DHBW CAS
Duale Hochschule
Baden-Württemberg
Center for Advanced Studies

Laborbericht im Modul

Data Science and Big Data

W3M20018.1

Mitglieder Gruppe 3	Manmeet Kaur (CAS-TMINF20-W) Daniel Kirste (CAS-WWB20-W) Felix Öchnser (CAS-TMIE19-W) Rouven Schnapka (CAS-WMWI20-W) Mario Scholz (CAS-WMWI20-S)
Abgabedatum	23.12.2020
Link zum Repository:	https://github.com/dkirste/datascience2020

Inhalt

Big Data Prototyp am Beispiel eines Webshops

Aufgabenstellung

Das Ziel dieser Gruppenaufgabe war es, eine Anwendung zu entwickeln, wie sie im Rahmen der Vorlesung stückweise aufgebaut wurde. Dazu sollte eine konkrete Anwendungsidee entwickelt werden, welche die in der Vorlesung vorgestellten konkreten Implementierungen verwendet und auf einer Lambda- oder Kappa-Architektur aufbaut. Die Wahl der passenden Architektur ist dabei eine zu treffende Designentscheidung.

Vorstellung Use-Case

Der vorliegende Use-Case befasst sich mit der Betrachtung eines Warenkorbs (eng. shopping cart) eines fiktiven Web-Shops für Elektronikartikel. Hierbei sollen die hinzugefügten Produkte über eine geeignete Architektur erfasst und verarbeitet werden. Die zugrundeliegende Architektur wird im nachfolgenden Abschnitt näher aufgegriffen und dargestellt.

Aufbau Architektur

Der beschriebene Use-Case wurde mithilfe einer Kappa-Architektur realisiert. Dabei kommen verschiedene Komponenten zum Einsatz, welche in Interaktion zueinanderstehen. Eine Übersicht des Zusammenspiels der einzelnen Komponenten, kann nachfolgender Abbildung entnommen werden.

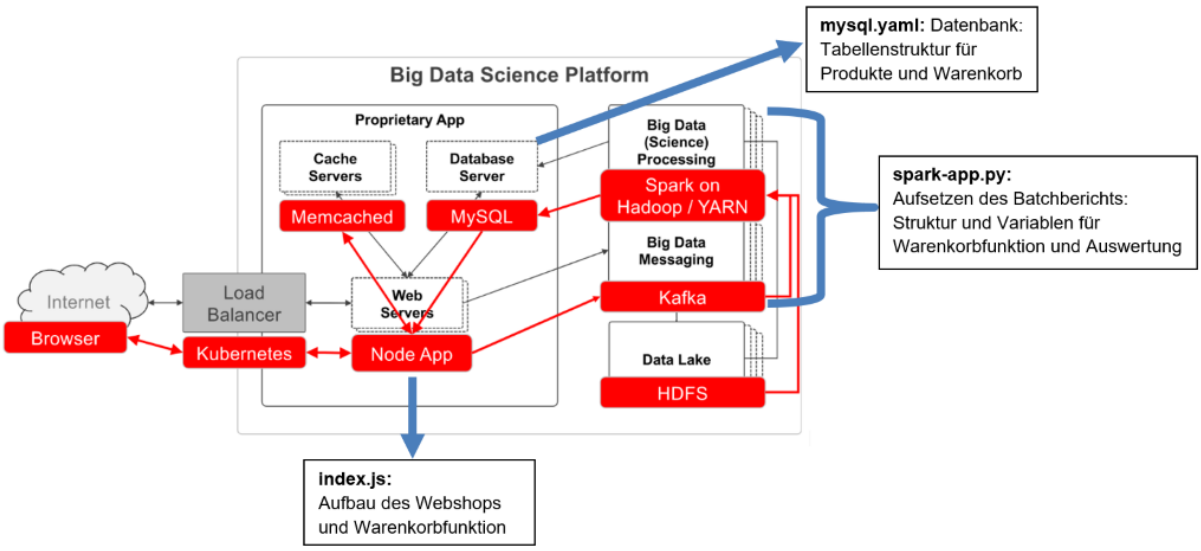


Abbildung 1 Big Data Science Platform

Durch das Hinzufügen eines Artikels zu dem Warenkorb (Button “Buy this Article“) auf der Homepage des Web-Shops sendet der Browser die Informationen an einen Load-Balancer, welcher in der vorliegenden Anwendung durch Kubernetes realisiert wurde. Über den Load-Balancer gelangt die Anfrage zur JavaScript-basierten Web-Applikation, welche die Verteilung der Informationen und Daten an den Cache-Server und das Kafka-Cluster übernimmt. Die Produkte auf der Webseite werden von der MySQL-Datenbank geladen. Um

diese zu entlasten wird der Cacher-Server verwendet, der die Daten zwischenspeichert. Wird ein Produkt in den Warenkorb gelegt, so wird mittels JavaScript eine Nachricht (eng. message) in das Topic “cart-data” des Kafka-Clusters gesendet. Die Web-Applikation ist somit der Producer im Kafkakontext. Die Spark App ist Consumer des Topics “cart-data”. Neue Messages werden von dieser ausgewertet und in Batches verarbeitet. Die Spark App speichert gleichzeitig die aus den Messages gewonnen Informationen persistent in der MySQL-Datenbank ab. Zur Darstellung des Warenkorbs, werden die Daten von der Web-Applikation wiederum direkt aus der Datenbank geladen.

Vorstellung elementarer Codebestandteile

Die nachstehende Funktion wird verwendet, um die gespeicherten Informationen zu den im Warenkorb hinterlegten Artikeln aus der Datenbank abzurufen. Die maximale Anzahl der auszugebenden Einträge im Warenkorb, kann dabei über die Mitgabe des Parameters “productCount” vorgegeben werden.

```
286  async function getShoppingCart(productCount) {
287      const query = "SELECT product, count FROM cart ORDER BY count DESC LIMIT ?"
288      return (await executeQuery(query, [productCount]))
289          .fetchAll()
290          .map(row => ({ product: row[0], count: row[1] }))
291  }
```

Abbildung 2 Funktion „productCount“

Wird im Web-Shop ein einzelner Artikel ausgewählt, wird ein Fetch auf /pushToCart/<productID> ausgeführt. Im Rahmen dieses Fetch wird die JavaScript Funktion “sendTrackingMessage” ausgeführt, die eine Message an das Kafka-Cluster sendet. Die zu übertragenden Informationen beinhalten dabei die Bezeichnung des Produktes, sowie den Zeitstempel des Zugriffs. Im Erfolgsfall wird in der Konsole die Rückmeldung des erfolgreichen Sendens der Informationen an Kafka quittiert. Andernfalls wird eine entsprechende Fehlermeldung an die Konsole ausgegeben.

```
330  // PushToCart
331  app.get("/pushToCart/:product", (req, res) => {
332      // Send the tracking message to Kafka
333      sendTrackingMessage({
334          product: req.params["product"],
335          timestamp: Math.floor(new Date() / 1000)
336      }).then(() => console.log("Sent to kafka"))
337          .catch(e => console.log("Error sending to kafka", e))
338      res.send(`<h1>Successed!`)
339  });
```

Abbildung 3 Fetch „pushToCart“

Data Science am Beispiel Human Resources

Aufgabenstellung

Das Ziel der Aufgabenstellung ist es, eine Data Science Anwendung zu entwickeln und dabei die in der Vorlesung kennengelernten Techniken einzusetzen. Die Anwendung wird gemäß des CRISP-DM Vorgehensmodells (siehe Abbildung 1) strukturiert und soll dessen Ablauf folgen.

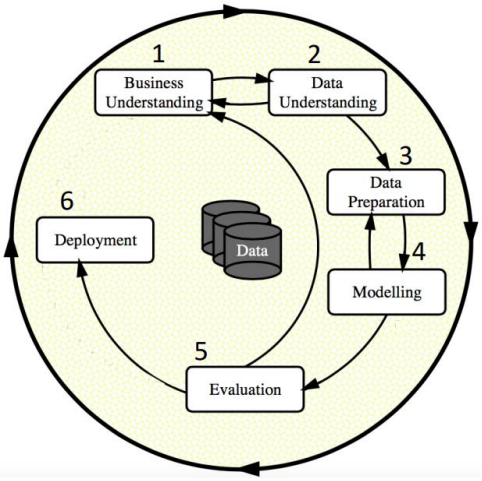


Abbildung 4 CRISP-DM Process Modell

Gewünscht sind hierbei ein oder mehrere Jupyter Notebooks, die unter Verwendung von pyspark und der mllib das Modell erzeugen (Schritt 4) und evaluieren (Schritt 5). Welche Algorithmen sie einsetzen, ist ihnen selbst überlassen. Das Deployment (Schritt 6) soll auf einem Hadoop Cluster erfolgen (gegebenenfalls als Python Anwendung). Wichtig ist die Dokumentation ihres iterativen Vorgehens. Grundlegende Entwicklungsstufen und Varianten des Modells und der Datenaufbereitung müssen für den Leser erhalten bleiben.

Use-Case 1 - mushrooms

Business Understanding

Die erste Phase konzentriert sich auf die präzise Beschreibung der betriebswirtschaftlichen Problemstellung und deren Ziele. Auf Basis dieser Ziele sollen im weiteren Verlauf des Datamining-Prozesses konkrete Anforderungen an die Datenanalyse formuliert und verfolgt werden können.

Im vorliegenden Fall steht die Betrachtung von Pilzen im Vordergrund, welche anhand verschiedener Merkmale in die Kategorien „essbar“ und „giftig“ eingeteilt werden können.

Data Understanding

In der zweiten Phase sollen Anforderungen an die Rohdaten für den Business-Case definiert werden. Dazu gehören Tätigkeiten, wie das Sammeln von Daten, deren Beschreibung und Untersuchung, sowie deren Bewertung.

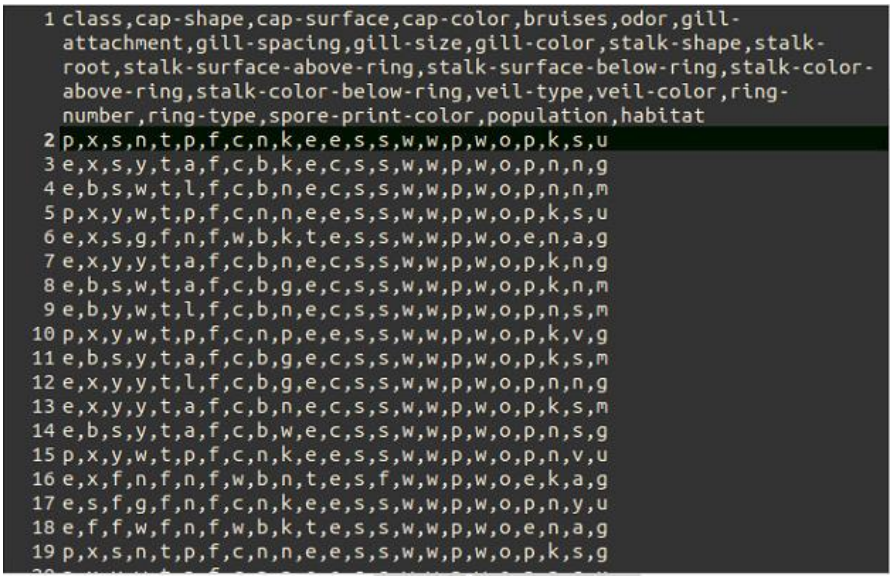
Der Prozess der Datensammlung ist mit enormem Aufwand verbunden, weshalb in der vorliegenden Projektarbeit auf online zugängige Datensätze zurückgegriffen wird.

Data Preparation

In der Data Preparation werden die Daten ausgewählt, bereinigt, transformiert und formatiert.

Für den oben beschriebenen Business-Case wurde zu Beginn ein passender Datensatz herausgesucht. Dieser enthält über 23 Arten von Pilzen, wobei jede Art als definitiv essbar, definitiv giftig oder von unbekannter Essbarkeit identifiziert wird. Diese letztere Klasse wird der giftigen Art zugeordnet. Des Weiteren enthält der ausgewählte Datensatz die Informationen von 8.124 Pilzen, welche anhand von 21 Kriterien beschrieben sind.

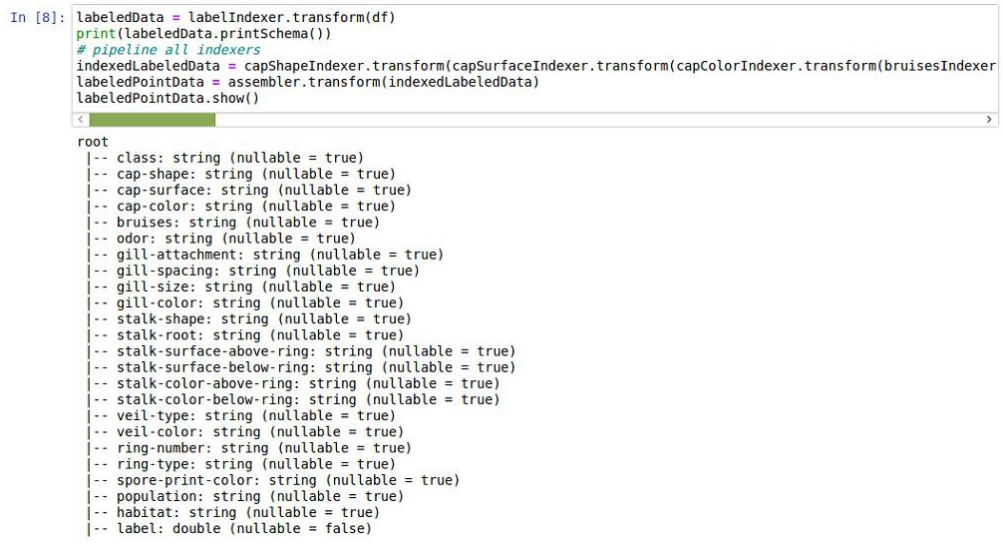
Um in diesen einen tieferen Einblick zu erhalten, wurde dieser für die weitere Verarbeitung vorbereitet. Dies umfasste die Umwandlung der zeichenbasierten Informationen in Zahlwerte, welche von den Algorithmen genutzt werden können. Bei dieser Aufbereitung sind wir auf Probleme bei der Indexierung gestoßen, welche im nachfolgenden näher beschrieben werden.



```
1 class,cap-shape,cap-surface,cap-color,bruises,odor,gill-
  attachment,gill-spacing,gill-size,gill-color,stalk-shape,stalk-
  root,stalk-surface-above-ring,stalk-surface-below-ring,stalk-color-
  above-ring,stalk-color-below-ring,veil-type,veil-color,ring-
  number,ring-type,spore-print-color,population,habitat
2 p,x,s,n,t,p,f,c,n,k,e,e,s,s,w,w,p,w,o,p,k,s,u
3 e,x,s,y,t,a,f,c,b,k,e,c,s,s,w,w,p,w,o,p,n,n,g
4 e,b,s,w,t,l,f,c,b,n,e,c,s,s,w,w,p,w,o,p,n,n,m
5 p,x,y,w,t,p,f,c,n,n,e,e,s,s,w,w,p,w,o,p,k,s,u
6 e,x,s,g,f,n,f,w,b,k,t,e,s,s,w,w,p,w,o,e,n,a,g
7 e,x,y,y,t,a,f,c,b,n,e,c,s,s,w,w,p,w,o,p,k,n,g
8 e,b,s,w,t,a,f,c,b,g,e,c,s,s,w,w,p,w,o,p,k,n,m
9 e,b,y,w,t,l,f,c,b,n,e,c,s,s,w,w,p,w,o,p,n,s,m
10 p,x,y,w,t,p,f,c,n,p,e,e,s,s,w,w,p,w,o,p,k,v,g
11 e,b,s,y,t,a,f,c,b,g,e,c,s,s,w,w,p,w,o,p,k,s,m
12 e,x,y,y,t,l,f,c,b,g,e,c,s,s,w,w,p,w,o,p,n,n,g
13 e,x,y,y,t,a,f,c,b,n,e,c,s,s,w,w,p,w,o,p,k,s,m
14 e,b,s,y,t,a,f,c,b,w,e,c,s,s,w,w,p,w,o,p,n,s,g
15 p,x,y,w,t,p,f,c,n,k,e,e,s,s,w,w,p,w,o,p,n,v,u
16 e,x,f,n,f,n,f,w,b,n,t,e,s,f,w,w,p,w,o,e,k,a,g
17 e,s,f,g,f,n,f,c,n,k,e,e,s,s,w,w,p,w,o,p,n,y,u
18 e,f,f,w,f,n,f,w,b,k,t,e,s,s,w,w,p,w,o,e,n,a,g
19 p,x,s,n,t,p,f,c,n,n,e,e,s,s,w,w,p,w,o,p,k,s,g
```

Abbildung 5 Datensatz mushrooms unbehandelt

Wie auf Abbildung 5 zu sehen ist, sind alle Daten zeichenbasierend gespeichert. Wie oben beschrieben, werden diese mithilfe von Indexern (siehe Abbildung 6) umgewandelt.



```
In [8]: labeledData = labelIndexer.transform(df)
print(labeledData.printSchema())
# pipeline all indexers
indexedLabeledData = capShapeIndexer.transform(capSurfaceIndexer.transform(capColorIndexer.transform(bruisesIndexer
labeledPointData = assembler.transform(indexedLabeledData)
labeledPointData.show()

root
 |-- class: string (nullable = true)
 |-- cap-shape: string (nullable = true)
 |-- cap-surface: string (nullable = true)
 |-- cap-color: string (nullable = true)
 |-- bruises: string (nullable = true)
 |-- odor: string (nullable = true)
 |-- gill-attachment: string (nullable = true)
 |-- gill-spacing: string (nullable = true)
 |-- gill-size: string (nullable = true)
 |-- gill-color: string (nullable = true)
 |-- stalk-shape: string (nullable = true)
 |-- stalk-root: string (nullable = true)
 |-- stalk-surface-above-ring: string (nullable = true)
 |-- stalk-surface-below-ring: string (nullable = true)
 |-- stalk-color-above-ring: string (nullable = true)
 |-- stalk-color-below-ring: string (nullable = true)
 |-- veil-type: string (nullable = true)
 |-- veil-color: string (nullable = true)
 |-- ring-number: string (nullable = true)
 |-- ring-type: string (nullable = true)
 |-- spore-print-color: string (nullable = true)
 |-- population: string (nullable = true)
 |-- habitat: string (nullable = true)
 |-- label: double (nullable = false)
```

Abbildung 6 Data Preparation mushrooms

Die umgewandelten Daten sind in Abbildung 7 ersichtlich.

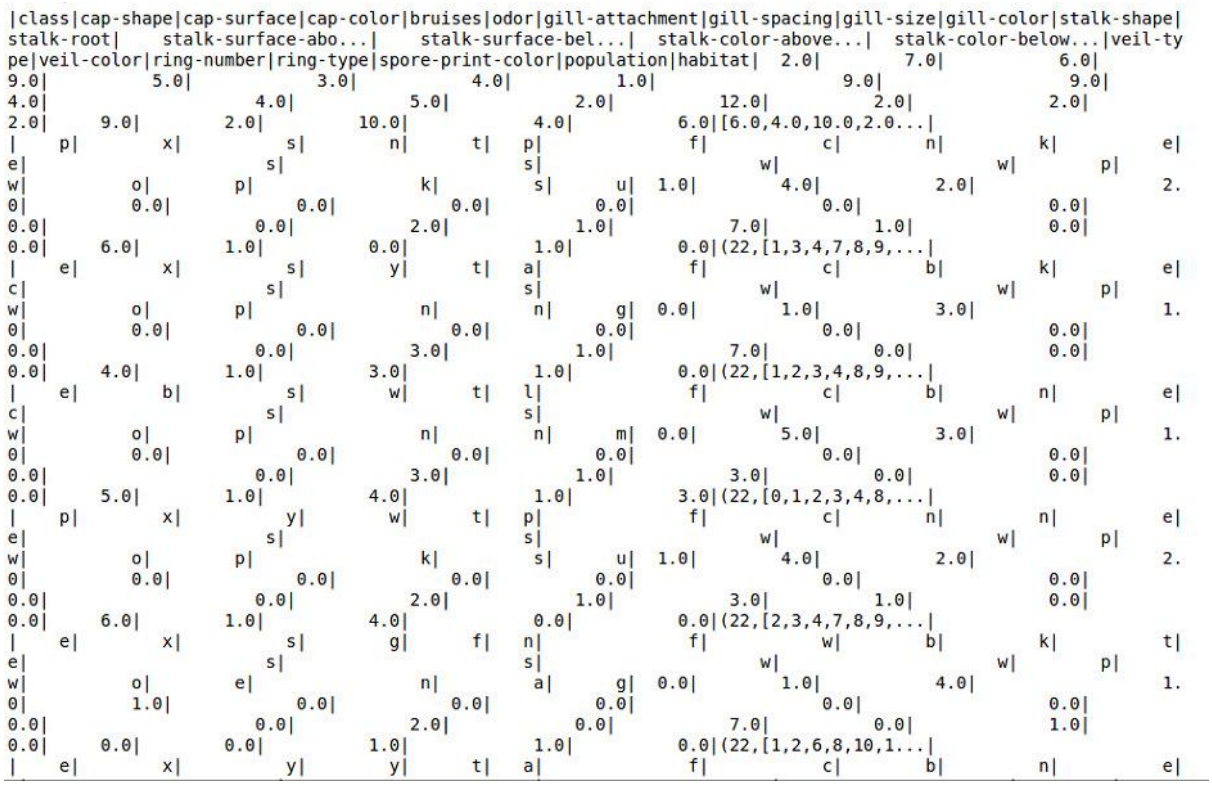


Abbildung 7 Datensatz mushrooms behandelt

Wie aus den Screenshots ersichtlich wird, ist der Aufwand für die Datenaufbereitung sehr hoch. Des Weiteren enthält der Datensatz keinen unique Identifier, weshalb ein Rückschluss der Algorithmen-Ergebnisse auf einzelne Daten nicht möglich ist.

Aufgrund der begrenzten Bearbeitungszeit entschied sich deshalb das Team, einen anderen Business-Case zu bearbeiten.

Use-Case 2 – Human Resources

Auf der Suche nach einem alternativen Datensatz sind wir auf Daten bezüglich Human Resources gestoßen. Anhand diesem wird nun ein neuer Use-Case aufgebaut und durchgeführt.

Business Understanding

Da die Akquise neuer Mitarbeiter für Unternehmen sehr zeitaufwändig und kostenintensiv ist, besteht großes Interesse darin, potenzielle Abgänge bestehender Mitarbeiter zu erkennen. Im ausgewählten Business-Case soll ein Vorhersagemodell entwickelt werden, welches es den Unternehmen ermöglicht, diese Mitarbeiter frühzeitig zu identifizieren. Anhand dieser Erkenntnisse können anschließend entsprechende Gegenmaßnahmen eingeleitet werden.

Data Understanding

Für den Business-Case wurde ein passender Datensatz zu Informationen von 312 Mitarbeitern ausgewählt, die jeweils durch 35 Merkmale (siehe Abbildung 8) repräsentiert werden. Dieser Datensatz beinhaltet Daten, wie zum Beispiel Name, Gehalt und Position im Unternehmen.

```
root
 |-- EmpLastname: string (nullable = true)
 |-- EmpFirstname: string (nullable = true)
 |-- EmpID: integer (nullable = true)
 |-- MarriedID: integer (nullable = true)
 |-- MaritalStatusID: integer (nullable = true)
 |-- GenderID: integer (nullable = true)
 |-- EmpStatusID: integer (nullable = true)
 |-- DeptID: integer (nullable = true)
 |-- PerfScoreID: integer (nullable = true)
 |-- FromDiversityJobFairID: integer (nullable = true)
 |-- Salary: integer (nullable = true)
 |-- Termd: integer (nullable = true)
 |-- PositionID: integer (nullable = true)
 |-- Position: string (nullable = true)
 |-- State: string (nullable = true)
 |-- Zip: integer (nullable = true)
 |-- DOB: string (nullable = true)
 |-- Sex: string (nullable = true)
 |-- MaritalDesc: string (nullable = true)
 |-- CitizenDesc: string (nullable = true)
 |-- HispanicLatino: string (nullable = true)
 |-- RaceDesc: string (nullable = true)
 |-- DateofHire: string (nullable = true)
 |-- DateofTermination: string (nullable = true)
 |-- TermReason: string (nullable = true)
 |-- EmploymentStatus: string (nullable = true)
 |-- Department: string (nullable = true)
 |-- ManagerName: string (nullable = true)
 |-- ManagerID: integer (nullable = true)
 |-- RecruitmentSource: string (nullable = true)
 |-- PerformanceScore: string (nullable = true)
 |-- EngagementSurvey: double (nullable = true)
 |-- EmpSatisfaction: integer (nullable = true)
 |-- SpecialProjectsCount: integer (nullable = true)
 |-- LastPerformanceReview_Date: string (nullable = true)
 |-- DaysLateLast30: integer (nullable = true)
 |-- Absences: integer (nullable = true)
```

Abbildung 8 Merkmale Human Ressources

Um ein initiales Verständnis über den Datensatz zu erlangen, wurde mit den bestehenden Daten eine Visualisierung für einen Anwendungsfall durchgeführt. Spezifisch wurde der Leistungsindex in Relation zum Rekrutierungsweg dargestellt. So wäre beispielsweise eine Erkenntnis aus dem Graphen, dass Bewerber von diversen Jobmessen die Leistungserwartungen vergleichsweise am Häufigsten übertreffen.

```
In [9]: plt.figure(figsize=(15,5))
        dfPandas= df.toPandas()
        perfscore_percentage = (dfPandas.groupby(['RecruitmentSource'])['PerformanceScore']
                                .value_counts(normalize=True)
                                .rename('percentage')
                                .mul(100)
                                .reset_index()
                                .sort_values('PerformanceScore'))
        sbs.barplot(x="RecruitmentSource", y="percentage", hue="PerformanceScore", data=perfscore_percentage)
        plt.xticks(rotation = 30)
```

```
Out[9]: (array([0, 1, 2, 3, 4, 5, 6, 7, 8]),
         [Text(0, 0, 'Website'),
          Text(1, 0, 'CareerBuilder'),
          Text(2, 0, 'Diversity Job Fair'),
          Text(3, 0, 'LinkedIn'),
          Text(4, 0, 'Employee Referral'),
          Text(5, 0, 'Indeed'),
          Text(6, 0, 'Google Search'),
          Text(7, 0, 'Other'),
          Text(8, 0, 'On-line Web application')])
```

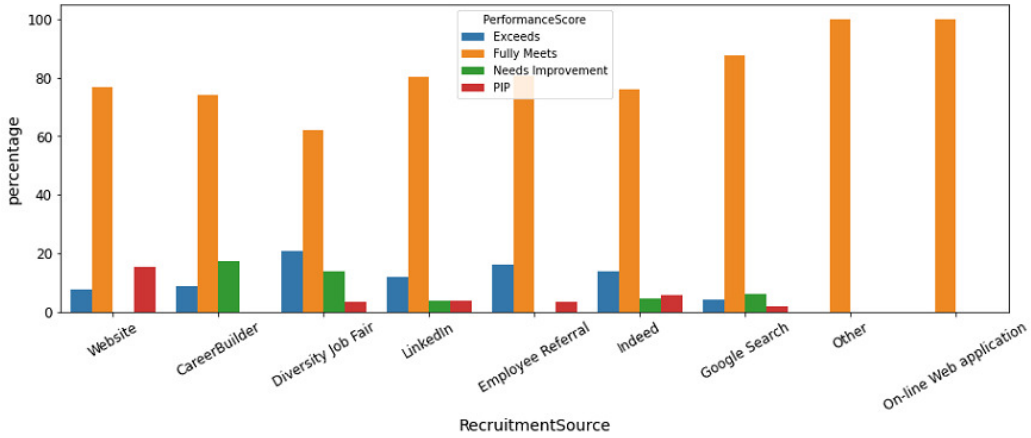


Abbildung 9 Leistungsindex in Relation zum Rekrutierungsweg

Data Preparation

Um die Daten in den Algorithmen nutzen zu können, wurden diese im Rahmen der Vorverarbeitung (Preprocessing) behandelt. Dabei wurden wie schon in Use-Case 1 alle String- in Integer-Werte mithilfe der in pyspark zur Verfügung stehenden Index-Funktionalitäten umgewandelt. Im Falle von Datumsfeldern wurden diese mithilfe der Funktion „unix_timestamp“ in Long formatiert.

Modeling

Im Rahmen der Modellierung werden die für die Aufgabenstellung geeigneten Methoden des Data Minings auf den in der Datenvorbereitung erstellten Datensatz ausgewählt. Typisch für diese Phase sind die Erstellung von Testmodellen, deren Bewertung, sowie die anschließende Optimierung der verwendeten Parameter.

In dem vorliegenden Fall wurde eine zusätzliche Spalte, welche die Differenz zwischen dem Einstellungs- und Kündigungsdatum eines Mitarbeiters enthält. Des Weiteren wurden zunächst die jeweiligen Stundenlöhne berechnet und anschließend mithilfe des Bucketizers den verschiedenen zuvor definierten Werten zugeordnet. Hierdurch konnte die Anzahl der unterschiedlichen Werte deutlich reduziert werden.

```
# Fill null values
df = df.na.fill( {'TermReason': 'Unknown', 'ManagerID': 0, 'DaysLateLast30': 0, 'DateofTermination': '1/01/2021', 'LastPerformanceReview_
Date': '1/01/2021'} )

# Converte date-strings to dates
df = df.withColumn('DateofHire', unix_timestamp(col('DateofHire'), 'M/dd/yyyy'))
df = df.withColumn('DOB', unix_timestamp(col('DOB'), 'MM/dd/yy'))
df = df.withColumn('DateofTermination', unix_timestamp(col('DateofTermination'), 'M/dd/yyyy'))
df = df.withColumn('LastPerformanceReview_Date', unix_timestamp(col('LastPerformanceReview_Date'), 'M/dd/yyyy'))

# Add new column for days worked in company
df = df.withColumn("DaysWorked", ((col("DateofTermination") - col("DateofHire"))/86400))

# Bucketizing Salary PayRate to hourly
df = df.withColumn('PayRate', (col('Salary')/2000))
bucketizer = Bucketizer(splits=[ 10, 15, 20, 25, 30, 40, 45, 50, 55, 60, 65, 70, 75, 80, 85, 90, float('Inf') ],inputCol="PayRate", outpu
tCol="PayRateBucket")
df = bucketizer.setHandleInvalid("keep").transform(df)

# Bucketizing Salary
bucketizer = Bucketizer(splits=[ 10000, 15000, 20000, 25000, 30000, 35000, 40000, 45000, 50000, 55000, 60000, 65000, 70000, 75000, 80000,
85000, 90000, 95000, 100000, 105000, 110000, 115000, 120000, 125000, 130000, 135000, 140000, 145000, 150000, 155000, 160000, 165000, 1700
00, 175000, 180000, 185000, 190000, 195000, 200000, float('Inf') ],inputCol="Salary", outputCol="SalaryBucket")
df = bucketizer.setHandleInvalid("keep").transform(df)
```

Abbildung 10 Berechnung „DaysWorked“

Scatter-Plot

Die resultierende Spalte „days worked“ berechnet mithilfe der Funktion dateDiff das gewünschte Ergebnis. Auf Basis dieser Spalte wurde ein Scatter-Plot aufgesetzt, welches unter anderem auf diese Differenz zugreift und diese zusammen mit dem Stundenlohn der Mitarbeiter darstellt (siehe Abbildung 11). Der Status des jeweiligen Mitarbeiters wird zudem über die Farbe visualisiert.

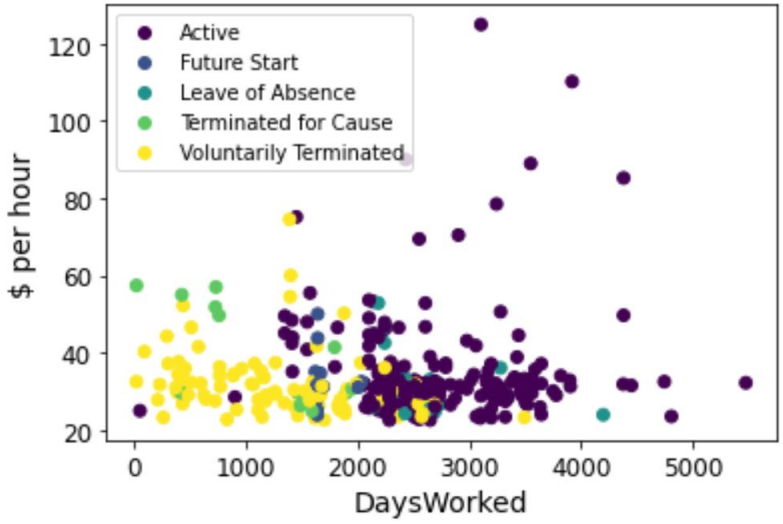


Abbildung 11 Scatter-Plot Datensatz

Decision-Tree

Daraufhin soll anhand von einem Decision-Tree vorhergesagt werden, welcher Mitarbeiter das Unternehmen als nächstes verlassen wird. Dieser Decision-Tree wird mit der Variable paramGrid parametriert. Hier wurde beispielsweise die maximale Tiefe des Entscheidungsbaumes durch maxDepth auf fünf Ebenen eingestellt. Des Weiteren wurde mit dem Parameter minInfoGain vorgegeben, dass nur Features mit einem minimalen Informationsgehalt von 0,05 bei der Entscheidung berücksichtigt werden.

Um im weiteren Verlauf den Algorithmus trainieren und validieren zu können, wurde der Datensatz in einen Trainings- und einen Testdatensatz im Verhältnis 60 zu 40 aufgeteilt. Als Validierungsansatz wurde auf die Methode der Cross-Validation gesetzt und anschließend das Ergebnis dargestellt.

```
featureIndexer = VectorIndexer(inputCol="features",outputCol="indexedFeatures", maxCategories=6)

predConverter = IndexToString(inputCol="prediction",outputCol="predictedLabel",labels=termdIndexer.labels)

dt = DecisionTreeClassifier(labelCol="Termd", featuresCol="features")

paramGrid = ParamGridBuilder().addGrid(dt.maxDepth, [ 5 ]) \
                              .addGrid(dt.minInfoGain, [0.05]) \
                              .addGrid(dt.minInstancesPerNode, [10]) \
                    .addGrid(dt.maxBins, [28]) \
                              .build()

splits = df.randomSplit([0.6, 0.4 ], 1234)
train = splits[0]
test = splits[1]
pipeline = Pipeline(stages= [termdIndexer, empLastnameIndexer, empFirstnameIndexer, positionIndexer, stateIdIndexer, sexIndexer, maritalDescIndexer, citizenDescIndexer, hispanicLatinoIndexer, raceDescIndexer, termReasonIndexer
evaluator =  BinaryClassificationEvaluator(labelCol="Termd",rawPredictionCol="rawPrediction", metricName="areaUnderROC")

cv = CrossValidator(estimator=pipeline, evaluator=evaluator,estimatorParamMaps=paramGrid,numFolds=3, parallelism=2)
cvModel = cv.fit(train)


treeModel = cvModel.bestModel.stages[7]
print("Learned classification tree model:\n",treeModel)
print("Best Params: \n", treeModel.explainParams())

predictions = cvModel.transform(test)
predictions.select("prediction", "Termd", "predictedLabel", "rawPrediction", "Termd", "features").show()

accuracy = evaluator.evaluate(predictions)
print("Test Error = " ,(1.0 - accuracy))
```

Abbildung 12 Decision-Tree Coding

Wie die Testergebnisse (siehe Abbildung 13) zeigen, weißt dieser Entscheidungsbaum (EB) eine 100%-ige Vorhersage (Test Error = 0.0) auf. Dies deutet auf einen Implementierungsfehler hin, da die Test- von den Trainingsdaten abweichen und somit eine 100%-ige Übereinstimmung nicht erreichbar ist. Auch nach intensiver Recherchearbeit kann der Grund hierfür nicht gefunden werden. Die Vermutung liegt nahe, dass der EB für den vorliegenden Datensatz ungeeignet zu sein scheint, da dieser ggf. zu viele Features enthält.

Aufgrund der begrenzten Bearbeitungszeit entschied sich das Team nach mehrmaligen gescheiterten Anläufen mit nicht validen Ergebnissen diesen Weg nicht weiter zu verfolgen.

```
+----------+-----+--------------+-------------+-----+--------------------+
|prediction|Termd|predictedLabel|rawPrediction|Termd|            features|
+----------+-----+--------------+-------------+-----+--------------------+
|       0.0|    0|             0|  [129.0,0.0]|    0|(24,[2,3,4,5,8,11...|
|       1.0|    1|             1|   [0.0,50.0]|    1|[1.0,1.0,1.0,5.0,...|
|       1.0|    1|             1|   [0.0,50.0]|    1|(24,[1,3,4,5,7,8,...|
|       0.0|    0|             0|  [129.0,0.0]|    0|(24,[1,2,3,4,5,8,...|
|       0.0|    0|             0|  [129.0,0.0]|    0|(24,[2,3,4,5,8,11...|
|       1.0|    1|             1|   [0.0,50.0]|    1|(24,[2,3,4,5,7,8,...|
|       0.0|    0|             0|  [129.0,0.0]|    0|(24,[1,3,4,5,8,11...|
|       1.0|    1|             1|   [0.0,50.0]|    1|(24,[3,4,5,7,8,11...|
|       1.0|    1|             1|   [0.0,50.0]|    1|[0.0,0.0,1.0,4.0,...|
|       0.0|    0|             0|  [129.0,0.0]|    0|[1.0,1.0,0.0,1.0,...|
|       1.0|    1|             1|   [0.0,50.0]|    1|(24,[2,3,4,5,7,8,...|
|       0.0|    0|             0|  [129.0,0.0]|    0|[1.0,1.0,0.0,1.0,...|
|       0.0|    0|             0|  [129.0,0.0]|    0|(24,[1,3,4,5,6,8,...|
|       1.0|    1|             1|   [0.0,50.0]|    1|[1.0,1.0,0.0,5.0,...|
|       0.0|    0|             0|  [129.0,0.0]|    0|(24,[3,4,5,8,11,1...|
|       0.0|    0|             0|  [129.0,0.0]|    0|(24,[0,1,2,3,4,5,...|
|       0.0|    0|             0|  [129.0,0.0]|    0|(24,[2,3,4,5,8,11...|
|       0.0|    0|             0|  [129.0,0.0]|    0|(24,[3,4,5,8,9,11...|
|       0.0|    0|             0|  [129.0,0.0]|    0|(24,[3,4,5,8,11,1...|
|       0.0|    0|             0|  [129.0,0.0]|    0|(24,[2,3,4,5,8,11...|
+----------+-----+--------------+-------------+-----+--------------------+
only showing top 20 rows

Test Error =  0.0
```

Abbildung 13 Decision-Tree Ergebnisse

Clustering

Um neue Erkenntnisse über den Datensatz zu erlangen, wurde ein Clustering über den Datensatz angewendet. Ziel hierbei war es, so die Daten in Cluster einzuteilen, dass von den Clustern auf das Angestelltenverhältnis rückgeschlossen werden kann. Somit soll bestenfalls aufgezeigt werden können, welche Mitarbeiter Merkmale besitzen, die bei anderen Mitarbeitern zu einem Abgang geführt haben.

Hierfür wurden diese Informationen in einem Feature-Array mithilfe des auf Abbildung 14 gezeigten Vector-Assembler zusammengefasst.

```
vecAssembler = VectorAssembler(inputCols=["PayRate", "DaysWorked"], outputCol="features")
new_df = vecAssembler.transform(df)
```

Abbildung 14 Vector Assembler Clustering

Für die Cluster-Analyse wurde auf den KMeans-Algorithmus gesetzt, was ein Verfahren zur Vektorquantisierung darstellt. Mithilfe des Parameters ‚k' wird die Anzahl der bekannten Cluster vorgegeben. Der Parameter „seed" steuert dabei die zufälligen Startwerte der KMeans-Cluster. Da KMeans sensibel auf die vorgegebenen Startwerte reagiert, müssen unterschiedliche Werte für diesen Parameter ausprobiert werden, um ein zufriedenstellendes Ergebnis zu erreichen.

In unserem auf Abbildung 15 gezeigten Fall werden die Daten in 5 Gruppen gegliedert und auf das Feature-Array angewandt und seed auf den Wert 1337 festgelegt. Zusätzlich wird bei der Vorbereitung zum Clustering Traings- und Testdaten mit dem Verhältnis 95 zu 5 festgelegt.

```
kmeans = KMeans(k=5, seed=1337)

# Split data
splits = new_df.randomSplit([0.95, 0.05 ], 420)
training = splits[0]
test = splits[1]
```

Abbildung 15 KMeans Clustering

Das Ergebnis der Cluster-Analyse wird mithilfe eines Scatter-Plots, wie er auf Abbildung 16 dargestellt ist, visualisiert. Um das Ergebnis direkt mit dem Angestelltenverhältnis vergleichen zu können, wird dieses nochmals auf Abbildung 17 dargestellt.

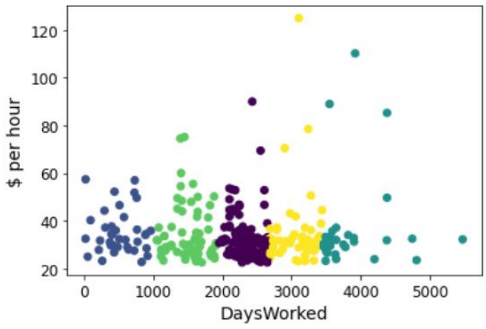


Abbildung 16 Scatter-Plot Clustering

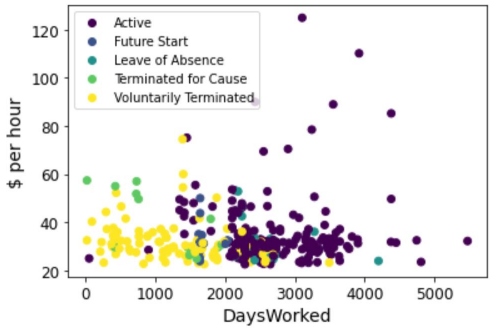


Abbildung 17 Scatter-Plot Datensatz

Wie der direkte Vergleich zeigt, finden sich die meisten aktiven Mitarbeiter (Abbildung 17 „Active") in den oberen Clustern (Cluster 3 bis 5) wieder. Des Weiteren sind die Mitarbeiter, welche das Unternehmen freiwillig verlassen haben (Abbildung 17 „Voluntarily Terminated"), den ersten beiden Clustern zugeordnet. Eine weitere bzw. detailliertere Aussage ist auf dieser Basis leider nicht möglich.

Random Forest

Als Alternative zum Clustering und um eine detailliertere Aussage treffen zu können, wurde daraufhin ein Random Forest angewandt. Dies ist ein Algorithmus, der sich für Klassifizierungs- und Regressionsaufgaben nutzen lässt. Er kombiniert die Ergebnisse vieler verschiedener Entscheidungsbäume, um bestmögliche Entscheidungen zu treffen. In diesem Fall sagt er aus, ob ein Mitarbeiter das Unternehmen verlässt.

Zunächst wurde, wie Abbildung 18 zeigt, der Datensatz in Trainings- und Testdaten im Verhältnis 90 zu 10 aufgeteilt.

```
labledPointDataSet = new_df
splits = labledPointDataSet.randomSplit([0.9, 0.1 ], 420)
training = splits[0]
test = splits[1]
```

Abbildung 18 Random Forest Aufteilung Trainings- und Testdaten

Um den Algorithmus anwenden zu können wurde zunächst der bereits vorhandene Vektorassembler mit den Parametern EngagementSurvey und SpecialProjectsCount erweitert.

```
vecAssembler = VectorAssembler(inputCols=["PayRate", "DaysWorked", "EngagementSurvey", "SpecialProjectsCount"], outputCol="
features")
new_df = vecAssembler.transform(df)
```

Abbildung 19 Vector Assembler Erweiterung für Random Forest

Anschließend wurde der Klassifizierungsalgorithmus (RandomForestClassifier) definiert und mithilfe diverser Einstellungen auf unseren Fall hin optimiert. Zu Beginn bestand der Wunsch, den Angestelltenstatus mithilfe des Random-Forest-Classifiers zu bestimmen. Hierbei stießen wir jedoch bei der Implementierung auf ähnliche Fehlerbilder wie beim zuvor angewandten Desicion-Tree. Die Fehlerursache bestand höchstwahrscheinlich darin, dass es sich beim Angestelltenstatus nicht um einen binären Wert handelt. Offensichtlich ist es auf unserer Datenbases nicht möglich, ein Feauture vorherzusagen, welches mehr als zwei Ausprägungen einnehmen kann. Daher entschied sich das Team das binäre Feature „Termd“ für die weitere Betrachtung zu verwenden. Diese Umstellung könnte auch beim Entscheidungsbaum ein möglicher Lösungsansatz gewesen sein, konnte jedoch aufgrund des begrenzten Zeitrahmens nicht näher verfolgt werden.

```
rf = RandomForestClassifier(labelCol="Termd", featuresCol="features",impurity="gini", \
                minInstancesPerNode=10, featureSubsetStrategy='sqrt', subsamplingRate=0.95, seed= 12345)
```

Abbildung 20 Random Forest Aufteilung Trainings- und Testdaten

Die Modellierungsphase wird mit dem Training des Random Forest abgeschlossen.

```
rfModel = rf.fit(training)
```

Abbildung 21 Random Forest Aufteilung Trainings- und Testdaten

Evaluation

Im Evaluationsschritt werden die erstellten Modelle und deren Ergebnisse bewertet und mit der zu Beginn definierten Aufgabenstellung abgeglichen. Ziel ist es, das für den Use-Case passendste Modell auszuwählen und gegebenenfalls zu optimieren.

Clustering

Zur Evaluierung der vom Algorithmus definierten Cluster werden im ersten Schritt die Testdaten visualisiert.

```
evaluate_model = model.transform(test)

pddf_pred = evaluate_model.toPandas().set_index('EmpID')
pddf_pred.head()
scatter = plt.scatter(pddf_pred.DaysWorked, pddf_pred.PayRate, c=pddf_pred.prediction)
plt.xlabel("DaysWorked")
plt.ylabel("$ per hour")
plt.show()
```

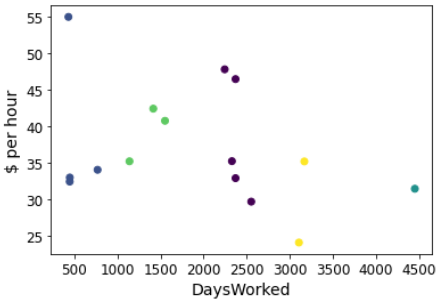


Abbildung 22 Evaluierung Clustering

Zusammen mit der Zuordnung zwischen Cluster und Angestelltenstatus wird die Vorhersage, ob ein Mitarbeiter das Unternehmen verlässt oder nicht, textuell ausgegeben.

So kann zum Beispiel aus der ersten Konsolenausgabe herausgelesen werden, dass die Mitarbeiterin „Sarah Akinkuolie“ mit einer Bezahlung von 32 $ pro Stunde und einer Unternehmenszugehörigkeit von 447 Tagen zur kündigungsanfälligen Gruppe gehört.

```
# Writing columns into dict
eval_dict = evaluate_model.toPandas().set_index('EmpID').T.to_dict('list')
#print(eval_dict)
job_status = ['will stay.', 'will terminate.', 'will stay.', 'will terminate.', 'will stay.']
for empID in eval_dict:
    print("{0} {1} earns {2:.0f} dollars per hour with {3:.0f} days worked and {4}"
          .format(
              eval_dict[empID][1],
              eval_dict[empID][0],
              eval_dict[empID][-5],
              eval_dict[empID][-6],
              job_status[int(eval_dict[empID][-1])]
          )
         )
```

```
Sarah Akinkuolie earns 32 dollars per hour with 447 days worked and will terminate.
Colby Andreola earns 48 dollars per hour with 2244 days worked and will stay.
Renee Becker earns 55 dollars per hour with 432 days worked and will terminate.
Mia Brown earns 32 dollars per hour with 4449 days worked and will stay.
David Gordon earns 24 dollars per hour with 3105 days worked and will stay.
Alfred Hitchcock earns 35 dollars per hour with 2328 days worked and will stay.
Ming Huynh earns 34 dollars per hour with 770 days worked and will terminate.
Lindsey Langford earns 33 dollars per hour with 448 days worked and will terminate.
Binh Le earns 41 dollars per hour with 1552 days worked and will terminate.
Giovanni Leruth earns 35 dollars per hour with 3168 days worked and will stay.
Louis Punjabhi earns 30 dollars per hour with 2552 days worked and will stay.
Andrew Szabo earns 46 dollars per hour with 2370 days worked and will stay.
Charlie Wang earns 42 dollars per hour with 1416 days worked and will terminate.
Jordan Winthrop earns 35 dollars per hour with 1140 days worked and will terminate.
Jason Woodson earns 33 dollars per hour with 2370 days worked and will stay.
```

Abbildung 23 Textuelle Ausgabe Clustering

Euklidische Distanz

Zur weiteren Evaluierung der Cluster wurde die euklidische Distanz berechnet. Diese repräsentiert den Abstand zwischen zwei Punkten als Strecke in einem Raum. In unserem Fall gibt diese Funktion die Distanz zwischen den einzelnen Clustermitten an.

Die Silhouette gibt zudem für eine Beobachtung an, wie gut die Zuordnung zu den beiden nächst gelegenen Clustern ist.

```
evaluator = ClusteringEvaluator()

silhouette = evaluator.evaluate(train_model)
print("Silhouette with squared euclidean distance = " + str(silhouette))

centers = model.clusterCenters()
print("Cluster Centers: ")
for center in centers:
    print(center)
```

```
Silhouette with squared euclidean distance = 0.6766397020968253
Cluster Centers:
[3.31744112e+01 2.33859813e+03 4.11728972e+00 1.43925234e+00]
[ 35.23352703 519.05405405   4.10594595   1.45945946]
[3.71466833e+01 3.85566667e+03 4.14733333e+00 9.33333333e-01]
[3.45861406e+01 1.52101562e+03 4.04421875e+00 1.14062500e+00]
[3.44007241e+01 3.03587931e+03 4.14086207e+00 6.20689655e-01]
```

Abbildung 24 Euklidische Distanz und Silhouette Clustering

Random Forest

Um den trainierten Random-Forest-Classifier bewerten zu können, werden nun die zuvor zurückgehaltenen Testdaten von einem BinaryClassificationEvaluator verwendet. Das dabei ausgegebene Ergebnis kann in Abbildung 25 eingesehen werden. Die Prediction stellt somit einen durch den Classifier vorhergesagten Wert für Termd dar.

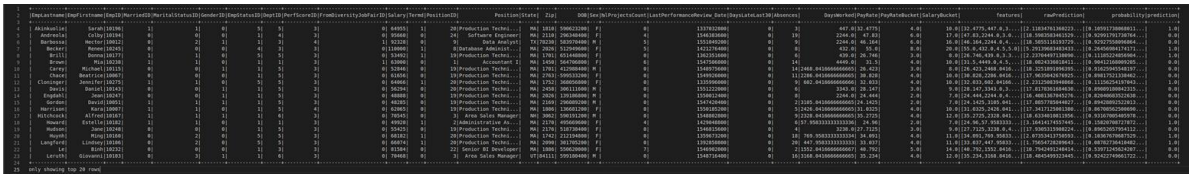


Abbildung 25 Random Forest Vorhersage

Um die Wahrscheinlichkeit der richtigen Vorhersage berücksichtigen zu können, wird diese vom Algorithmus zurückgegeben und in der Variablen „accuracy“ gespeichert. Hieraus wird anschließend die Fehlerwahrscheinlichkeit ausgerechnet und ausgegeben.

Mit einem Fehlerwahrscheinlichkeit von ca. 10 % gibt das trainierte Modell für den vorliegenden Anwendungsfall ein hinreichend aussagekräftiges Ergebnis.

```
predictions = rfModel.transform(test)
```

```
evaluator = BinaryClassificationEvaluator(labelCol="Termd",rawPredictionCol="rawPrediction", metricName="areaUnderROC")
accuracy = evaluator.evaluate(predictions)
print("Test Error",(1.0 - accuracy))
```

```
Test Error 0.10096153846153855
```

Abbildung 26 Euklidische Distanz und Silhouette Clustering

Deployment

Die Deployment-Phase bildet in der Regel die Endphase eines Data Mining-Projektes. Hier werden die gewonnenen Erkenntnisse so geordnet und präsentiert, dass der Auftraggeber dieses Wissen nutzen kann. Dazu gehört eine eventuelle Implementierungsstrategie, die Überwachung der Gültigkeit der Modelle, ein zusammenfassender Bericht und eine Präsentation.

Für eine mögliche Implementierung wurde ein pyspark-basierter Ansatz gewählt. Den Abschluss dieser Projektarbeit stellt die Vollendung dieses Berichtes dar.