Примена на линеарна алгебра за намалување на димензионалноста со помош на Principal Component Analysis

Александар Ивановски
Универзитет Св. Кирил и Методиј во Скопје
Факултет за информатички науки и компјутерско инженерство
Северна Македонија
Email: aleksandar.ivanovski.1@students.finki.ukim.mk

Концептите обработувани во предметот линеарна алгебра и примени, ќе се обидеме да ги примениме за решавање на еден доста присутен проблем во машинско учење и наука за податоци, а тоа е проблемот на голема димензионалност на податоците.

1 Опис

Во рамки на оваа семинарска работа ќе се фокусираме на еден апликативен аспект кој што ги применува дел од концептите кои беа изучувани во текот на предметот линеарна алгебра и примени.

Имено ќе го дискутираме проблемот на голема димензионалност на податоците, кој што проблем е доста присутен во машинско учење и во науката за податоци. Ќе дискутираме што всушност претставува димензионалноста на податоците, зошто и кога е потребно да се намали истата и како и кои концепти од линеарна алгебра може да се искористат за намалување на димензионалноста на податоците, а притоа да се задржат што е можно повеќе од оригиналните карактеристики на податоците. Детално ќе го разгледаме и проучиме методот Principal Component Analysis (PCA), ќе ги покажеме неговите карактеристики и како истиот функционира, и ќе покажеме практична примена на истиот во Python или R над вистинско високо-димензионално податочно множество.

2 Димензионалност на податоците

2.1 Дефинирање

Пред да навлеземе во детали, да дефинираме што точно претставува димензионалноста во контекст на линеарна алгреба.

Во набљудувано и ненабљудувано учење најчесто имаме податочно множество, од n примероци, каде што за секој примерок имаме m карактеристики. Секој примерок од аспект на линеарна алгебра претставува вектор од m-димензионален векторски простор, а имаме n такви примероци. Карактеристиките на податоците може да бидат од произволен податочен тип, но после процесот на нивно енкодирање, којшто е дел од фазата на пред-процесирање на податоците, вектортскиот простор на примероците е најчесто R^n над полето R.

Целото податочно множество може да биде репрезентирано со помош на матрица со димензии $m \times n$, односно матрица со димензија на колонскиот простор еднаква на димензијата на векторскиот простор во кој се наоѓаат векторите, и редичен простор со димензии еднаков на бројот на вектори.

Како заклучок, димензионалноста на податоците може да ја дефинираме како димензијата на векторскиот простор во кој припаѓаат векторите со кои се репрезентирани примероците од податочното множество.

2.2 Потреба од намалување на димензионалноста

Голема димензионалност претставуваат податочни множества со стотици, илјадници, па некогаш и милиони карактеристики.

Помалку карактеристики, често претставува помалку параматри и поедноставна структура на моделот за машинско учење, уште наречено степени на слобода. Модел со многу степени на слобода има голема веројатност да направи overfit на тренрирачкото множество, и би имал слаби перформанси при тестирање.

Општа цел во машинско учење е да се има поедноставни модели кои добро генерализираат, што значи ни требаат податоци со помалку карактерстики, т.е. помала димензија на векторскиот простор на кој припаѓаат векторите кои ги репрезенираат тие податоци. За да се постигне оваа цел, потребно е да ја намалиме димензионалноста на податочното множество.

Голем дел од науката за податоци се занимава со визуелизација на податоците. Интересен феномен претставува податочното множество наречено Anscombe's quartet, кое всушност ја потенцира важноста на визуелизацијата, бидејќи 4 различни податочни множества кои имаат исти дескриптивни статистики, нивната визуелизација покажува тотално други процеси и зависности меѓу податоците. Од аспект на визуелизација, ние луѓето сме ограничени до 3 димензии, па овде се поставува прашањето како да се визуелизира податочно множество со повеќе од 3 карактеристики. Еден одговор на тоа прашање се наоѓа точно во намалувањето на димензионалноста. Имено произволно податочно множество од n-димензии може да се намали до 3 димензии, и потоа истото да биде визуелизирано во 3Д простор.

Важна напомена која произлегува од дискусијата е тоа што, сакаме после намалувањето на димензионалноста да изгубиме што е можно помалку од зависностите и карактеристиките. Идеално е векторскиот простор со помлаку димензии да биде изоморфен со оригиналниот векторски простор, но тоа е невозможно, па затоа проблемот се сведува на намалување на загубата на карактеристики, т.е. станува збор за оптимизациски проблем.

Друг интересен феномен, потенциран од Richard E. Bellman претставува „проклетството на димензионалноста“ *(анг. the curse of dimensionality)*. Еден дел од овој феномен се однесува на евклидовото растојание во високо димензионален векторски простор, имено како што се зголемува бројот на димензии, примерцоите се натрупуваат кон еден мал дел од тој високо димензионален простор, и растојанието меѓу било кои два произволни примероци тежнее кон ист број, и самиот концепт на растојание го губи неговото значење.

3 Principal Component Analysis

3.1 Општ преглед

Principle Component Analysis (PCA) претставува еден метод за намалување на димензионалноста, со цел да се задржат што е можно повеќе од оригиналните карактеристики. Користејќи PCA се прави компромис меѓу точноста на моделот и неговата едноставност.

Клучен дел од PCA претставува искористување на концептите на ортогоналност и сопствени вредности и вектори применети врз матрица на коваријанси, која матрица произлегува од векторите кои го репрезентираат податочното множество.

Бидејќи овој метод всушност претставува алгоритам, да ги разгледаме неговите карактеристики.

- **Влез** во алгортимот е оригиналното високо-димензионално податочно множество (матрица),
- **Излез** од алгоритмот е ново добиено податочно множество со помала димензија од влезното (матрица),

- **Временската комплексност** изнесува $O(min(p^3, n^3))$, каде n е бројот на вектори, а p е димензијата на векторскиот простор во кој се наоѓаат n-те вектори,
- **Просторната комплексност** изнесува $O(np)$, каде n е бројот на вектори, а p е димензијата на векторскиот простор во кој се наоѓаат n-те вектори.

Алгоритмот се состои од 5 чекори, кои детално ќе бидат објаснети во следното поглавие.

3.2 Детално проучување на методот

Клучен дел од PCA, претставуваат principle components *(скратно: PCs)*, што всушност претставуваат вектори кои се линеарна комбинација од карактеристики со оптималани тежини. Бројот на principle components е помал или еднаков од иницијалната димензионалност на податоците. Дел од својствата на PCs, се следниве:

- PCs се ортогонални вектори,
- Варијансата присутна во PCs се намалува како што се движиме од првиот principle component кон последниот. Варијансата се користи како метрика за значење (анг. importance measure) на PCs.

Principle component-от со најмало значење може да биде искористен во регресија или детекција на outlier-и.

Да ги разгледаме чекорите на алгоритмот.

Чекор 1: Стандардизација

Како што ќе биде дискутирано во следните чекори, од исклучителна важност е вредностите на секоја карактеристика [1] да бидат споредливи едни со други, односно сакаме варијансите на карактеристиките да бидат споредливи меѓусебно. Демонстрација на овој проблем е карактеристика со вредности од 0 до 100, секогаш ќе има поголема варијанса од карактеристики со вредности од 0 до 1, па таа секогаш ќе доминира во линеарните комбинации при градење на principle components.

Математичко решение на овој проблем, претставува стандардизација на податоците, односно одземање на просекот и делење со стандардната девијација за секој примерок од секоја карактеристика.

$$z = \frac{value - mean}{standarddeviation}$$

После стандардизацијата сите променливи ќе бидат споредливи меѓусебно. Стандардизираните податоци ќе бидат сместени во матрица $StandardizedDataSet$. Каде што колони се карактеристиките, а редици се вредностите за тие карактеристики за секој примерок.

Чекор 2: Пресметка на матрица на коваријанси

Целта на овој чекор е да увидеме како карактеристиките се разликуваат од нивниот просек, и тоа да се спореди меѓу сите карактеристики. Односно, сакаме да видиме дали постои некаква зависност меѓу различните карактеристики, бидејќи понекогаш зависноста меѓу две карактеристики е толку голема, што едната од нив е непотребна. За да ги забележиме овие зависности, ја пресметуваме матрицата на коваријанси.

Матрицата на коваријанси претставува матрица со димензии $p \times p$, каде p е бројот на димензии. Оваа матрица ги содржи коваријансите меѓу сите парови од карактеристики. Имено истата е од следниот облик:

[1] Да забележиме дека термините карактеристика и променлива ќе бидат користени најзменично со истото значење и во истиот контекст.

$$CovMatrix = \begin{bmatrix} Cov(x_1, x_1) & Cov(x_1, x_2) & \dots & Cov(x_1, x_p) \\ Cov(x_2, x_1) & Cov(x_2, x_2) & \dots & Cov(x_2, x_p) \\ \vdots & \vdots & \dots & \vdots \\ Cov(x_p, x_1) & Cov(x_p, x_2) & \dots & Cov(x_p, x_p) \end{bmatrix}$$

Да забележиме дека коваријанста меѓу две променливи a и b, се пресметува со следнава формула:

$$Cov(a, b) = \frac{\sum(a_i - \overline{a})(b_i - \overline{b})}{N-1}$$

каде a_i е вредност на променливата a, b_i е вредност на променливата b, $\overline{a}$ е просекот на променливата a, $\overline{b}$ е просекот на променливата b и N е бројот на примероци (вредности на променливите).

Од $Cov(a, a) = Var(a)$, по главната дијагонала ќе ги имаме варијансите на секоја променлива, и исто така од комутативноста на коваријансата $Cov(a, b) = Cov(b, a)$, следува дека оваа матрица ќе биде симетрична, со оска на симетрија - главната дијагонала.

Чекор 3: Пресметка на сопствени вредности и вектори матрицата на коваријанси

Како што беше потенцирано неколкупати до сега, клучна цел е да ги одредиме principal components. За нивно одредување ќе ги искористиме сопствените вредности и вектори на матрицата на коваријанси.

Principal components претставуваат нови променливи добиени со линеарна комбинација на постоечките променливи со кои се репрезентираат податоците. Линеарната комбинација е изведена на начин што новите променливи - principle components се не корелирани и најмногу од информациите од оригиналните податоци се репрезентирани со привте principle components. Идејата е дека од m димензионален простор на податоците, ќе добиеме m principal components, но со PCA, ќе се обидеме најголемиот број од спецификите на податоците да ги репрезенитраме со помош на првиот principal component, па остатокот со вториот, и така натаму додека не достигнеме праг на прифатливост.

Организацијата на податоците со помош на principle components ни дозволува да ја намалиме нивната димензионалност, со максимално задржување на нивните карактеристики.

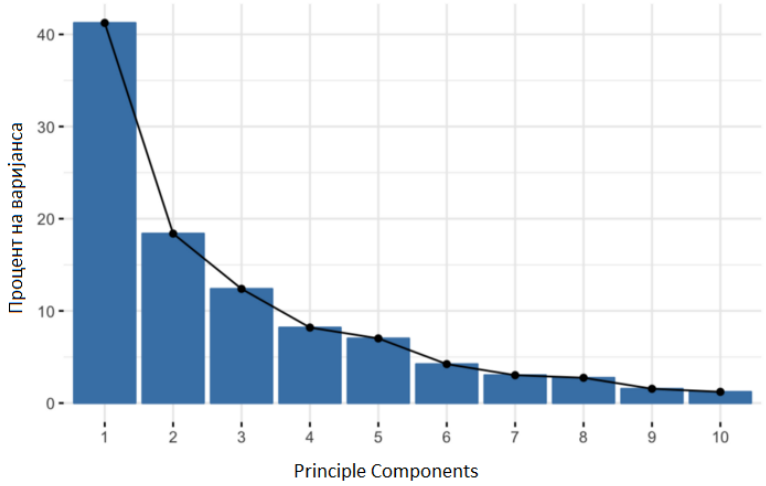
Овде треба да се потенцира дека principle components претставуваат поголем предизвик за интерпретација, бидејќи истите се всушност линеарна комбинација од оригиналните податоци.

Првиот principle component се пресметува на начин што се одбира векторот на кој проекцијата на податоците е максимално распространета, односно онаа проекција која што ја максимизира варијансата (просек од квадратно растојание меѓу проекциите на податоците). Вториот principle component се пресметува на истиот начин со разлика во тоа што е некорелиран, т.е. е ортогонален на првиот. Овој процес се повтара се додека вкупно p principle components не се пресметани.

Во Слика 1 е претставен процентот на варијансата која ја репрезентира секој principle component за податоци репрезентирани со вектори од 10-димензионален векторски простор.

Сопствените вектори и вредности ни помагаат во овој процес, имено сопствените вектори на матрицата на коваријанси се самите principle components - оските во кои што има најмногу варијанса.

Подредувањето на сопствените вектори според соодветните сопствени вредности во опаѓачки редослед ни ги дава principle components подредени според нивното значење.



1: Дистрибуција на варијансата низ princple components

Чекор 4: Избор на компоненти и добивање на feature вектор

Во овој чекор одбираме кои од principle components ќе ги задржеме, а кои ќе ги отфрлиме. За таа цел најчесто се дефнира праг на значајност, или едноставно априори се одредува димензиајта до која сакаме да ги намалиме оригиналните податоци.

Principle component-ите кои ќе ги задржеме се запишуваат како колони на матрица, и таа матрица се нарекува feature vector.

$$FeatureVector = \begin{bmatrix} v_1 & v_2 & \dots & v_k \end{bmatrix}$$

каде што $k \leq p$, а v_i се principle component-ите.

Чекор 5: Намалување на димензијата

Намалувањето на димензијата всушност претставува линеарна трансформација од оригиналниот векторски простор во кои се векторите кои ги репрезентираат податоците, во пониско димензионален векториски простор. Векторската репрезентација на податоците во пониско димензионалниот векторски простор ќе бидат запишени со матрицата $FinalDataSet$.

Оваа дискусија може да ја формализираме на следниот начин:

$$FinalDataSet = FeatureVector^T \times StandardizedDataSet^T$$

4 Практична примена

[1]:
```
import pandas as pd
import numpy as np
from sklearn.preprocessing import StandardScaler
import plotly.express as px
```

Во прилог ќе бидат практично применети концептите за намалување на димензионалноста со помош на Principal Component Analysis (PCA).

4.1 Дефинирање на податочното множество

Податочното множество кое ќе биде користено е Heart Disease UCI, достапно на следниот линк. Станува збор за податочно множество со 14 променливи(карактеристики) па векторите на нашите податоци се од 14 димензионален векторски простор.

Наша цел е да ги визуелизираме овие податоци, со цел увидеме дали постојат повеќе кластери на испитаници. Но, за да направиме визуелизација мора да имаме најмногу 3 димензии.

[2]:
```
dataset = pd.read_csv('datasets/heart.csv')
dataset.head()
```

[2]:
```
   age  sex  cp  trestbps  chol  fbs  restecg  thalach  exang  oldpeak ␣
↪slope  \
0   63    1   3       145   233    1        0      150      0      2.3      0
1   37    1   2       130   250    0        1      187      0      3.5      0
2   41    0   1       130   204    0        0      172      0      1.4      2
3   56    1   1       120   236    0        1      178      0      0.8      2
4   57    0   0       120   354    0        1      163      1      0.6      2

   ca  thal  target
0   0     1       1
1   0     2       1
2   0     2       1
3   0     2       1
4   0     2       1
```

Во нашето податочно множество, немаме променливи(карактеристики) за кои недостасуваат вредности за одредени примероци.

[3]:
```
dataset.isna().any().any()
```

[3]:
```
False
```

Сите вредности се нумерички, па нема потреба од нивно енкодирање.

[4]:
```
dataset.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 303 entries, 0 to 302
Data columns (total 14 columns):
 #   Column    Non-Null Count  Dtype
---  ------    --------------  -----
```

```
 0   age       303 non-null    int64
 1   sex       303 non-null    int64
 2   cp        303 non-null    int64
 3   trestbps  303 non-null    int64
 4   chol      303 non-null    int64
 5   fbs       303 non-null    int64
 6   restecg   303 non-null    int64
 7   thalach   303 non-null    int64
 8   exang     303 non-null    int64
 9   oldpeak   303 non-null    float64
 10  slope     303 non-null    int64
 11  ca        303 non-null    int64
 12  thal      303 non-null    int64
 13  target    303 non-null    int64
dtypes: float64(1), int64(13)
memory usage: 33.3 KB
```

Податоците претставени како матрица:

[5]:
```
dataset.values
```

[5]:
```
array([[63.,  1.,  3., …,  0.,  1.,  1.],
       [37.,  1.,  2., …,  0.,  2.,  1.],
       [41.,  0.,  1., …,  0.,  2.,  1.],
       …,
       [68.,  1.,  0., …,  2.,  3.,  0.],
       [57.,  1.,  0., …,  1.,  3.,  0.],
       [57.,  0.,  1., …,  1.,  2.,  0.]])
```

4.2 Чекори од Principal Component Analysis

4.2.1 Чекор 1: Стандардизација

[6]:
```
scaler = StandardScaler()

standardized_dataset = pd.DataFrame(scaler.fit_transform(dataset.values))
standardized_dataset.columns = dataset.columns
```

[7]:
```
standardized_dataset.head()
```

[7]:

	age	sex	cp	trestbps	chol	fbs	restecg
0	0.952197	0.681005	1.973123	0.763956	-0.256334	2.394438	-1.005832
1	-1.915313	0.681005	1.002577	-0.092738	0.072199	-0.417635	0.898962
2	-1.474158	-1.468418	0.032031	-0.092738	-0.816773	-0.417635	-1.005832
3	0.180175	0.681005	0.032031	-0.663867	-0.198357	-0.417635	0.898962
4	0.290464	-1.468418	-0.938515	-0.663867	2.082050	-0.417635	0.898962

	thalach	exang	oldpeak	slope	ca	thal	target
0	0.015443	-0.696631	1.087338	-2.274579	-0.714429	-2.148873	0.914529
1	1.633471	-0.696631	2.122573	-2.274579	-0.714429	-0.512922	0.914529
2	0.977514	-0.696631	0.310912	0.976352	-0.714429	-0.512922	0.914529

```
3  1.239897 -0.696631 -0.206705  0.976352 -0.714429 -0.512922  0.914529
4  0.583939  1.435481 -0.379244  0.976352 -0.714429 -0.512922  0.914529
```

4.2.2 Чекор 2: Пресметка на матрица на ковариjанси

[8]:
```
standardized_dataset.cov()
```

[8]:
```
                age       sex        cp  trestbps      chol       fbs  \
age        1.003311 -0.098773 -0.068880  0.280276  0.214385  0.121709
sex       -0.098773  1.003311 -0.049516 -0.056957 -0.198568  0.045181
cp        -0.068880 -0.049516  1.003311  0.047765 -0.077159  0.094757
trestbps   0.280276 -0.056957  0.047765  1.003311  0.123582  0.178118
chol       0.214385 -0.198568 -0.077159  0.123582  1.003311  0.013338
fbs        0.121709  0.045181  0.094757  0.178118  0.013338  1.003311
restecg   -0.116596 -0.058389  0.044568 -0.114481 -0.151540 -0.084468
thalach   -0.399842 -0.044166  0.296741 -0.046852 -0.009973 -0.008595
exang      0.097121  0.142133 -0.395586  0.067840  0.067245  0.025750
oldpeak    0.210708  0.096411 -0.149724  0.193856  0.054131  0.005766
slope     -0.169373 -0.030812  0.120113 -0.121877 -0.004051 -0.060093
ca         0.277241  0.118653 -0.181653  0.101725  0.070744  0.138436
thal       0.068227  0.210737 -0.162271  0.062416  0.099130 -0.032125
target    -0.226185 -0.281867  0.435235 -0.145411 -0.085521 -0.028139

             restecg   thalach     exang   oldpeak     slope        ca  \
age       -0.116596 -0.399842  0.097121  0.210708 -0.169373  0.277241
sex       -0.058389 -0.044166  0.142133  0.096411 -0.030812  0.118653
cp         0.044568  0.296741 -0.395586 -0.149724  0.120113 -0.181653
trestbps  -0.114481 -0.046852  0.067840  0.193856 -0.121877  0.101725
chol      -0.151540 -0.009973  0.067245  0.054131 -0.004051  0.070744
fbs       -0.084468 -0.008595  0.025750  0.005766 -0.060093  0.138436
restecg    1.003311  0.044270 -0.070967 -0.058965  0.093353 -0.072281
thalach    0.044270  1.003311 -0.380066 -0.345327  0.388065 -0.213883
exang     -0.070967 -0.380066  1.003311  0.289177 -0.258602  0.116123
oldpeak   -0.058965 -0.345327  0.289177  1.003311 -0.579449  0.223420
slope      0.093353  0.388065 -0.258602 -0.579449  1.003311 -0.080421
ca        -0.072281 -0.213883  0.116123  0.223420 -0.080421  1.003311
thal      -0.012021 -0.096758  0.207438  0.210940 -0.105111  0.152335
target     0.137684  0.423137 -0.438203 -0.432122  0.347022 -0.393021

              thal    target
age       0.068227 -0.226185
sex       0.210737 -0.281867
cp       -0.162271  0.435235
trestbps  0.062416 -0.145411
chol      0.099130 -0.085521
fbs      -0.032125 -0.028139
restecg  -0.012021  0.137684
thalach  -0.096758  0.423137
exang     0.207438 -0.438203
oldpeak   0.210940 -0.432122
```

```
slope    -0.105111  0.347022
ca        0.152335 -0.393021
thal      1.003311 -0.345168
target   -0.345168  1.003311
```

[9]:
```
cov_matrix = np.matrix(standardized_dataset.cov().to_numpy())
```

4.2.3 Чекор 3: Пресметка на сопствени вредности и вектори матрицата на коваријанси

За пресметка на сопствените вредности ќе користиме `np.linalg`, и основна функција за нивно сортирање.

[10]:
```
eigen_values = -np.sort(-np.linalg.eigvals(cov_matrix))
```

[11]:
```
eigen_values, eigen_vectors = np.linalg.eig(cov_matrix)

idx = eigen_values.argsort()[::-1]
eigen_values = eigen_values[idx]
eigen_vectors = eigen_vectors[:,idx]
```

4.2.4 Чекор 4: Избор на компоненти и добивање на feature вектор

Во Чекор 4, и Чекор 5, меѓу-резултантните матрици се веќе транспонирани поради defaults, па за добивање на резултатот ќе ја користиме следната линеарна трансформација

$$FinalDataSet = (FeatureVector \times StandardizedDataSet^T)^T$$

Како што беше дискутирано претходно, за одбирање на компоненти имаме две опции, и тоа:

- дефинирање праг на значајност
- априори дефинирање на димензија до која сакаме да ги намалиме податоците

Во овој пример априори ќе ја дефинираме димензијата, и ќе одбереме таа да биде 3, бидејќи крајната цел е да ги визуелизираме податоците.

[12]:
```
TARGET_DIMENSION = 3
feature_vector = eigen_vectors[:TARGET_DIMENSION]
```

[13]:
```
feature_vector.shape
```

[13]:
```
(3, 14)
```

4.2.5 Чекор 5: Намалување на димензијата

[14]:
```
final_dataset = np.transpose(np.matmul(feature_vector, np.
↪transpose(standardized_dataset)))
```

[15]:
```
final_dataset
```

[15]:
```
             0         1         2
0   -0.286687  0.110460 -1.914001
1   -0.467832  0.684439 -0.893856
2   -1.265855  0.513491  0.352265
3   -0.385241 -0.656087 -0.169573
4   -1.396762  1.160139  0.477217
..        …         …         …
298 -0.416647  0.652530  0.303550
299 -0.376835 -2.061995 -0.526859
300  1.121740  1.366236 -0.713181
301  1.648491  0.125585  0.280630
302  0.047174  0.139822 -0.800181

[303 rows x 3 columns]
```

4.3 Визуелизација на податоците

Сега нашите податоци се со помала димензија, односно векторите кои ги репрезентираат се од 3 димензионален векторски простор, па истите можеме да ги визуелизираме.

[16]:
```
fig = px.scatter_3d(final_dataset, x=0, y=1, z=2,
                    title="                    3     ",
                    labels={str(i):f"PC: {i}" for i in␣
 ↪range(0,TARGET_DIMENSION)},
                    width=800,
                    height=800
                   )
fig.show()
```

Поради неможноста во tex да биде репрезентирана JavaScript визуелизација, истата е достапна во .ipynb прилогот.

Целиот овој jupyter notebook е достапен на следниот линк.