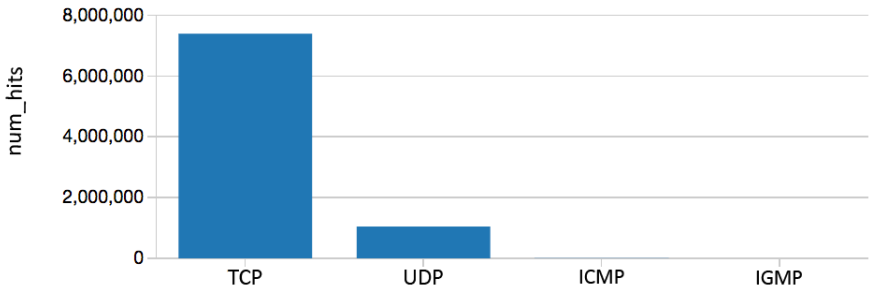
Untersuchen der für die Angriffe verwendeten Quell-IP

```
%sql
-- Verwenden Sie die parameterbasierte Abfrageoption, damit ein Viewer dynamisch einen Wert für N festlegen kann.
-- Die Anzahl von Ergebnissen muss nicht beschränkt werden.
-- Die Anzahl von zurückgegebenen Werten wird automatisch auf 1000 begrenzt.
-- Es gibt Optionen, um eine Abbildung anzuzeigen, die alle Daten enthält, um die Trends anzuzeigen.
SELECT srcip, COUNT(*) AS total FROM temp_idsdata GROUP BY srcip HAVING total > $N order by total desc

Befehl übersprungen
```

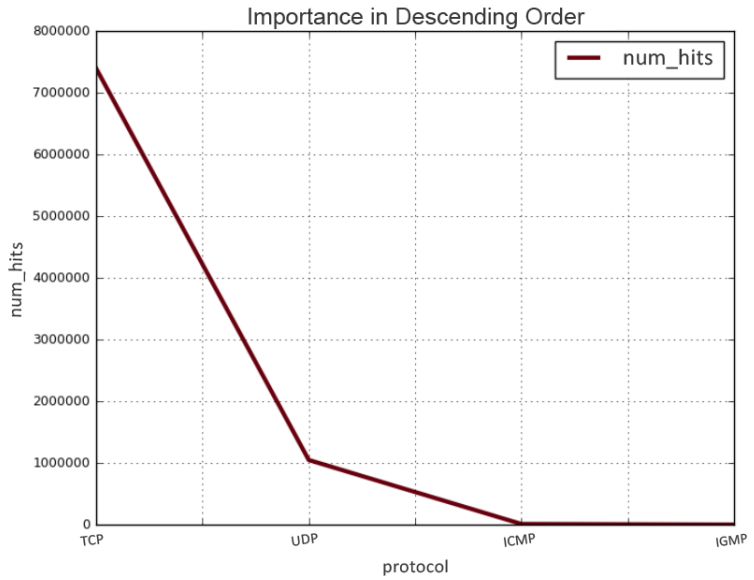
Durchsuchen von Statistiken über das für Angriffe verwendete Protokoll mithilfe von Spark SQL

```
%sql
-- Zeigen Sie eine Abbildung über die Verteilung der Trefferanzahl in Bezug auf die Endpunkte an.
SELECT Proto, count(*) as num_hits FROM temp_idsdata GROUP BY Proto ORDER BY num_hits DESC
```



Durchsuchen von Statistiken über das für Angriffe verwendete Protokoll mithilfe von Matplotlib

```
%python
import matplotlib.pyplot as plt
importance = sqlContext.sql("SELECT Proto as protocol, count(*) as
num_hits FROM temp_idsdata GROUP BY Proto ORDER BY num_hits DESC")
importanceDF = importance.toPandas()
ax = importanceDF.plot(x="protocol", y="num_hits",
lw=3,colormap='Reds_r',title='Importance in Descending Order', fontsize=9)
ax.set_xlabel("protocol")
ax.set_ylabel("num_hits")
plt.xticks(rotation=12)
plt.grid(True)
plt.show()
display()
```




```
%r
library(SparkR)
library(ggplot2)
importance_df  = collect(sql(sqlContext,'SELECT Proto as protocol,
count(*) as num_hits FROM temp_idsdata GROUP BY Proto ORDER BY num_hits
DESC'))
ggplot(importance_df, aes(x=protocol, y=num_hits)) + geom_
bar(stat='identity') + scale_x_discrete(limits=importance_
df[order(importance_df$num_hits), "protocol"]) + coord_flip()
```

Schritt 5: Modellerstellung

```
import org.apache.spark.ml.linalg.Vector;
import org.apache.spark.ml.feature._;
import org.apache.spark.ml.classification.LogisticRegression
import org.apache.spark.ml.feature.StringIndexer

case class Data(label: Double, feature: Seq[Double])

val indexer1 = new StringIndexer()
                    .setInputCol("proto")
                    .setOutputCol("protoIndex")
                    .fit(dfReformat)

val indexed1 = indexer1.transform(dfReformat)

val indexer2 = new StringIndexer()
                    .setInputCol("label")
                    .setOutputCol("labelIndex")
                    .fit(indexed1)

val indexed2 = indexer2.transform(indexed1)

val features = indexed2.rdd.map(row =>
Data(
   row.getAs[Double]("labelIndex"),
   Seq(row.getAs[Double]("duration"),row.getAs[Double]("protoIndex"))
)).toDF


val assembler = new VectorAssembler()
  .setInputCols(Array("duration", "protoIndex"))
  .setOutputCol("feature")

val output = assembler.transform(indexed2)
println("Assembled columns 'hour', 'mobile', 'userFeatures' to vector
column 'features'")
```

```
output.select("feature", "labelIndex").show(false)

val labeled = output.rdd.map(row =>
LabeledPoint(
   row.getAs[Double]("labelIndex"),
   row.getAs[org.apache.spark.ml.linalg.Vector]("feature")
)).toDF


val splits = labeled randomSplit Array(0.8, 0.2)

val training = splits(0) cache
val test = splits(1) cache

val algorithm = new LogisticRegression()
val model = algorithm fit training

val prediction = model.transform(test)

val predictionAndLabel = prediction.rdd.zip(test.rdd.map(x =>
x.getAs[Double]("label")))

predictionAndLabel.foreach((result) => println(s"predicted label:
${result._1}, actual label: ${result._2}"))
Assembled columns 'hour', 'mobile', 'userFeatures' to vector column 'features'
+-----------+-----------+
|feature    |labelIndex|
+-----------+-----------+
|(2,[],[])  |0.0        |
|(2,[],[])  |0.0        |
|[0.004,0.0]|0.0        |
|[0.004,0.0]|0.0        |
|(2,[],[])  |0.0        |
|[0.004,0.0]|0.0        |
|(2,[],[])  |0.0        |
|(2,[],[])  |0.0        |
|(2,[],[])  |0.0        |
|(2,[],[])  |0.0        |
```

```
|(2,[],[])    |0.0       |
|(2,[],[])    |0.0       |
|[0.082,0.0]  |0.0       |
|[0.083,0.0]  |0.0       |
|[0.089,0.0]  |0.0       |
|[0.083,0.0]  |0.0       |
|[0.089,0.0]  |0.0       |
|[0.086,0.0]  |0.0       |
|[0.0,1.0]    |0.0       |
|[0.0,1.0]    |0.0       |
+-------------+----------+
only showing top 20 rows

warning: there were two feature warnings; re-run with -feature for
details
import org.apache.spark.ml.linalg.Vector
import org.apache.spark.ml.feature._
import org.apache.spark.ml.classification.LogisticRegression
import org.apache.spark.ml.feature.StringIndexer
defined class Data
indexer1: org.apache.spark.ml.feature.StringIndexerModel = strIdx_8d1e73586ec7
indexed1: org.apache.spark.sql.DataFrame = [label: string,
datefirstseen: timestamp ... 15 more fields]
indexer2: org.apache.spark.ml.feature.StringIndexerModel = strIdx_
cf17935c04d4
indexed2: org.apache.spark.sql.DataFrame = [label: string,
datefirstseen: timestamp ... 16 more fields]
features: org.apache.spark.sql.DataFrame = [label: double, feature:
array<double>]
assembler: org.apache.spark.ml.feature.VectorAssembler = vecAssembler_
c4fe808dd912
output: org.apache.spark.sql.DataFrame = [label: string, datefirstseen:
timestamp ... 17 more fields]
labeled: org.apache.spark.sql.DataFrame = [label: double, features: vector]
splits: Array[org.apache.spark.sql.Dataset[org.apache.spark.sql.Row]]
= Array([label: double, features: vector], [label: double, features:
vector])
training: org.apache.spark.sql.Dataset[org.apache.spark.sql.Row] =
[label: double, features: vector]
test: org.apache.spark.sql.Dataset[org.apache.spark.sql.Row] = [label:
double, features: vector]
algorithm: org.apache.spark.ml.classification.LogisticRegression =
logreg_96ba18adfd6f
model: org.apache.spark.ml.classification.LogisticRegressionModel =
logreg_96ba18adfd6f
prediction: org.apache.spark.sql.DataFrame = [label: double, features:
vector ... 3 more fields]
predictionAndLabel: org.apache.spark.rdd.RDD[(org.apache.spark.sql.Row,
Double)] = ZippedPartitionsRDD2[610] at zip at command-622314:55
```

```
val loss = predictionAndLabel.map { case (p, l) =>
  val err = p.getAs[Double](“prediction”) - l
  err * err
}.reduce(_ + _)

val numTest = test.count()
val rmse = math.sqrt(loss / numTest)
loss: Double = 407383.0
numTest: Long = 1687519
rmse: Double = 0.491334336329945
```

Interpretation der Ergebnisse

Die obige Abbildung zeigt den Index an, der zum Messen der einzelnen Angriffstypen verwendet wird.

1. Der häufigste Angriffsart war Denial-of-Service (DOS), gefolgt von Portscans.
2. IP-192.168.220.16 war der Ursprung der meisten Angriffe (mindestens 14 % aller Angriffe).
3. Die meisten Angriffe nutzten das TCP-Protokoll.
4. Aus dem RMSE zeigt sich beim Ausführen des Modells auf die Testdaten für eine Prognose der Angriffsart, dass wir eine hohe Präzision von 0,4919 erreichen.

Schlussbemerkungen

Microsoft

Erhalten Sie relevante Einblicke in alle Ihre Daten

Wie Sie in diesen Szenarien sehen konnten, wurde Azure Databricks entwickelt, um Ihnen mehrere Möglichkeiten zum Optimieren Ihrer Insights sowie zum Lösen von Problemen zu bieten. Es wurde für Sie und Ihr Team entwickelt und bietet Ihnen mehr Möglichkeiten für die Zusammenarbeit, mehr Analyseleistung und eine schnelle Lösung für die Probleme, vor denen Ihr Unternehmen steht. Wir hoffen, dass Sie interessante Informationen erhalten haben und Azure Databricks bald selbst testen.

Jetzt einsteigen ➔