KAPITEL 3

Dimensionsreduktion

In diesem Kapitel konzentrieren wir uns auf eine der großen Herausforderungen bei der Entwicklung von erfolgreichen Lösungen für angewandtes maschinelles Lernen: den Fluch der Dimensionalität. Unsupervised Learning hat ein großartiges Mittel gegen den Fluch – *Dimensionsreduktion*. In diesem Kapitel führen wir dieses Konzept ein und bauen darauf auf, sodass Sie ein Gespür dafür entwickeln können, wie das alles funktioniert.

In Kapitel 4 erstellen wir unsere eigene unüberwachte Lernlösung, die auf Dimensionsreduktion basiert – insbesondere ein System zur Erkennung von Kreditkartenbetrug, das mit Unsupervised Learning arbeitet (im Gegensatz zu dem System, das wir in Kapitel 2 mit Supervised Learning aufgebaut haben). Diese Art der unüberwachten Betrugserkennung ist die sogenannte Anomalieerkennung, ein schnell wachsender Bereich auf dem Gebiet des angewandten Unsupervised Learning.

Doch bevor wir ein Anomalieerkennungssystem entwickeln, machen wir Sie in diesem Kapitel mit der Dimensionsreduktion bekannt.

Die Motivation zur Dimensionsreduktion

Wie in Kapitel 1 erwähnt, hilft die Reduzierung der Dimensionalität – auch Dimensionsreduktion genannt –, einem der im maschinellen Lernen am häufigsten auftretenden Probleme entgegenzuwirken – dem Fluch der Dimensionalität, durch den Algorithmen aufgrund der schieren Größe des Featureraums weder effektiv noch effizient auf den Daten trainieren können.

Algorithmen zur Dimensionsreduktion projizieren hochdimensionale Daten in einen Raum mit weniger Dimensionen, wobei die wichtigen Informationen so gut wie möglich beibehalten und redundante Informationen entfernt werden. Wenn sich die Daten im Raum mit weniger Dimensionen befinden, sind maschinelle Lernalgorithmen in der Lage, interessante Muster effektiver und effizienter zu identifizieren, da das Rauschen zu einem Großteil verringert wurde.

Manchmal ist die Dimensionsreduktion das Ziel selbst – um zum Beispiel Anomalieerkennungssysteme zu erstellen, wie wir im nächsten Kapitel zeigen werden.

In anderen Anwendungen ist die Dimensionsreduktion kein Selbstzweck, sondern ein Mittel für einen anderen Zweck. So ist Dimensionsreduktion üblicherweise ein Teil der Pipeline für maschinelles Lernen, um große, rechenintensive Probleme mit Bildern, Videos, Sprache und Text zu lösen.

Die MNIST-Zifferndatenbank

Bevor wir die Algorithmen zur Dimensionsreduktion einführen, untersuchen wir das Dataset, mit dem wir in diesem Kapitel arbeiten. Es handelt sich dabei um ein einfaches Dataset für Computervision: die MNIST-Datenbank handgeschriebener Ziffern (MNIST steht für *Mixed National Institute of Standards and Technology*), eins der bekanntesten Datasets für maschinelles Lernen. Wir verwenden die Version des MNIST-Datasets, die auf der Website von Yann LeCun öffentlich zugänglich ist.[1] Zur Vereinfachung greifen wir auf die aufbereitete Version von *deeplearning.net* zurück.[2]

Diese Version des Datasets ist in drei Sets aufgeteilt worden – ein Trainingsset mit 50.000 Beispielen, ein Validierungsset mit 10.000 Beispielen und ein Testset mit 10.000 Beispielen. Sämtliche Beispiele haben wir mit Labels versehen.

Das Dataset besteht aus Bildern handgeschriebener Ziffern mit einer Größe von 28 × 28 Pixeln. Jeder einzelne Datenpunkt (d. h. jedes Bild) lässt sich als Array von Zahlen vermitteln, wobei jede Zahl beschreibt, wie dunkel jedes Pixel ist. Mit anderen Worten: Ein 28 × 28-Array mit Zahlen entspricht einem Bild mit 28 × 28 Pixeln.

Um eine einfachere Darstellung zu erhalten, können wir jedes Array zu einem Vektor mit 28 × 28 oder 784 Dimensionen abflachen. Jede Komponente des Vektors ist eine Gleitkommazahl zwischen 0 und 1, die die Intensität für jedes Pixel im Bild angibt. Der Wert 0 steht für Schwarz, der Wert 1 für Weiß. Die Labels sind Zahlen zwischen 0 und 9 und zeigen an, welche Ziffer das Bild darstellt.

Daten erfassen und untersuchen

Bevor wir mit den Algorithmen zur Dimensionsreduktion arbeiten, laden wir zuerst die erforderlichen Bibliotheken:

```
# Bibliotheken importieren
'''Main'''

import numpy as np
import pandas as pd
```

1 Siehe die Website *MNIST database of handwritten digits* (*http://yann.lecun.com/exdb/mnist/*), mit freundlicher Genehmigung von Yann LeCun.

2 Siehe die Website *http://deeplearning.net/tutorial/gettingstarted.html*, mit freundlicher Genehmigung von *deeplearning.net*.

```
import os, time
import pickle, gzip


'''Daten visualisieren'''
import matplotlib.pyplot as plt
import seaborn as sns
color = sns.color_palette()
import matplotlib as mpl

%matplotlib inline

'''Daten vorbereiten und Modell bewerten'''
from sklearn import preprocessing as pp
from scipy.stats import pearsonr
from numpy.testing import assert_array_almost_equal
from sklearn.model_selection import train_test_split
from sklearn.model_selection import StratifiedKFold
from sklearn.metrics import log_loss
from sklearn.metrics import precision_recall_curve, average_precision_score
from sklearn.metrics import roc_curve, auc, roc_auc_score
from sklearn.metrics import confusion_matrix, classification_report

'''Algorithmen'''
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier
import xgboost as xgb
import lightgbm as lgb
```

Die MNIST-Datasets laden

Laden Sie nun die MNIST-Datasets:

```
# Die Datasets laden
current_path = os.getcwd()
file = '\\datasets\\mnist_data\\mnist.pkl.gz'

f = gzip.open(current_path+file, 'rb')
train_set, validation_set, test_set = pickle.load(f, encoding='latin1')
f.close()

X_train, y_train = train_set[0], train_set[1]
X_validation, y_validation = validation_set[0], validation_set[1]
X_test, y_test = test_set[0], test_set[1]
```

Die Form der Datasets überprüfen

Überprüfen Sie als Nächstes die Form der Datasets, um sich davon zu überzeugen, dass sie richtig geladen wurden:

```
# Die Form der Datasets verifizieren
print("Shape of X_train: ", X_train.shape)
print("Shape of y_train: ", y_train.shape)
print("Shape of X_validation: ", X_validation.shape)
print("Shape of y_validation: ", y_validation.shape)
print("Shape of X_test: ", X_test.shape)
print("Shape of y_test: ", y_test.shape)
```

Die Ausgabe des obigen Codes bestätigt, dass die Datasets wie erwartet aussehen:

```
Shape of X_train:       (50000, 784)
Shape of y_train:       (50000,)
Shape of X_validation:  (10000, 784)
Shape of y_validation:  (10000,)
Shape of X_test:        (10000, 784)
Shape of y_test:        (10000,)
```

Pandas-DataFrames aus den Datasets erzeugen

Konvertieren wir nun die numpy-Arrays in Pandas-DataFrames, sodass es leichter ist, sie zu untersuchen und mit ihnen zu arbeiten:

```
# Pandas-DataFrames aus den Datasets erzeugen
train_index = range(0,len(X_train))
validation_index = range(len(X_train), \
                         len(X_train)+len(X_validation))
test_index = range(len(X_train)+len(X_validation), \
                   len(X_train)+len(X_validation)+len(X_test))

X_train = pd.DataFrame(data=X_train,index=train_index)
y_train = pd.Series(data=y_train,index=train_index)

X_validation = pd.DataFrame(data=X_validation,index=validation_index)
y_validation = pd.Series(data=y_validation,index=validation_index)

X_test = pd.DataFrame(data=X_test,index=test_index)
y_test = pd.Series(data=y_test,index=test_index)
```

Die Daten untersuchen

Der folgende Code erzeugt eine Zusammenfassungsansicht der Daten:

```
# Die Trainingsmatrix beschreiben
X_train.describe()
```

Tabelle 3-1: Datenuntersuchung

	0	1	2	3	4	5	6
count	50000.0	50000.0	50000.0	50000.0	50000.0	50000.0	50000.0
mean	0.0	0.0	0.0	0.0	0.0	0.0	0.0
std	0.0	0.0	0.0	0.0	0.0	0.0	0.0
min	0.0	0.0	0.0	0.0	0.0	0.0	0.0
25%	0.0	0.0	0.0	0.0	0.0	0.0	0.0
50%	0.0	0.0	0.0	0.0	0.0	0.0	0.0
75%	0.0	0.0	0.0	0.0	0.0	0.0	0.0
max	0.0	0.0	0.0	0.0	0.0	0.0	0.0

8 rows x 784 columns

Tabelle 3-1 zeigt eine Zusammenfassungsansicht der Bilddaten. Viele Werte sind Nullen – d. h., die meisten Pixel in den Bildern sind schwarz. Das ist verständlich, da die Ziffern weiß sind und in der Mitte des Bilds auf einem schwarzen Hintergrund erscheinen.

Die Label-Daten stehen in einem eindimensionalen Vektor, der den eigentlichen Bildinhalt darstellt. Die Labels für die ersten Bilder lauten wie folgt:

```
# Die Labels anzeigen
y_train.head()
```

```
0  5
1  0
2  4
3  1
4  9
dtype: int64
```

Die Bilder anzeigen

Um das Bild zusammen mit seinem Label anzuzeigen, definieren wir folgende Funktion:

```
def view_digit(example):
    label = y_train.loc[0]
    image = X_train.loc[example,:].values.reshape([28,28])
    plt.title('Example: %d Label: %d' % (example, label))
    plt.imshow(image, cmap=plt.get_cmap('gray'))
    plt.show()
```

Eine Ansicht des ersten Bilds – nachdem der 784-dimensionale Vektor in ein 28 x 28-Pixel-Bild umgeformt wurde – zeigt die Ziffer Fünf (siehe Abbildung 3-1).

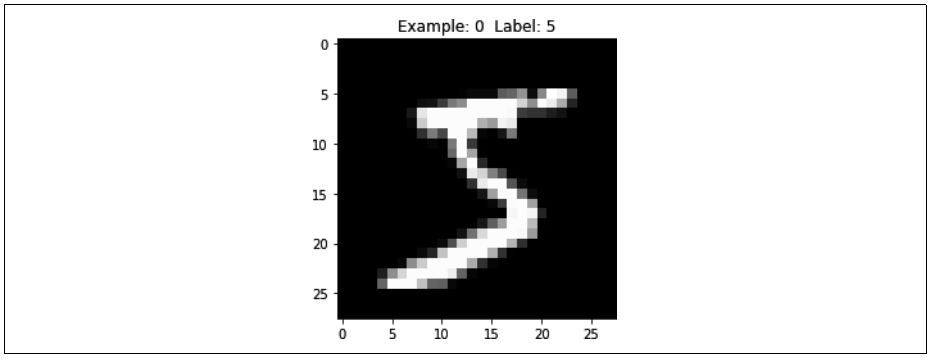

Abbildung 3-1: Ansicht der ersten Ziffer

Algorithmen zur Dimensionsreduktion

Nachdem wir das MNIST-Ziffern-Dataset geladen und untersucht haben, geht es jetzt um Algorithmen zur Dimensionsreduktion. Für jeden Algorithmus führen wir

zuerst das Konzept ein und vertiefen dann unser Verständnis dafür, indem wir den Algorithmus auf das MNIST-Ziffern-Dataset anwenden.

Lineare Projektion vs. Manifold Learning

Es gibt zwei Hauptzweige der Dimensionsreduktion. Der erste ist als *lineare Projektion* bekannt und projiziert Daten aus einem hochdimensionalen Raum in einen Raum mit weniger Dimensionen. Dazu gehören Techniken wie *Hauptkomponentenanalyse*, *Singulärwertzerlegung* und *Zufallsprojektion*.

Der zweite Zweig ist das sogenannte *Manifold Learning*, das auch als *nicht lineare Dimensionsreduktion* bezeichnet wird. Hierzu gehören Techniken wie *Isomap*, die den gekrümmten Abstand (auch *geodätischer Weg* genannt) zwischen zwei Punkten statt des euklidischen Abstands lernt. Andere Techniken sind *multidimensionale Skalierung* (*MDS*), *lokal lineare Einbettung* (*LLE*), *stochastische Nachbarschaftseinbettung mit Student-t-Verteilung* (*t-SNE*), *Dictionary Learning*, *Random-Trees-Einbettung* und *Unabhängigkeitsanalyse*.

Hauptkomponentenanalyse

Wir untersuchen mehrere Versionen der Hauptkomponentenanalyse (PCA), einschließlich Standard-PCA, inkrementelle PCA, sparse PCA und Kernel-PCA.

Hauptkomponentenanalyse, das Konzept

Beginnen wir mit Standard-PCA, einer der am häufigsten verwendeten Techniken der linearen Dimensionsreduktion. In PCA sucht der Algorithmus eine niederdimensionale Repräsentation der Daten, wobei möglichst viele der Variationen (d. h. wichtige Informationen) beibehalten werden.

PCA betrachtet hierzu die Korrelation zwischen den Features. Wenn die Korrelation in einer Teilmenge der Features sehr hoch ist, versucht PCA, die stark korrelierten Features zusammenzufassen und diese Daten mit einer kleineren Anzahl von linear unkorrelierten Features zu repräsentieren. Der Algorithmus führt diese Reduzierung der Korrelation fort, wobei er nach den Richtungen der maximalen Varianz in den ursprünglich hochdimensionalen Daten sucht und sie in einen Raum mit weniger Dimensionen projiziert. Diese neu abgeleiteten Komponenten werden als Hauptkomponenten bezeichnet.

Mit diesen Komponenten lassen sich die ursprünglichen Features rekonstruieren – zwar nicht genau, aber im Allgemeinen nahe genug. Der PCA-Algorithmus versucht während seiner Suche nach den optimalen Komponenten, den Rekonstruktionsfehler aktiv zu minimieren.

In unserem MNIST-Beispiel hat der ursprüngliche Featureraum 784 Dimensionen, als d Dimensionen bezeichnet. PCA projiziert nun die Daten auf einen kleineren

Teilraum von k Dimensionen (wobei $k < d$), während möglichst viele der wichtigen Informationen beibehalten werden. Diese k Dimensionen sind die sogenannten Hauptkomponenten.

Die Anzahl der für uns verbleibenden bedeutungsvollen Hauptkomponenten ist beträchtlich kleiner als die Anzahl der Dimensionen im ursprünglichen Dataset. Beim Übergang in diesen Raum mit weniger Dimensionen haben wir etwas Varianz (d.h. Informationen) eingebüßt, doch die zugrunde liegende Struktur der Daten lässt sich leichter identifizieren. Dadurch können wir auch Aufgaben wie Anomalieerkennung und Clustering effektiver und effizienter ausführen.

Durch die reduzierte Dimensionalität verringert PCA auch die Größe der Daten. Das wiederum verbessert die Performance der maschinellen Lernalgorithmen in der maschinellen Lernpipeline (zum Beispiel für Aufgaben wie die Bildklassifizierung).

Es ist wichtig, vor der Ausführung der Hauptkomponentenanalyse die Features zu skalieren. PCA ist sehr empfindlich gegenüber den relativen Bereichen der ursprünglichen Features. Wir müssen die Daten im Allgemeinen skalieren, um sicherzustellen, dass sich die Features im gleichen relativen Bereich befinden. Da aber für unser MNIST-Ziffern-Dataset die Features bereits in einen Bereich von 0 bis 1 skaliert wurden, können wir diesen Schritt überspringen.

PCA in der Praxis

Da Sie nun eine bessere Vorstellung davon haben werden, wie PCA funktioniert, wenden wir PCA auf das MNIST-Ziffern-Dataset an und sehen, wie PCA die wichtigsten Informationen über die Ziffern erfasst, wenn der Algorithmus die Daten aus dem ursprünglichen 784-dimensionalen Raum in einen Raum mit weniger Dimensionen projiziert.

Die Hyperparameter festlegen

Der folgende Code legt die Hyperparameter für den PCA-Algorithmus fest:

```
from sklearn.decomposition import PCA

n_components = 784
whiten = False
random_state = 2018

pca = PCA(n_components=n_components, whiten=whiten, \
    random_state=random_state)
```

PCA anwenden

Die Anzahl der Hauptkomponenten legen wir auf die ursprüngliche Anzahl von Dimensionen fest (d.h. 784). Dann erfasst die Hauptkomponentenanalyse die wichtigen Informationen von den ursprünglichen Dimensionen und beginnt, die Hauptkomponenten zu erzeugen. Sobald diese Komponenten erzeugt sind, bestimmen

wir, wie viele Hauptkomponenten wir benötigen, um die größte Varianz (d.h. die meisten Informationen) aus diesem ursprünglichen Featureset zu erfassen.

Wir passen nun unsere Trainingsdaten an und transformieren sie, um diese Hauptkomponenten zu generieren:

```
X_train_PCA = pca.fit_transform(X_train)
X_train_PCA = pd.DataFrame(data=X_train_PCA, index=train_index)
```

PCA bewerten

Da wir die Dimensionalität überhaupt noch nicht reduziert (sondern lediglich die Daten transformiert) haben, sollte die Varianz (die Informationen) der ursprünglichen Daten, die von den 784 Hauptkomponenten erfasst wurde, 100% betragen:

```
# Prozentuale Varianz, die von 784 Hauptkomponenten erfasst wurde
print("Variance Explained by all 784 principal components: ", \
    sum(pca.explained_variance_ratio_))

Variance Explained by all 784 principal components: 0.9999999999999997
```

Beachten Sie aber, dass die Wichtigkeit der 784 Hauptkomponenten sehr unterschiedlich ist. Die Wichtigkeiten der ersten X Hauptkomponenten werden hier summiert:

```
# Prozentuale Varianz, die von X Hauptkomponenten erfasst wurde
importanceOfPrincipalComponents = \
    pd.DataFrame(data=pca.explained_variance_ratio_)
importanceOfPrincipalComponents = importanceOfPrincipalComponents.T

print('Variance Captured by First 10 Principal Components: ',
    importanceOfPrincipalComponents.loc[:,0:9].sum(axis=1).values)
print('Variance Captured by First 20 Principal Components: ',
    importanceOfPrincipalComponents.loc[:,0:19].sum(axis=1).values)
print('Variance Captured by First 50 Principal Components: ',
    importanceOfPrincipalComponents.loc[:,0:49].sum(axis=1).values)
print('Variance Captured by First 100 Principal Components: ',
    importanceOfPrincipalComponents.loc[:,0:99].sum(axis=1).values)
print('Variance Captured by First 200 Principal Components: ',
    importanceOfPrincipalComponents.loc[:,0:199].sum(axis=1).values)
print('Variance Captured by First 300 Principal Components: ',
    importanceOfPrincipalComponents.loc[:,0:299].sum(axis=1).values)

Variance Captured by First 10 Principal Components: [0.48876238]
Variance Captured by First 20 Principal Components: [0.64398025]
Variance Captured by First 50 Principal Components: [0.8248609]
Variance Captured by First 100 Principal Components: [0.91465857]
Variance Captured by First 200 Principal Components: [0.96650076]
Variance Captured by First 300 Principal Components: [0.9862489]
```

Die ersten zehn Komponenten erfassen insgesamt etwa 50% der Varianz, die ersten 100 Komponenten über 90% und die ersten 300 Komponenten fast 99% der Varianz; die Informationen in den übrigen Hauptkomponenten können vernachlässigt werden.

Die Wichtigkeit jeder Hauptkomponente kann man auch grafisch darstellen, geordnet von der ersten zur letzten Hauptkomponente. Damit die Übersicht nicht verloren geht, zeigt Abbildung 3-2 nur die ersten zehn Komponenten.

Die Leistungsfähigkeit der Hauptkomponentenanalyse sollte damit deutlicher werden. Mit nur den ersten 200 Hauptkomponenten (weit weniger als die ursprünglichen 784 Dimensionen) erfassen wir über 96 % der Varianz beziehungsweise Informationen.

PCA ermöglicht uns, die Dimensionalität der ursprünglichen Daten erheblich zu reduzieren, dabei aber die meisten der wichtigen Informationen beizubehalten. Im Featureset, das durch PCA reduziert wurde, ist es für andere Algorithmen – die in der Pipeline des maschinellen Lernens nachgelagert sind – einfacher, die Datenpunkte im Raum zu trennen (d.h. Aufgaben wie Anomalieerkennung und Clustering auszuführen) und dabei mit weniger Rechenressourcen auszukommen.

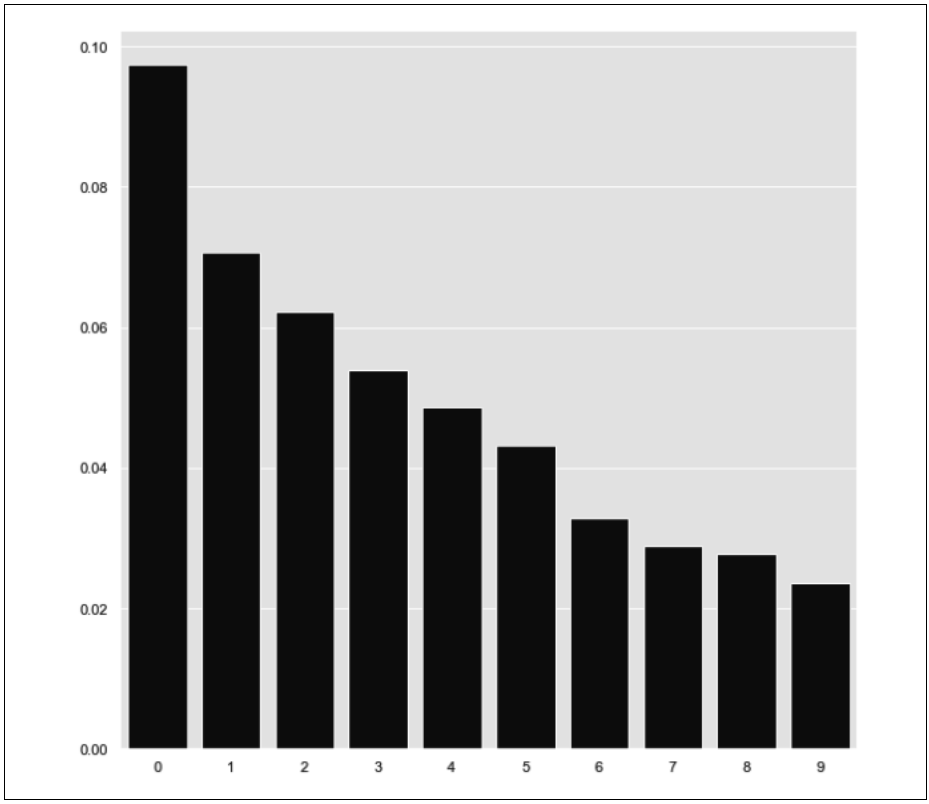

Abbildung 3-2: Wichtigkeit der PCA-Komponenten

Die Trennung der Punkte im Raum visualisieren

Um zu demonstrieren, wie leistungsfähig die Hauptkomponentenanalyse ist, um effizient und kompakt die Varianz (die Informationen) in den Daten zu erfassen,

stellen wir die Beobachtungen in zwei Dimensionen dar. Hierzu zeichnen wir ein *Streudiagramm* der ersten und zweiten Hauptkomponente und markieren die Beobachtungen mit dem wahren Label. Zu diesem Zweck erstellen wir eine Funktion `scatterPlot()`, da wir später auch Visualisierungen für die anderen Dimensionalitätsalgorithmen darstellen wollen:

```
def scatterPlot(xDF, yDF, algoName):
    tempDF = pd.DataFrame(data=xDF.loc[:,0:1], index=xDF.index)
    tempDF = pd.concat((tempDF,yDF), axis=1, join="inner")
    tempDF.columns = ["First Vector", "Second Vector", "Label"]
    sns.lmplot(x="First Vector", y="Second Vector", hue="Label", \
               data=tempDF, fit_reg=False)
    ax = plt.gca()
    ax.set_title("Separation of Observations using "+algoName)

scatterPlot(X_train_PCA, y_train, "PCA")
```

Wie aus Abbildung 3-3 hervorgeht, ist die Hauptkomponentenanalyse allein mit zwei Hauptkomponenten in der Lage, die Punkte im Raum einwandfrei zu trennen, sodass der Abstand zwischen ähnlichen Punkten im Allgemeinen kleiner ist als zu anderen Punkten, die weniger ähnlich sind. Mit anderen Worten: Bilder derselben Ziffer liegen enger beieinander, während die Bilder anderer Ziffern weiter entfernt sind.

PCA erreicht dies, ohne irgendwelche Labels zu verwenden. Dies zeigt die Leistungsfähigkeit des Unsupervised Learning, um die zugrunde liegende Struktur der Daten zu erfassen, wodurch sich verdeckte Muster beim Fehlen von Labels leichter entdecken lassen.

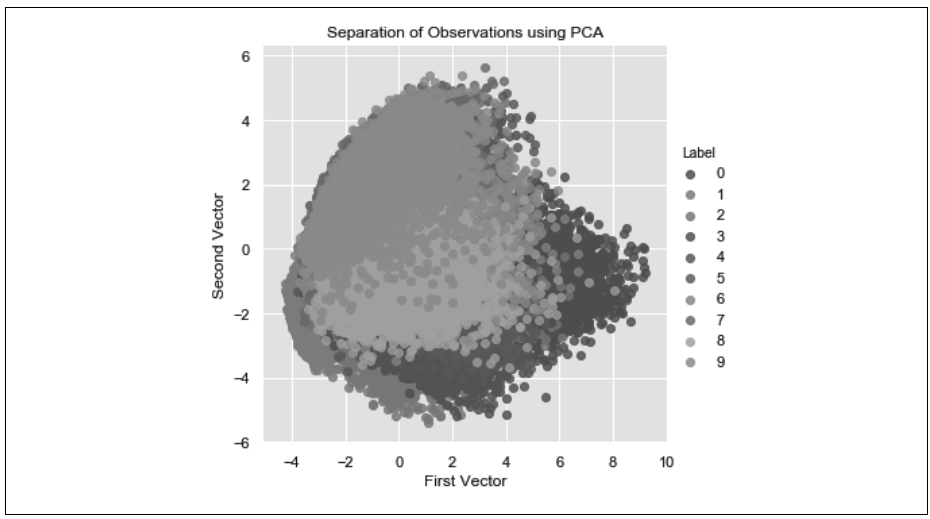

Abbildung 3-3: Trennung von Beobachtungen mithilfe der Hauptkomponentenanalyse

Wenn wir das gleiche zweidimensionale Streudiagramm mit den beiden wichtigsten Features aus der ursprünglichen 784-Feature-Menge – bestimmt durch das

Training eines überwachten Lernmodells – darstellen, ist die Trennung bestenfalls schlecht (siehe Abbildung 3-4).

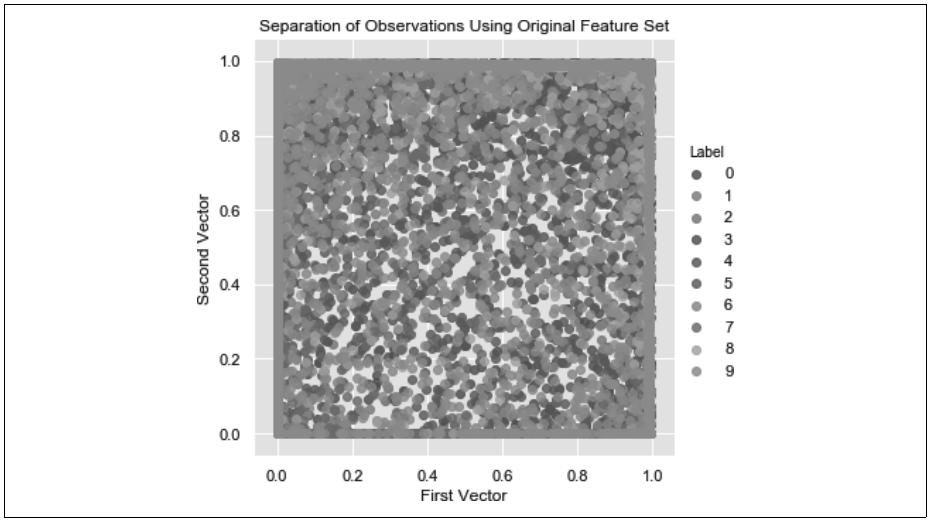

Abbildung 3-4: Trennung von Beobachtungen ohne Hauptkomponentenanalyse

Der Vergleich zwischen Abbildung 3-3 und Abbildung 3-4 zeigt, wie leistungsfähig PCA dahin gehend ist, die zugrunde liegende Struktur des Datasets ohne irgendwelche Labels zu lernen – selbst mit nur zwei Dimensionen können wir die Bilder nach den angezeigten Ziffern sinnvoll trennen.

Die Hauptkomponentenanalyse ist nicht nur dabei behilflich, die Daten zu trennen, damit wir versteckte Muster leichter entdecken können, sie hilft auch, die Größe der Featuremenge zu verringern, wodurch es – sowohl hinsichtlich der Zeit als auch in Bezug auf die rechentechnischen Ressourcen – kostengünstiger wird, Modelle des maschinellen Lernens zu trainieren.

Mit dem MNIST-Dataset wird die Verringerung der Trainingszeit bestenfalls moderat ausfallen, da das Dataset sehr klein ist – wir haben nur 784 Features und 50.000 Beobachtungen. Aber bei einem Dataset mit Millionen von Features und Milliarden von Beobachtungen würde durch die Dimensionsreduktion die Trainingszeit der Algorithmen in der Pipeline für maschinelles Lernen erheblich verkürzt.

Schließlich verwirft PCA einige Informationen, die im ursprünglichen Featureset vorhanden sind, tut dies aber so raffiniert, dass die wichtigsten Elemente erfasst und die weniger wertvollen geopfert werden. Zwar schneidet ein Modell, das auf einem per PCA reduzierten Featureset trainiert wird, hinsichtlich Genauigkeit möglicherweise nicht so gut ab wie ein Modell, das auf dem gesamten Featureset trainiert wird, doch laufen sowohl das Training als auch die Vorhersage wesentlich schneller ab. Dies ist einer der wichtigen Kompromisse, die Sie bei der Entscheidung berücksichtigen müssen, ob Sie Dimensionsreduktion in Ihrem Produkt des maschinellen Lernens einsetzen wollen.

Inkrementelle Hauptkomponentenanalyse

Für sehr große Datasets, die nicht in den Arbeitsspeicher passen, lässt sich die Hauptkomponentenanalyse inkrementell in kleinen Batches durchführen, wobei jeder Batch vollständig im Arbeitsspeicher untergebracht werden kann. Die Batch-Größe kann entweder manuell festgelegt oder automatisch bestimmt werden. Diese Batch-basierte Form der PCA bezeichnet man als *inkrementelle PCA*. Die resultierenden Hauptkomponenten von PCA und inkrementeller PCA sind im Allgemeinen recht ähnlich (siehe Abbildung 3-5). Der Code für inkrementelle PCA sieht folgendermaßen aus:

```
# Inkrementelle PCA
from sklearn.decomposition import IncrementalPCA

n_components = 784
batch_size = None

incrementalPCA = IncrementalPCA(n_components=n_components, \
                                batch_size=batch_size)

X_train_incrementalPCA = incrementalPCA.fit_transform(X_train)
X_train_incrementalPCA = \
    pd.DataFrame(data=X_train_incrementalPCA, index=train_index)
X_validation_incrementalPCA = incrementalPCA.transform(X_validation)
X_validation_incrementalPCA = \
    pd.DataFrame(data=X_validation_incrementalPCA, index=validation_index)

scatterPlot(X_train_incrementalPCA, y_train, "Incremental PCA")
```

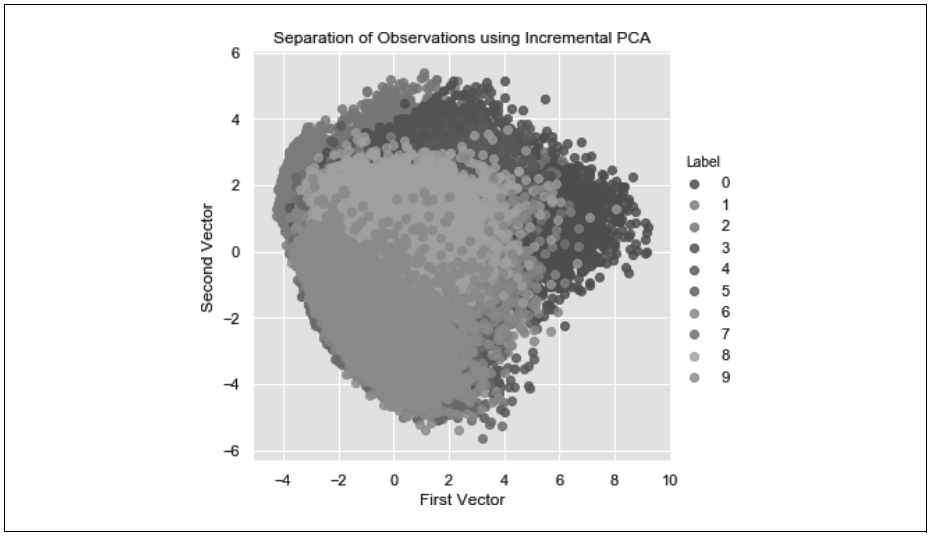

Abbildung 3-5: Trennung von Beobachtungen mittels inkrementeller PCA