# Feature Selection and Dimension Reduction

# 23

by **Patrick Boily**, with contributions from **Olivier Leduc**, **Andrew Macfie**, **Aditya Maheshwari**, and **Maia Pelletier**

Data mining is the collection of processes by which we can extract **useful insights** from data. Hidden in this definition is the idea of **data reduction**: useful insights (whether in the form of summaries, sentiment analyses, and so on) should be "smaller" and "more organized" than the original raw data.

But the challenges presented by high data dimensionality (the so-called **curse of dimensionality**) must be addressed in order to achieve insightful and interpretable analytical results.

In this chapter, we introduce the basic principles of **dimensionality reduction** and a number of **feature selection** methods (filter, wrapper, regularization); we also discuss related advanced topics (SVD, spectral feature selection, UMAP).

## 23.1 Data Reduction for Insight

For small datasets, the benefits of data mining may not always be evident. Consider, for instance, the following excerpt from a lawn mowing instruction manual (which we consider to be data for the time being):

> Before starting your mower inspect it carefully to ensure that there are no loose parts and that it is in good working order.

It is a **short** and **organized** way to convey a message. It could be further shortened and organized, perhaps, but what one would gain from such a process is not entirely clear.

### 23.1.1 Reduction of an NHL Game

For a meatier example, consider the NHL game that took place between the Ottawa Senators and the Toronto Maple Leafs on February 18, 2017 [286].

As a first approximation, we shall think of a hockey game as a series of sequential and non-overlapping "events" involving two teams of skaters. What does it mean to have extracted useful insights from such a series of events?

At some level, the most complete raw understanding of that night's game belongs to the game's active and passive participants (players, referees, coaches, general managers, official scorer and time-keeper, etc.).[1]

[1] This simple assumption is rather old-fashioned and would be disputed by many in the age of hockey analytics, but we let it stand for now.

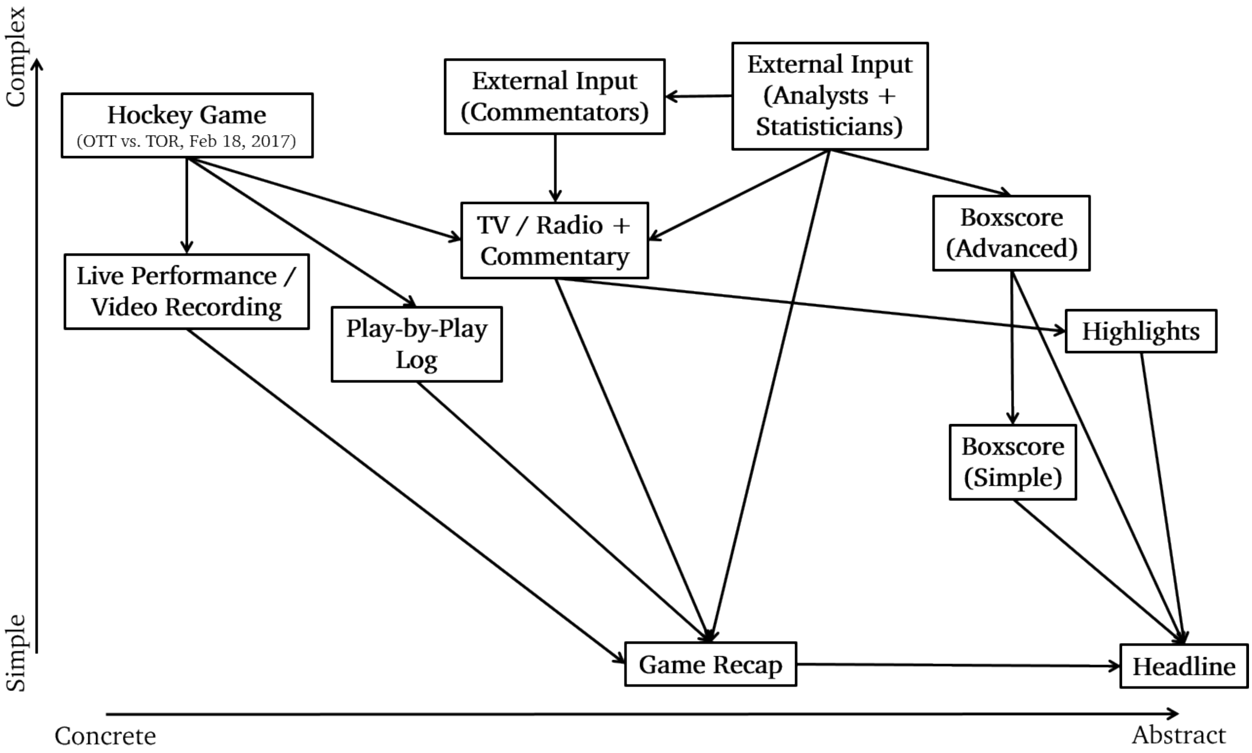


**Figure 23.1:** A schematic diagram of data reduction as it could apply to a professional hockey game.

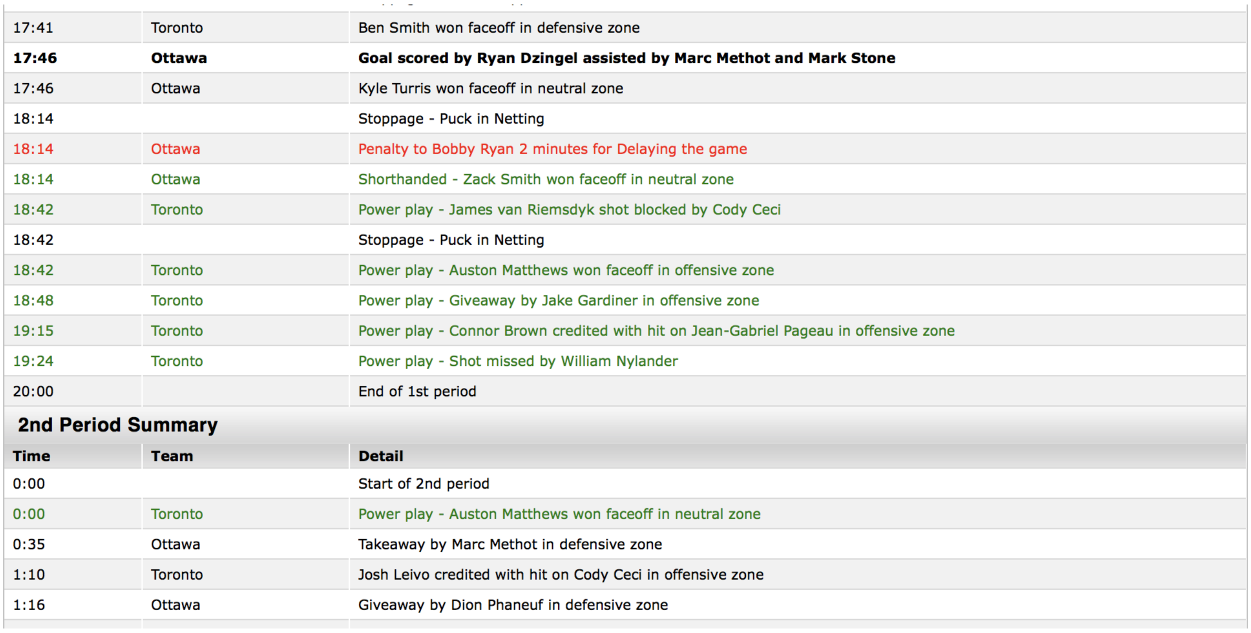
The larger group of individuals who attended the game in person, watched it on TV/Internet, or listened to it on the radio presumably also have a lot of the facts at their disposal, with some contamination, as it were, by commentators (in the two latter cases).

Presumably, the participants and the witnesses also possess insights into the specific game: how could that information best be relayed to members of the public who did not catch the game? There are many ways to do so, depending on the intended level of abstraction and on the target audience (see Figure 23.1).

**Play-by-Play Text File** If a hockey game is a series of events, why not simply list the events, in the order in which they occurred? Of course, not everything that happens in the "**raw**" game requires reporting – it might be impressive to see Auston Matthews skate by Dion Phaneuf on his way to the Senators' net at the 8:45 mark of the 2nd period, say, but reporting this "event" would only serve to highlight the fact that Matthews is a better skater than Phaneuf. It is true, to be sure, but some level of filtering must be applied in order to retain only **relevant** (or "high-level") information, such as:

> blocked shots, face-off wins, giveaways, goals, hits, missed shots, penalties, power play events, saves, shorthanded events, shots on goal, stoppage (goalie stopped, icing, offside, puck in benches), takeaways, etc.

In a typical game, between 300 and 400 events are recorded (see Figure 23.2 for an extract of the play-by-play file for the game under consideration; the full list is found at [286]).



| | | |
|---|---|---|
| 17:41 | Toronto | Ben Smith won faceoff in defensive zone |
| **17:46** | **Ottawa** | **Goal scored by Ryan Dzingel assisted by Marc Methot and Mark Stone** |
| 17:46 | Ottawa | Kyle Turris won faceoff in neutral zone |
| 18:14 | | Stoppage - Puck in Netting |
| 18:14 | Ottawa | Penalty to Bobby Ryan 2 minutes for Delaying the game |
| 18:14 | Ottawa | Shorthanded - Zack Smith won faceoff in neutral zone |
| 18:42 | Toronto | Power play - James van Riemsdyk shot blocked by Cody Ceci |
| 18:42 | | Stoppage - Puck in Netting |
| 18:42 | Toronto | Power play - Auston Matthews won faceoff in offensive zone |
| 18:48 | Toronto | Power play - Giveaway by Jake Gardiner in offensive zone |
| 19:15 | Toronto | Power play - Connor Brown credited with hit on Jean-Gabriel Pageau in offensive zone |
| 19:24 | Toronto | Power play - Shot missed by William Nylander |
| 20:00 | | End of 1st period |
| **2nd Period Summary** | | |
| **Time** | **Team** | **Detail** |
| 0:00 | | Start of 2nd period |
| 0:00 | Toronto | Power play - Auston Matthews won faceoff in neutral zone |
| 0:35 | Ottawa | Takeaway by Marc Methot in defensive zone |
| 1:10 | Toronto | Josh Leivo credited with hit on Cody Ceci in offensive zone |
| 1:16 | Ottawa | Giveaway by Dion Phaneuf in defensive zone |

**Figure 23.2:** Play-by-play extract, Ottawa Senators  Toronto Maple Leafs, February 18, 2017 [286].

Some knowledge about the sport is required to make sense of some of the entries (colouring, use of bold text, etc.), but with patience we can rather easily[2] re-constitute the flow of the game.

This approach, as we can see, is **fully descriptive**.

**Boxscore** The play-by-play does convey the game's events, but the relevance of its entries is sometimes questionable. In the general context of the game, how useful is to know that Nikita Zaitsev blocked a shot by Erik Karlsson at the 2:38 mark of the 1st period? Had this blocked shot saved a guaranteed Senators goal or directly lead to a Maple Leafs goal, one could have argued for its inclusion in the list of crucial events to report, but only the most fastidious observer[3] would bemoan its removal from the game's report.

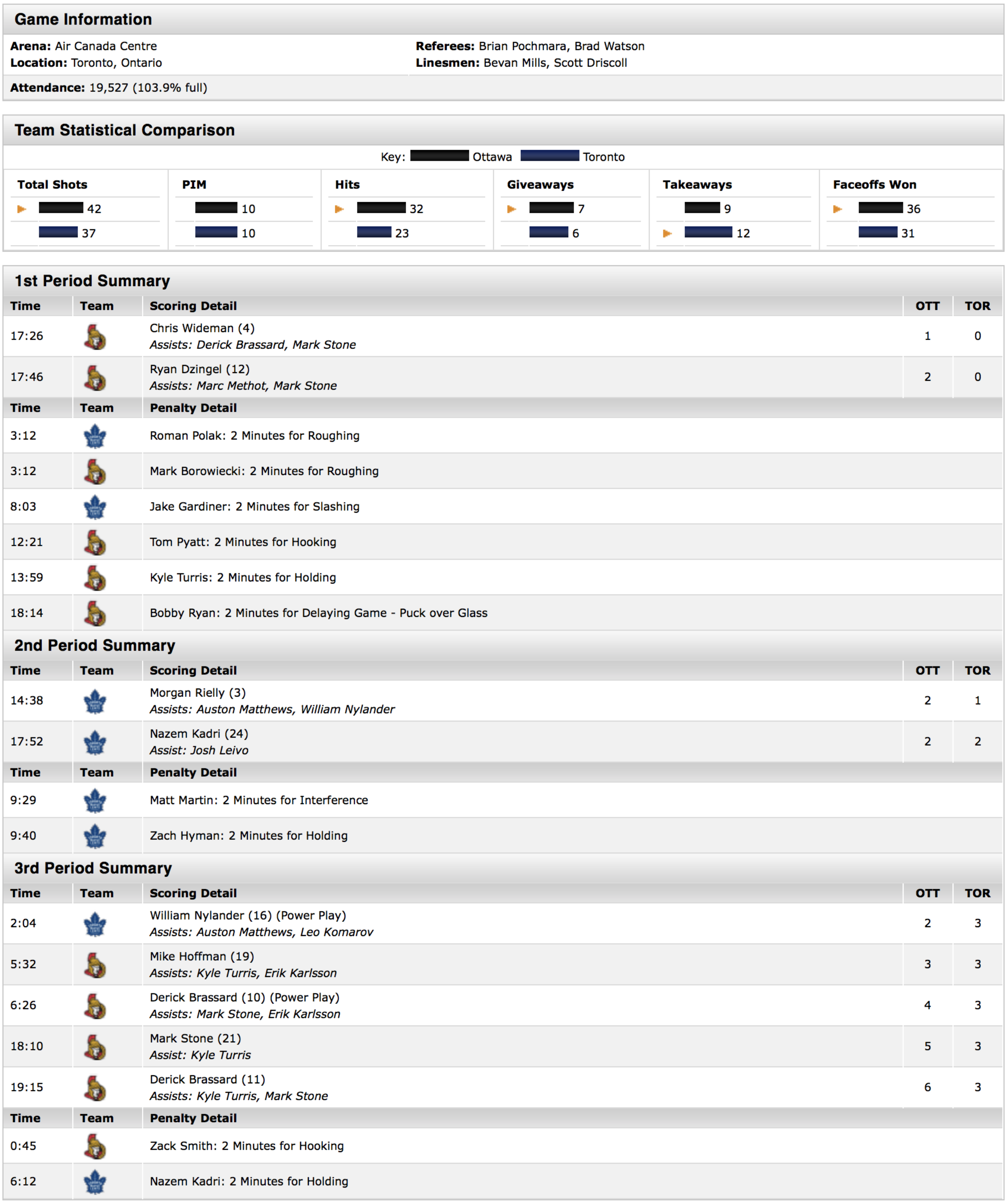
The game's boxscore provides relevant information, at the cost of **completeness**: it distills the play-by-play file into a series of meaningful **statistics** and **summaries**, providing insights into the game that even a fan in attendance might have missed while the game was going on (see Figures 23.3-23.5).

Once again, a certain amount of **knowledge** about the sport is required to make sense of the statistics – to place them in the right context: is it meaningful that the Senators won 36 faceoffs to the Maple Leafs' 31? That Mark Stone was a +4 on the night? That both teams went "1-for-4' ' on the powerplay?

We cannot re-constitute the full flow of the game from the boxscore alone, but the approach is not solely descriptive – questions can be asked, and answers provided... the **analytical** game is afoot!

[2] That is to say, mechanically.

[3] Or a statistical analyst



## Game Information

**Arena:** Air Canada Centre
**Location:** Toronto, Ontario
**Referees:** Brian Pochmara, Brad Watson
**Linesmen:** Bevan Mills, Scott Driscoll
**Attendance:** 19,527 (103.9% full)

## Team Statistical Comparison

Key: ▬ Ottawa ▬ Toronto

| Total Shots | PIM | Hits | Giveaways | Takeaways | Faceoffs Won |
|---|---|---|---|---|---|
| ▶ ▬ 42 | ▬ 10 | ▶ ▬ 32 | ▶ ▬ 7 | ▬ 9 | ▶ ▬ 36 |
| ▬ 37 | ▬ 10 | ▬ 23 | ▬ 6 | ▶ ▬ 12 | ▬ 31 |

## 1st Period Summary

| Time | Team | Scoring Detail | OTT | TOR |
|---|---|---|---|---|
| 17:26 | 🛡 | Chris Wideman (4)<br>*Assists: Derick Brassard, Mark Stone* | 1 | 0 |
| 17:46 | 🛡 | Ryan Dzingel (12)<br>*Assists: Marc Methot, Mark Stone* | 2 | 0 |

| Time | Team | Penalty Detail |
|---|---|---|
| 3:12 | 🍁 | Roman Polak: 2 Minutes for Roughing |
| 3:12 | 🛡 | Mark Borowiecki: 2 Minutes for Roughing |
| 8:03 | 🍁 | Jake Gardiner: 2 Minutes for Slashing |
| 12:21 | 🛡 | Tom Pyatt: 2 Minutes for Hooking |
| 13:59 | 🛡 | Kyle Turris: 2 Minutes for Holding |
| 18:14 | 🛡 | Bobby Ryan: 2 Minutes for Delaying Game - Puck over Glass |

## 2nd Period Summary

| Time | Team | Scoring Detail | OTT | TOR |
|---|---|---|---|---|
| 14:38 | 🍁 | Morgan Rielly (3)<br>*Assists: Auston Matthews, William Nylander* | 2 | 1 |
| 17:52 | 🍁 | Nazem Kadri (24)<br>*Assist: Josh Leivo* | 2 | 2 |

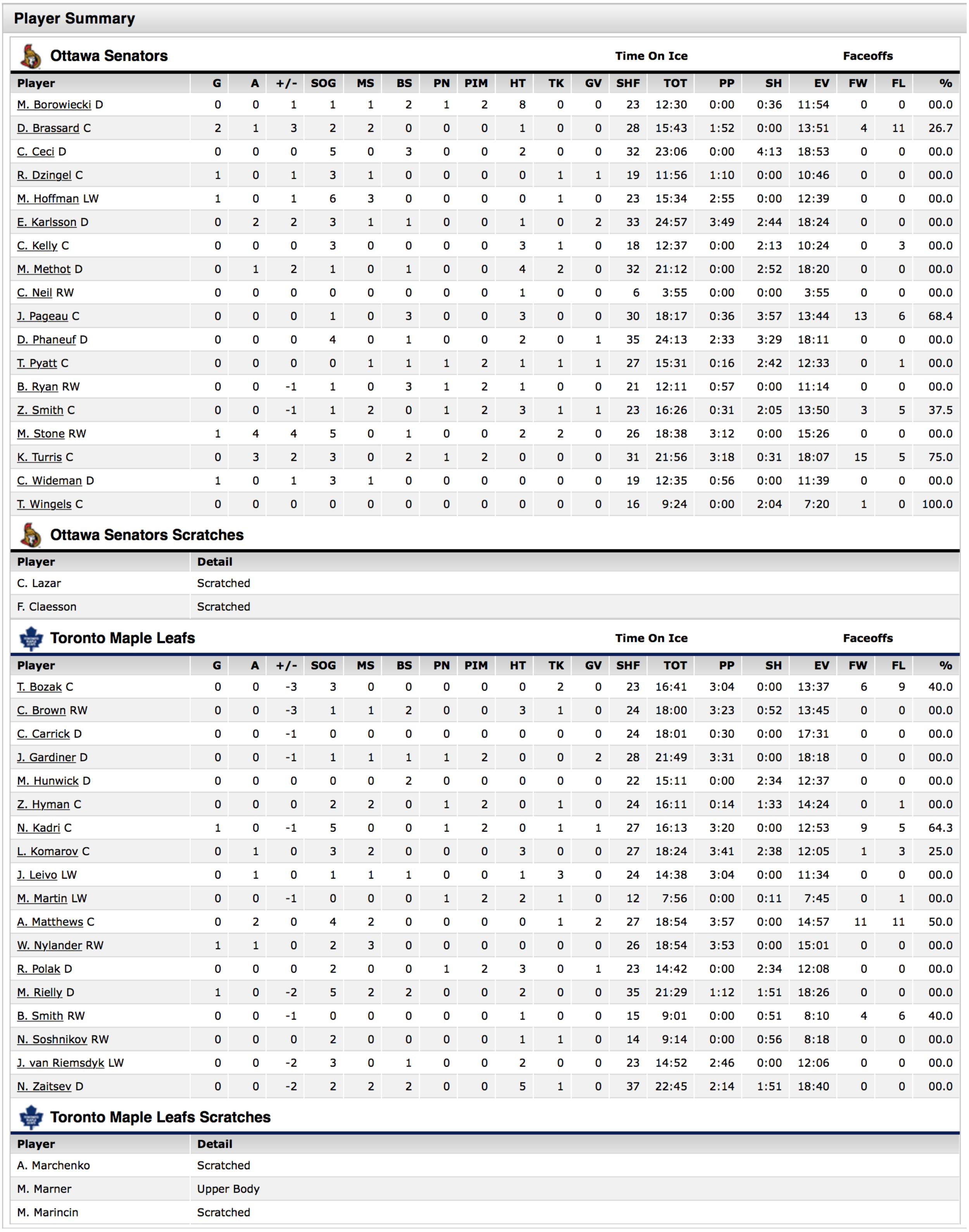
| Time | Team | Penalty Detail |
|---|---|---|
| 9:29 | 🍁 | Matt Martin: 2 Minutes for Interference |
| 9:40 | 🍁 | Zach Hyman: 2 Minutes for Holding |

## 3rd Period Summary

| Time | Team | Scoring Detail | OTT | TOR |
|---|---|---|---|---|
| 2:04 | 🍁 | William Nylander (16) (Power Play)<br>*Assists: Auston Matthews, Leo Komarov* | 2 | 3 |
| 5:32 | 🛡 | Mike Hoffman (19)<br>*Assists: Kyle Turris, Erik Karlsson* | 3 | 3 |
| 6:26 | 🛡 | Derick Brassard (10) (Power Play)<br>*Assists: Mark Stone, Erik Karlsson* | 4 | 3 |
| 18:10 | 🛡 | Mark Stone (21)<br>*Assist: Kyle Turris* | 5 | 3 |
| 19:15 | 🛡 | Derick Brassard (11)<br>*Assists: Kyle Turris, Mark Stone* | 6 | 3 |

| Time | Team | Penalty Detail |
|---|---|---|
| 0:45 | 🛡 | Zack Smith: 2 Minutes for Hooking |
| 6:12 | 🍁 | Nazem Kadri: 2 Minutes for Holding |

**Figure 23.3:** Advanced Boxscore (I), Ottawa Senators  Toronto Maple Leafs, February 18, 2017 [286].



## Player Summary

### Ottawa Senators

| | | | | | | | | | | | | | Time On Ice | | | | Faceoffs | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **Player** | **G** | **A** | **+/-** | **SOG** | **MS** | **BS** | **PN** | **PIM** | **HT** | **TK** | **GV** | **SHF** | **TOT** | **PP** | **SH** | **EV** | **FW** | **FL** | **%** |
| M. Borowiecki D | 0 | 0 | 1 | 1 | 1 | 2 | 1 | 2 | 8 | 0 | 0 | 23 | 12:30 | 0:00 | 0:36 | 11:54 | 0 | 0 | 00.0 |
| D. Brassard C | 2 | 1 | 3 | 2 | 2 | 0 | 0 | 0 | 1 | 0 | 0 | 28 | 15:43 | 1:52 | 0:00 | 13:51 | 4 | 11 | 26.7 |
| C. Ceci D | 0 | 0 | 0 | 5 | 0 | 3 | 0 | 0 | 2 | 0 | 0 | 32 | 23:06 | 0:00 | 4:13 | 18:53 | 0 | 0 | 00.0 |
| R. Dzingel C | 1 | 0 | 1 | 3 | 1 | 0 | 0 | 0 | 0 | 1 | 1 | 19 | 11:56 | 1:10 | 0:00 | 10:46 | 0 | 0 | 00.0 |
| M. Hoffman LW | 1 | 0 | 1 | 6 | 3 | 0 | 0 | 0 | 0 | 1 | 0 | 23 | 15:34 | 2:55 | 0:00 | 12:39 | 0 | 0 | 00.0 |
| E. Karlsson D | 0 | 2 | 2 | 3 | 1 | 1 | 0 | 0 | 1 | 0 | 2 | 33 | 24:57 | 3:49 | 2:44 | 18:24 | 0 | 0 | 00.0 |
| C. Kelly C | 0 | 0 | 0 | 3 | 0 | 0 | 0 | 0 | 3 | 1 | 0 | 18 | 12:37 | 0:00 | 2:13 | 10:24 | 0 | 3 | 00.0 |
| M. Methot D | 0 | 1 | 2 | 1 | 0 | 1 | 0 | 0 | 4 | 2 | 0 | 32 | 21:12 | 0:00 | 2:52 | 18:20 | 0 | 0 | 00.0 |
| C. Neil RW | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 6 | 3:55 | 0:00 | 0:00 | 3:55 | 0 | 0 | 00.0 |
| J. Pageau C | 0 | 0 | 0 | 1 | 0 | 3 | 0 | 0 | 3 | 0 | 0 | 30 | 18:17 | 0:36 | 3:57 | 13:44 | 13 | 6 | 68.4 |
| D. Phaneuf D | 0 | 0 | 0 | 4 | 0 | 1 | 0 | 0 | 2 | 0 | 1 | 35 | 24:13 | 2:33 | 3:29 | 18:11 | 0 | 0 | 00.0 |
| T. Pyatt C | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 2 | 1 | 1 | 1 | 27 | 15:31 | 0:16 | 2:42 | 12:33 | 0 | 1 | 00.0 |
| B. Ryan RW | 0 | 0 | -1 | 1 | 0 | 3 | 1 | 2 | 1 | 0 | 0 | 21 | 12:11 | 0:57 | 0:00 | 11:14 | 0 | 0 | 00.0 |
| Z. Smith C | 0 | 0 | -1 | 1 | 2 | 0 | 1 | 2 | 3 | 1 | 1 | 23 | 16:26 | 0:31 | 2:05 | 13:50 | 3 | 5 | 37.5 |
| M. Stone RW | 1 | 4 | 4 | 5 | 0 | 1 | 0 | 0 | 2 | 2 | 0 | 26 | 18:38 | 3:12 | 0:00 | 15:26 | 0 | 0 | 00.0 |
| K. Turris C | 0 | 3 | 2 | 3 | 0 | 2 | 1 | 2 | 0 | 0 | 0 | 31 | 21:56 | 3:18 | 0:31 | 18:07 | 15 | 5 | 75.0 |
| C. Wideman D | 1 | 0 | 1 | 3 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 19 | 12:35 | 0:56 | 0:00 | 11:39 | 0 | 0 | 00.0 |
| T. Wingels C | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 16 | 9:24 | 0:00 | 2:04 | 7:20 | 1 | 0 | 100.0 |

### Ottawa Senators Scratches

| **Player** | **Detail** |
|---|---|
| C. Lazar | Scratched |
| F. Claesson | Scratched |

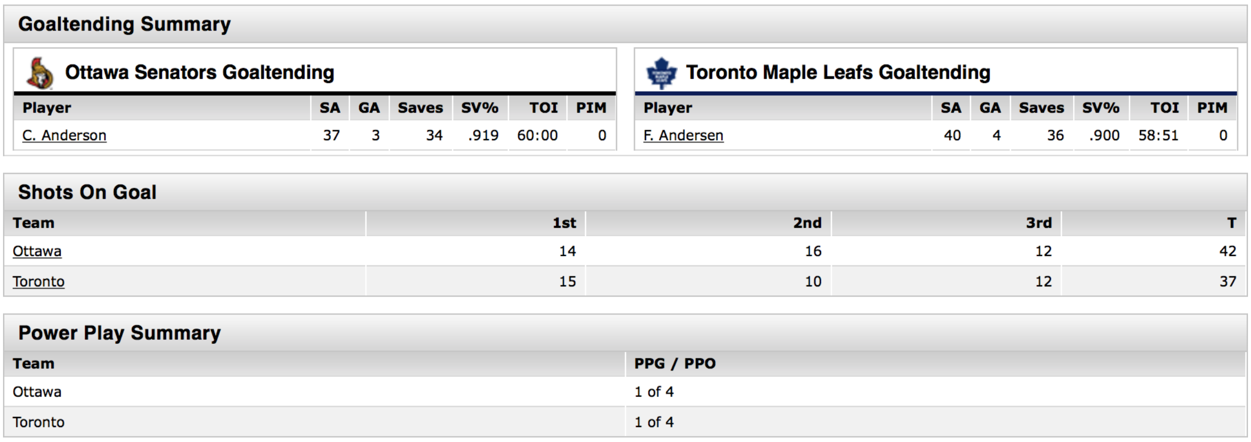
### Toronto Maple Leafs

| | | | | | | | | | | | | | Time On Ice | | | | Faceoffs | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **Player** | **G** | **A** | **+/-** | **SOG** | **MS** | **BS** | **PN** | **PIM** | **HT** | **TK** | **GV** | **SHF** | **TOT** | **PP** | **SH** | **EV** | **FW** | **FL** | **%** |
| T. Bozak C | 0 | 0 | -3 | 3 | 0 | 0 | 0 | 0 | 0 | 2 | 0 | 23 | 16:41 | 3:04 | 0:00 | 13:37 | 6 | 9 | 40.0 |
| C. Brown RW | 0 | 0 | -3 | 1 | 1 | 2 | 0 | 0 | 3 | 1 | 0 | 24 | 18:00 | 3:23 | 0:52 | 13:45 | 0 | 0 | 00.0 |
| C. Carrick D | 0 | 0 | -1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 24 | 18:01 | 0:30 | 0:00 | 17:31 | 0 | 0 | 00.0 |
| J. Gardiner D | 0 | 0 | -1 | 1 | 1 | 1 | 1 | 2 | 0 | 0 | 2 | 28 | 21:49 | 3:31 | 0:00 | 18:18 | 0 | 0 | 00.0 |
| M. Hunwick D | 0 | 0 | 0 | 0 | 0 | 2 | 0 | 0 | 0 | 0 | 0 | 22 | 15:11 | 0:00 | 2:34 | 12:37 | 0 | 0 | 00.0 |
| Z. Hyman C | 0 | 0 | 0 | 2 | 2 | 0 | 1 | 2 | 0 | 1 | 0 | 24 | 16:11 | 0:14 | 1:33 | 14:24 | 0 | 1 | 00.0 |
| N. Kadri C | 1 | 0 | -1 | 5 | 0 | 0 | 1 | 2 | 0 | 1 | 1 | 27 | 16:13 | 3:20 | 0:00 | 12:53 | 9 | 5 | 64.3 |
| L. Komarov C | 0 | 1 | 0 | 3 | 2 | 0 | 0 | 0 | 3 | 0 | 0 | 27 | 18:24 | 3:41 | 2:38 | 12:05 | 1 | 3 | 25.0 |
| J. Leivo LW | 0 | 1 | 0 | 1 | 1 | 1 | 0 | 0 | 1 | 3 | 0 | 24 | 14:38 | 3:04 | 0:00 | 11:34 | 0 | 0 | 00.0 |
| M. Martin LW | 0 | 0 | -1 | 0 | 0 | 0 | 1 | 2 | 2 | 1 | 0 | 12 | 7:56 | 0:00 | 0:11 | 7:45 | 0 | 1 | 00.0 |
| A. Matthews C | 0 | 2 | 0 | 4 | 2 | 0 | 0 | 0 | 0 | 1 | 2 | 27 | 18:54 | 3:57 | 0:00 | 14:57 | 11 | 11 | 50.0 |
| W. Nylander RW | 1 | 1 | 0 | 2 | 3 | 0 | 0 | 0 | 0 | 0 | 0 | 26 | 18:54 | 3:53 | 0:00 | 15:01 | 0 | 0 | 00.0 |
| R. Polak D | 0 | 0 | 0 | 2 | 0 | 0 | 1 | 2 | 3 | 0 | 1 | 23 | 14:42 | 0:00 | 2:34 | 12:08 | 0 | 0 | 00.0 |
| M. Rielly D | 1 | 0 | -2 | 5 | 2 | 2 | 0 | 0 | 2 | 0 | 0 | 35 | 21:29 | 1:12 | 1:51 | 18:26 | 0 | 0 | 00.0 |
| B. Smith RW | 0 | 0 | -1 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 15 | 9:01 | 0:00 | 0:51 | 8:10 | 4 | 6 | 40.0 |
| N. Soshnikov RW | 0 | 0 | 0 | 2 | 0 | 0 | 0 | 0 | 1 | 1 | 0 | 14 | 9:14 | 0:00 | 0:56 | 8:18 | 0 | 0 | 00.0 |
| J. van Riemsdyk LW | 0 | 0 | -2 | 3 | 0 | 1 | 0 | 0 | 2 | 0 | 0 | 23 | 14:52 | 2:46 | 0:00 | 12:06 | 0 | 0 | 00.0 |
| N. Zaitsev D | 0 | 0 | -2 | 2 | 2 | 2 | 0 | 0 | 5 | 1 | 0 | 37 | 22:45 | 2:14 | 1:51 | 18:40 | 0 | 0 | 00.0 |

### Toronto Maple Leafs Scratches

| **Player** | **Detail** |
|---|---|
| A. Marchenko | Scratched |
| M. Marner | Upper Body |
| M. Marincin | Scratched |

**Figure 23.4:** Advanced Boxscore (II), Ottawa Senators Toronto Maple Leafs, February 18, 2017 [286].



Goaltending Summary

Ottawa Senators Goaltending

| Player | SA | GA | Saves | SV% | TOI | PIM |
|---|---|---|---|---|---|---|
| C. Anderson | 37 | 3 | 34 | .919 | 60:00 | 0 |

Toronto Maple Leafs Goaltending

| Player | SA | GA | Saves | SV% | TOI | PIM |
|---|---|---|---|---|---|---|
| F. Andersen | 40 | 4 | 36 | .900 | 58:51 | 0 |

Shots On Goal

| Team | 1st | 2nd | 3rd | T |
|---|---|---|---|---|
| Ottawa | 14 | 16 | 12 | 42 |
| Toronto | 15 | 10 | 12 | 37 |

Power Play Summary

| Team | PPG / PPO |
|---|---|
| Ottawa | 1 of 4 |
| Toronto | 1 of 4 |

**Figure 23.5:** Advanced Boxscore (III), Ottawa Senators  Toronto Maple Leafs, February 18, 2017 [286].

**Recap/Highlights** One of the boxscore's shortcomings is that it does not provide much in the way of **narrative**, which has become a staple of sports reporting – what really happened during that game? How does it impact the current season for either team?

> **Associated Press, 19 February 2017 TORONTO**
> *The Ottawa Senators have the Atlantic Division lead in their sights.*
>
> Mark Stone had a goal and four assists, Derick Brassard scored twice in the third period and the Senators recovered after blowing a two-goal lead to beat the Toronto Maple Leafs 6-3 on Saturday night.
>
> The Senators pulled within two points of Montreal for first place in the Atlantic Division with three games in hand. "We like where we're at. We're in a good spot," Stone said. "But there's a little bit more that we want. Obviously, there's teams coming and we want to try and create separation, so the only way to do that is keep winning hockey games."
>
> Ottawa led 2-0 after one period but trailed 3-2 in the third before getting a tying goal from Mike Hoffman and a power-play goal from Brassard. Stone and Brassard added empty-netters, and Chris Wideman and Ryan Dzingel also scored for the Senators. Ottawa has won four of five overall and three of four against the Leafs this season. Craig Anderson stopped 34 shots.
>
> Morgan Rielly, Nazem Kadri and William Nylander scored and Auston Matthews had two assists for the Maple Leafs. Frederik Andersen allowed four goals on 40 shots. Toronto has lost eight of 11 and entered the night with a tenuous grip on the final wild-card spot in the Eastern Conference.
>
> "The reality is we're all big boys, we can read the standings. You've got to win hockey games," Babcock said. After Nylander made it 3-2 with a power-play goal 2:04 into the third, Hoffman tied it by rifling a shot from the right faceoff circle off the post and in. On a power play 54 seconds later, Andersen stopped Erik Karlsson's point shot, but Brassard jumped on the rebound and put it in for a 4-3 lead.
>
> Wideman started the scoring in the first, firing a point shot through traffic moments after Stone beat Nikita Zaitsev for a puck behind

the Leafs goal. Dzingel added to the lead when he deflected Marc Methot's point shot 20 seconds later.

Andersen stopped three shots during a lengthy 5-on-3 during the second period, and the Leafs got on the board about three minutes later. Rielly scored with 5:22 left in the second by chasing down a wide shot from Matthews, carrying it to the point and shooting through a crowd in front.

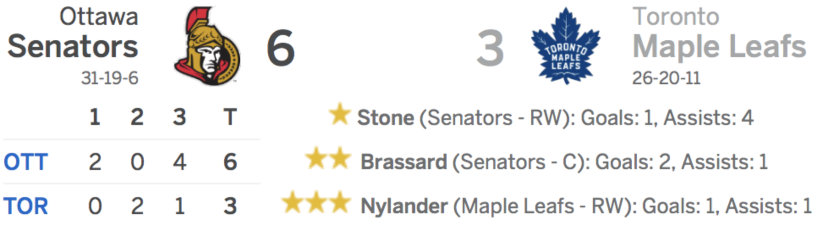
About three minutes later, Zaitsev fired a shot from the right point that sneaked through Anderson's pads and slid behind the net. Kadri chased it down and banked it off Dzingel's helmet and in for his 24th goal of the season. Dzingel had fallen in the crease trying to prevent Kadri from stuffing the rebound in.

"Our game plan didn't change for the third period, and that's just the maturity we're gaining over time," Senators coach Guy Boucher said. "Our leaders have been doing a great job, but collectively, the team has grown dramatically in terms of having poise, executing under pressure."

**Game notes**: Mitch Marner sat out for Toronto with an upper-body injury. Marner leads Toronto with 48 points and is also expected to sit Sunday night against Carolina.

UP NEXT Senators: Host Winnipeg on Sunday night. Maple Leafs: Travel to Carolina for a game Sunday night.

**Simple Boxscore** A gambler or a member of a hockey pool might be interested in the fact that Auston Matthews spent nearly 4 minutes on the powerplay (see Figure **??**), but a casual observer is likely to find the full boxscore a monstrous overkill. How much **crucial information** is lost/provided by the simple boxscore of Figure 23.6, instead?



Ottawa
Senators
31-19-6

6

3

Toronto
Maple Leafs
26-20-11

|  | 1 | 2 | 3 | T |
|---|---|---|---|---|
| OTT | 2 | 0 | 4 | 6 |
| TOR | 0 | 2 | 1 | 3 |

★ **Stone** (Senators - RW): Goals: 1, Assists: 4

★★ **Brassard** (Senators - C): Goals: 2, Assists: 1

★★★ **Nylander** (Maple Leafs - RW): Goals: 1, Assists: 1

**Figure 23.6:** Simple boxscore, Ottawa Senators  Toronto Maple Leafs, February 18, 2017 [286].

**Headline** If we take the view that humans **impose** a narrative on sporting events (rather than unearth it), we could argue that the only **"true" informational content** is found in the following headline:

> **Sens rally after blowing lead, beat Leafs, gain on Habs.** [286]

**Visualization** It is easy to get lost in row after row of statistics and events description, or in large bodies of text.[4] Visualizations can help complement our understanding of any data analytic situation.

While they can be appealing on their own, a certain amount of external context is required to make sense of most of them (see Figure 23.7).

[4] Doubly so for a machine in the latter case.

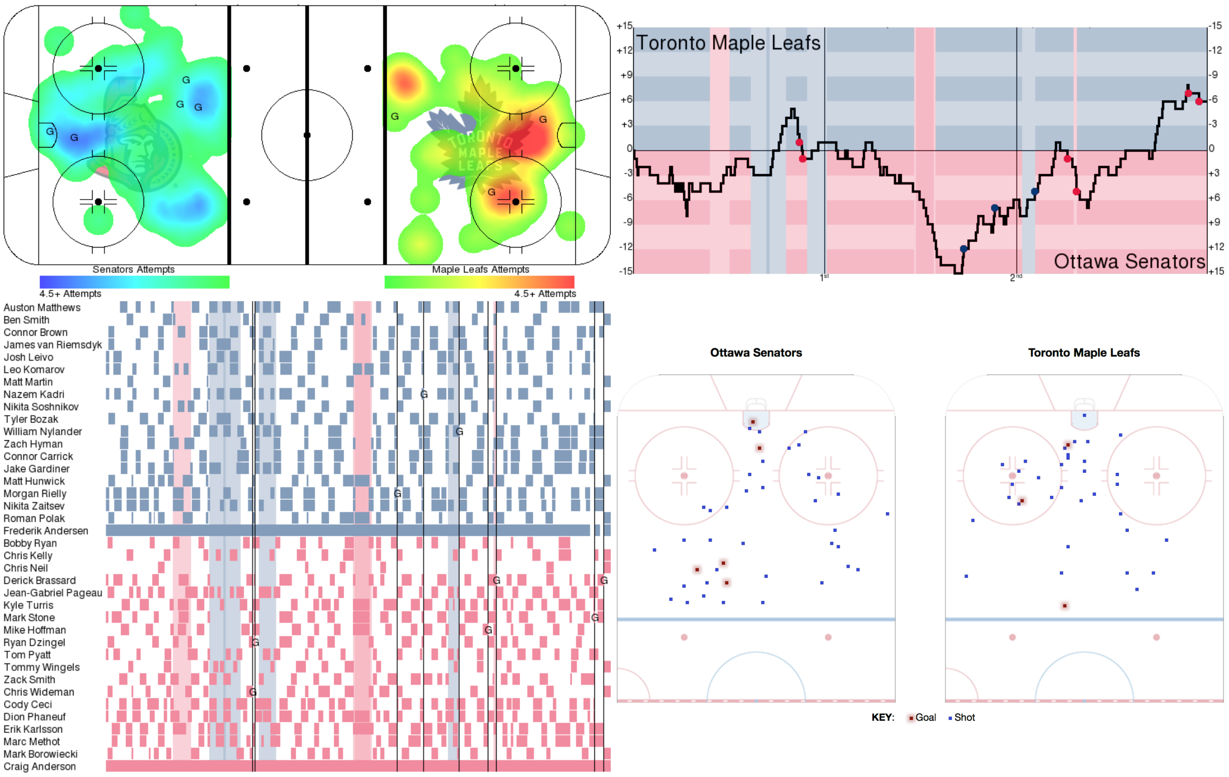


**Figure 23.7:** Visualizations, Ottawa Senators  Toronto Maple Leafs, February 18, 2017: offensive zone unblocked shots heat map (top left), gameflow chart, Corsi +/- , all situations (top right), player shift chart (bottom left), shots and goals (bottom right) [287].

**General Context** A document which is prepared for analysis is often part of a **more general context** (or a collection).

Can the analysis of all the games between the Senators and the Maple Leafs shed some light on their rivalry on the ice? Obviously, the more arcane the representation method, the more in-depth knowledge of the game and its statistics is required, but to those in the know, **summaries** and **visualizations** can provide valuable insight (see Figure 23.8).

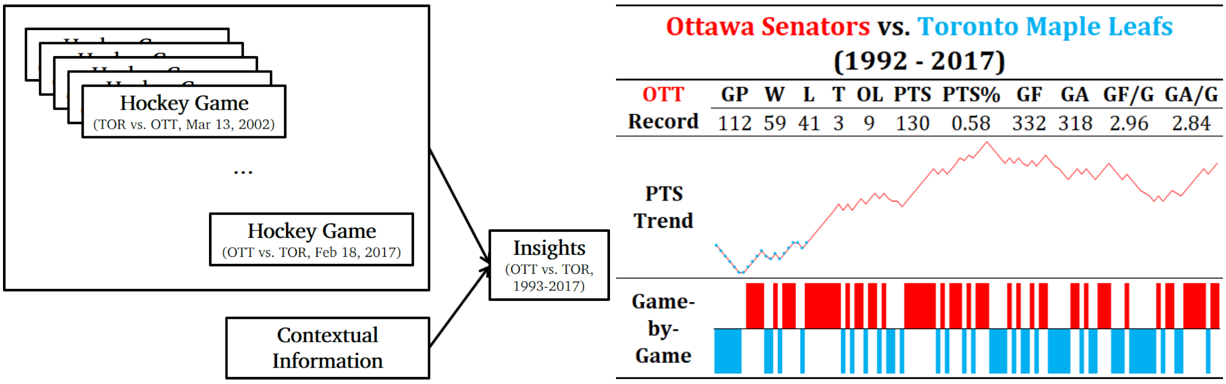
There are thus various ways to understand a single hockey game – and a series of games – depending on the desired (or required) levels of abstraction and complexity.

But as is the case for **all** quantitative methods, data reduction for insight is subject to **analytic choices** – you may have noticed that we conspicuously averted reporting on playoff results, and on post-2017 results. Would the overall "understanding" of the game in question (and the rivalry, in general) change if they were included?[5]

---

The specific details of data reduction as it applies to a professional hockey game are not usually **portable** to general situations, but the **main concepts** are, as we illustrate presently.

[5] Unfortunately for this lifelong Sens fan, it most definitely would...



| OTT Record | GP | W | L | T | OL | PTS | PTS% | GF | GA | GF/G | GA/G |
|---|---|---|---|---|---|---|---|---|---|---|---|
| | 112 | 59 | 41 | 3 | 9 | 130 | 0.58 | 332 | 318 | 2.96 | 2.84 |

**Figure 23.8:** A schematic diagram of data reduction as it applies to a *corpus* of professional hockey games, with visualization and summarizing of regular season games between the Ottawa Senators and Toronto Maple Leafs (1993-2017).

## 23.1.2 Meaning in Macbeth

> It is a tale told by an idiot, full of sound and fury, signifying nothing. [*Macbeth*, V.5, line 30]

In a sense, in order to extract the full **meaning** out of a document, said document needs to be read and understood in its entirety.[6] But even if we have the luxury of doing so, some issues appear:

- do all readers extract the same meaning?
- does meaning stay constant over time?
- is meaning retained by the language of the document?
- do the author's intentions constitute the true (baseline) meaning?
- does re-reading the document change its meaning?

Given the uncertain nature of what a document's meaning actually is, it is counter-productive to talk about insight or meaning (in the **singular**); rather we look for insights and meanings (in the **plural**).

Consider the following passage from *Macbeth* (Act I, Scene 5, Lines 45-52):

> [Enter **MACBETH**]
> **LADY MACBETH**: Great Glamis, worthy Cawdor,
> Greater than both, by the all-hail hereafter,
> Thy letters have transported me beyond
> This ignorant present, and I feel now
> The future in the instant
> **MACBETH**: My dearest love, Duncan comes here tonight.
> **LADY MACBETH**: And when goes hence?
> **MACBETH**: Tomorrow, as he purposes.

What is the "meaning" of this scene? What is the "meaning" of *Macbeth* as a whole? As a starting point, it's crucial to note that the "meaning" of the scene is likely not independent of the play's context up to this scene.[7]

Does the plot description carry the same "meaning" as the play itself? What about TVTropes ↗'s laconic description of *Macbeth* [289]:

> Hen-pecked Scottish nobleman murders his king and spends the rest of the play regretting it.

[6] 6: This section also serves as an introduction to Chapter 29 (*Text Analysis and Text Mining*).

[7] 7: A description of the plot in modern prose is provided in [288].

Or Mister Apple's haiku description (same site)?

> Macbeth and his wife
> Want to become the royals
> So they kill 'em all.

Or this literary description, from an unknown author?

> Macbeth dramatizes the battle between good and evil, exploring the psychological effects of King Duncan's murder on Macbeth and Lady Macbeth. His conflicting feelings of guilt and ambition embody this timeless battle of good vs evil.

Or yet again the (fantastic) 2001 movie *Scotland, PA* , featuring James LeGros, Maura Tierney, and Christopher Walken [290]?

For non-native English speakers (and for a number of native speakers as well, it should be said...), the play (to say nothing of the quoted passage above) might prove difficult to parse and understand.

A modern translation (which is a form of data reduction) is available at *No Fear Shakespeare*, shedding some light on the semantic role of the scene:

> ***MACBETH** enters.*
> **LADY MACBETH**: Great thane of Glamis! Worthy thane of Cawdor! You'll soon be greater than both those titles, once you become king! Your letter has transported me from the present moment, when who knows what will happen, and has made me feel like the future is already here.
> **MACBETH**: My dearest love, Duncan is coming here tonight.
> **LADY MACBETH**: And when is he leaving?
> **MACBETH**: He plans to leave tomorrow.

Consider, also, the French translation by F. Victor Hugo:

> *Entre **MACBETH**.*
> **LADY MACBETH**, *continuant*: Grand Glamis! Digne Cawdor! plus grand que tout cela par le salut futur! Ta lettre m'a transportée au delà de ce présent ignorant, et je ne ne sens plus dans l'instant que l'avenir.
> **MACBETH**: Mon cher amour, Duncan arrive ici ce soir.
> **LADY MACBETH**: Et quand repart-il?
> **MACBETH**: Demain... C'est son intention.

Do these all carry the same *Macbeth* essence? Do they all even carry a *Macbeth* essence? Are they all *Macbeth*? How much, if anything, of *Macbeth* do they preserve? The French translation, for instance, adds a very ominous tone to Macbeth's last retort to his wife.

Those of us who have read the rest of the play know that the tone is in keeping with the events that will eventually transpire, but does the translation add some foreshadowing that is simply not present up to that point in the original? If so, does it matter?

## 23.2 Dimension Reduction

There are advantages to working with **reduced, low-dimensional data**:

- **visualisation methods** of all kinds are available and (more) readily applicable to such data in order to extract and present insights;
- high-dimensional data is subject to the **curse of dimensionality** (CoD), which asserts (among other things) that multi-dimensional spaces are ... well, **vast**, and when the number of features in a model **increases**, the number of observations required to maintain predictive power also **increases**, but at a **substantially larger rate**;
- a consequence of CoD is that in high-dimension sets, all observations are roughly **dissimilar** to one another – observations tend to be **nearer** the dataset's **boundaries** than to one another.

Dimension reduction techniques such as the ubiquitous **principal component analysis**, **independent component analysis**, **factor analysis**,[8] or **multiple correspondence analysis**[9] project multi-dimensional datasets onto **low-dimensional** but **high-information** spaces.[10]

Some information is necessarily lost in the process, but in many instances the drain can be kept under control and the gains made by working with smaller datasets can offset the loss of completeness.

### 23.2.1 Sampling Observations

Datasets can be "**big**" in a variety of ways:

- they can be too large for the **hardware** to handle,[11] or
- the dimensions can go against specific *modeling assumptions*.[12]

For instance, multiple sensors which record 100+ observations per second in a large geographical area over a long time period can lead to excessively big datasets, say.

A natural question, regarding such a dataset, is whether every one of its row needs to be used: if rows are selected randomly (with or without replacement), the resulting sample might be **representative**[13] of the entire dataset, and the smaller set might be easier to handle.

There are some drawbacks to the sampling approach, however:

- if the signal of interest is **rare**, sampling might lose its presence altogether;
- if **aggregation** happens at some point in the reporting process, sampling will necessarily affect the totals,[14] and
- even **simple** operations on large files (finding the # of rows, say) can be taxing on the memory or computation time – some knowledge or **prior information** about the dataset structure can help.

Sampled datasets can also be used to work the kinks out of the data analysis workflows, but the key take-away is that if data is too big to store, access, and manipulate in a reasonable amount of time, the issue is mostly a **Big Data problem** – this is the time to start considering the use of distributed computing (see Chapter 18, *What's the Big Deal with Big Data?*).

[8] For numerical data.

[9] For categorical data.

[10] The so-called **Manifold Hypothesis**, see Section 23.2.4.

[11] That is to say, they cannot be stored or accessed properly due to the number of observations, the number of features, or the overall size of the dataset.

[12] Such as the number of features being much larger than the number of observations, say.

[13] An entire field of statistical endeavour – **statistical survey sampling** – has been developed to quantify the extent to which the sample is representative of the population, see Chapter 11 (*Survey Sampling Methods*).

[14] For instance, if we are interested in predicting the number of passengers per flight leaving YOW (Macdonald-Cartier International Airport) and the total population of passengers is sampled, then the sampled number of passengers per flight is necessarily below the actual number of passengers per flight. Estimation methods exist to overcome these issues (see Chapter 11).

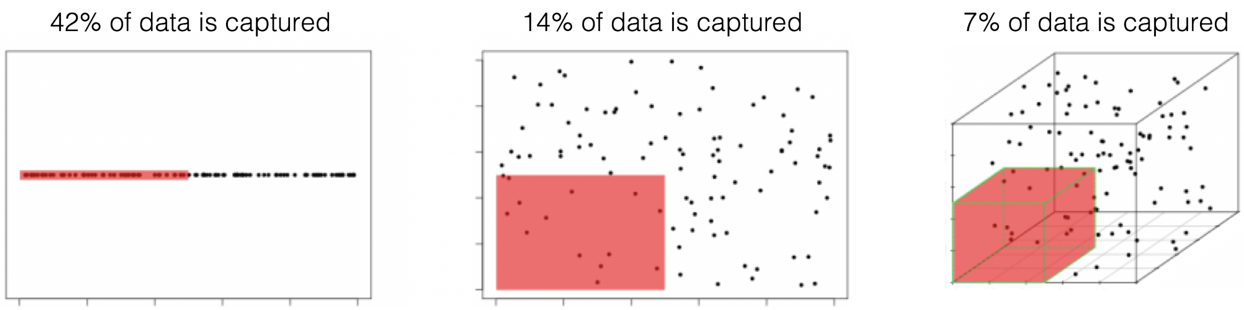


**Figure 23.9:** Illustration of the curse of dimensionality; $N = 100$ observations are uniformly distributed on the unit hypercube $[0,1]^d$, $d = 1,2,3$. The red regions represent the smaller hypercubes $[0,0.5]^d$, $d = 1,2,3$. The percentage of captured data points is seen to decrease with an increase in $d$ [186].

### 23.2.2 The Curse of Dimensionality

A model is said to be **local** if it depends solely on the observations near the input vector ($k$ nearest neigbours classification is local, whereas linear regression is global). With a large training set, increasing $k$ in a $k$NN model, say, will yield enough data points to provide a solid approximation to the theoretical classification boundary.

The **curse of dimensionality** (CoD) is the breakdown of this approach in high-dimensional spaces: when the number of features increases, the number of observations required to maintain predictive power also increases, **but at a substantially higher rate** (see Figure 23.9 for an illustration of the CoD in action).

**Manifestations of CoD** Let $x_i \sim U^1(0,1)$ be i.i.d. for $i = 1,\ldots,n$. For any $z \in [0,1]$ and $\varepsilon \in (0,1]$ such that

$$I_1(z;\varepsilon) = \{y \in \mathbb{R} : |z-y|_\infty < \varepsilon/2\} \subseteq [0,1],$$

the expected number of observations $x_i$ in $I_1(z;\varepsilon)$ is

$$\left|I_1(z;\varepsilon) \cap \{x_i\}_{i=1}^n\right| \approx \varepsilon \cdot N,$$

so an $\varepsilon_\infty$−ball subset of $[0,1]^1$ contains approximately $\varepsilon$ of the observations in $\{x_i\}_{i=1}^n \subseteq \mathbb{R}$, **on average**.

Let $\mathbf{x}_i \sim U^2(0,1)$ be i.i.d. for all $i = 1,\ldots,n$. For any $\mathbf{z} \in [0,1]^2$ and $\varepsilon \in (0,1]$ such that

$$I_2(\mathbf{z};\varepsilon) = \{\mathbf{Y} \in \mathbb{R}^2 : \|\mathbf{z}-\mathbf{Y}\|_\infty < \varepsilon/2\} \subseteq [0,1]^2,$$

the expected number of observations $\mathbf{x}_i$ in $I_2(\mathbf{z};\varepsilon)$ is

$$\left|I_1(\mathbf{z};\varepsilon) \cap \{\mathbf{x}_i\}_{i=1}^n\right| \approx \varepsilon^2 \cdot N,$$

so an $\varepsilon_\infty$−ball subset of $[0,1]^2$ contains approximately $\varepsilon^2$ of the observations in $\{\mathbf{x}_i\}_{i=1}^n \subseteq \mathbb{R}^2$, **on average**.

In general, this reasoning shows that an $\varepsilon_\infty$−ball subset of $[0,1]^p \subseteq \mathbb{R}^p$ contains approximately $\varepsilon^p$ of the observations in $\{\mathbf{x}_i\}_{i=1}^n \subseteq \mathbb{R}^p$, **on average**.

To capture $r$% of uniformly distributed observations in a unit $p-$hypercube, we need a $p-$hypercube with edge $\varepsilon_p(r) = r^{1/p}$, on average. For instance, to capture $r = 1/3$ of the observations in a unit $p-$hypercube in $\mathbb{R}$, $\mathbb{R}^2$, and $\mathbb{R}^{10}$, a hyper-subset with edge $\varepsilon_1(1/3) \approx 0.33$, $\varepsilon_2(1/3) \approx 0.58$, and $\varepsilon_{10}(1/3) \approx 0.90$, respectively.

The inference is simple, but far-reaching: in general, as $p$ increases, the nearest observations to a given point $\mathbf{x}_j \in \mathbb{R}^p$ are in fact quite distant from $\mathbf{x}_j$, in the Euclidean sense, on average – **locality is lost**![15]

This can wreak havoc on models and algorithms that rely on the (Euclidean) nearness of observations ($k$ nearest neighbours, $k-$means clustering, etc.). The CoD manifests itself in various ways.

In datasets with a large number of features:

- most observations are **nearer the edge of the sample than they are to other observations**, and
- realistic training sets are necessarily **sparse**.

Imposing restrictions on models can help mitigate the effects of the CoD, but if they are not warranted the end result may be catastrophic.

### 23.2.3 Principal Component Analysis

**Principal component analysis** (PCA) can be used to find the combinations of variables along which the data points are **most spread out**; it attempts to fit a $p-$**ellipsoid** to a centered representation of the data. The ellipsoid axes are the **principal components** of the data.

Small axes are components along which the variance is "small"; removing these components leads, in an ideal setting, to a "small" loss of information[16] (see Figure 23.10). The procedure is simple:

1. centre and "scale" the data to obtain a matrix $\mathbf{X}$;[17]
2. compute the data's covariance matrix $\mathbf{K} = \mathbf{X}^\top\mathbf{X}$;
3. find $\mathbf{K}$'s eigenvalues $\mathbf{\Lambda}$ and its orthonormal eigenvectors matrix $\mathbf{W}$;
4. each eigenvector $\mathbf{w}$ (also known as **loading**) represents an axis, whose variance is given by the associated eigenvalue $\lambda$.

The loading that explains the most variance along a single axis (the **1st PC**) is the eigenvector of the empirical covariance matrix corresponding to the largest eigenvalue, and that variance is proportional to the eigenvalue; the 2nd largest eigenvalue and its corresponding eigenvector yields the **2nd PC** and variance pair, and so on, yielding **orthonormal** principal components $\text{PC}_1, \ldots, \text{PC}_r$, where $r = \text{rank}(\mathbf{X})$.[18]

PCA can provide an avenue for dimension reduction, by "removing" components with small eigenvalues (as in Figure 23.10). The **proportion of the spread in the data** which can be explained by each principal component can be placed in a **scree plot** (a plot of eigenvalues against ordered component indices), and we retain the ordered PCs:

- for which the eigenvalue is above some threshold (say, 25%);
- for which the cumulative proportion of the spread falls below some threshold (say 95%), or
- prior to a **kink** in the scree plot.

[15] The situation may not be as stark if the observations are not i.i.d., but the principle remains the same – in high-dimensional spaces, it is harder for observations to be near one another than it is so in low-dimensional spaces.

[16] Although there are scenarios where it could be those "small" axes that are more interesting – such as is the case with the "pancake stack" problem.

[17] This is NOT the design matrix as defined in regression analysis.

[18] If some of the eigenvalues are 0, then $r < p$, and *vice-versa*, implying that the data was embedded in a $r-$dimensional manifold to begin with.

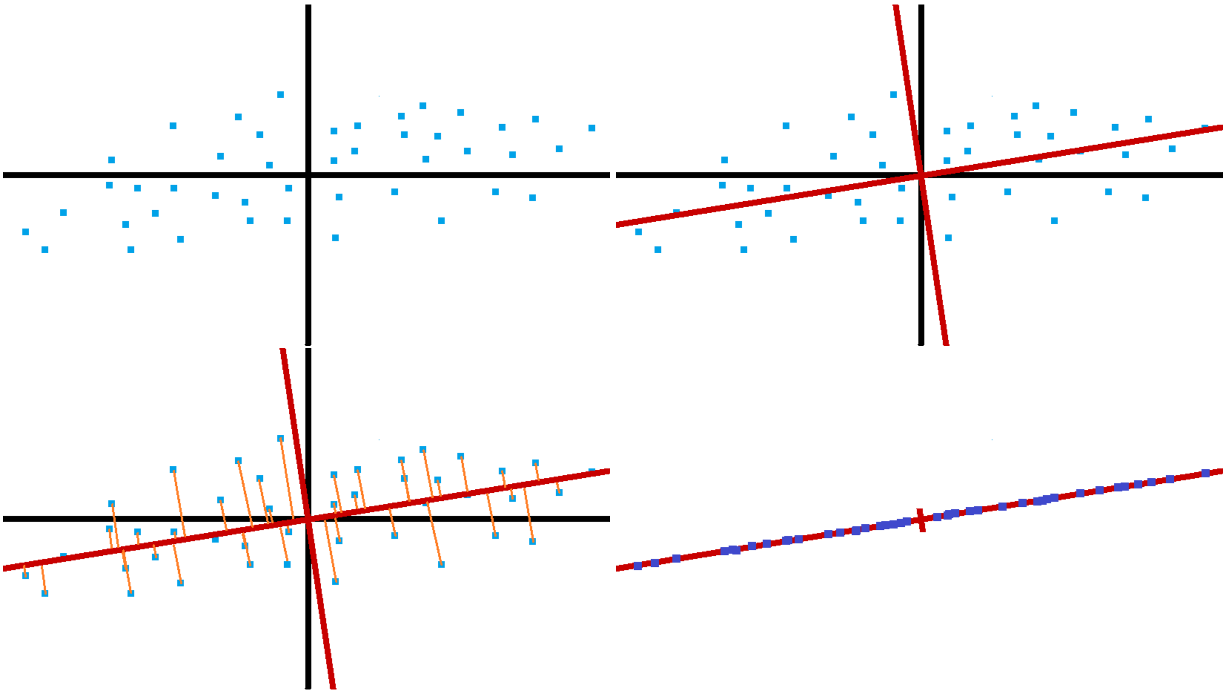


**Figure 23.10:** Illustration of PCA on an artificial 2D dataset (top left). The red axes (top right) represent the axes of the best elliptic fit. Removing the minor axis by projecting the points on the major axis (bottom left) leads to a dimension reduction and a (small) loss of information (bottom right).

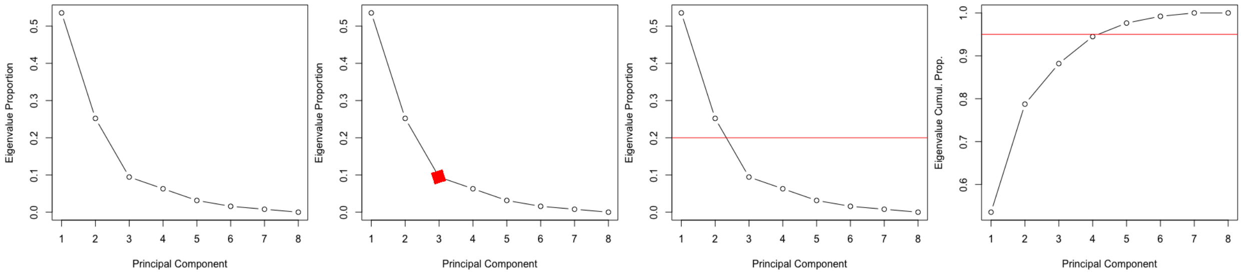


**Figure 23.11:** Selecting the number of principal component – the proportion of the variance explained by each (ordered) component is shown in the first 3 charts; the cumulative proportion is shown in the last chart. The kink method is shown in the top right image, the individual threshold component in the bottom left image, and the cumulative proportion in the bottom right image.

For instance, consider an 8−dimensional dataset for which the ordered PCA eigenvalues are provided below:

| **PC** | **1** | **2** | **3** | **4** | **5** | **6** | **7** | **8** |
|---|---|---|---|---|---|---|---|---|
| **Var** | 17 | 8 | 3 | 2 | 1 | 0.5 | 0.25 | 0 |
| **Prop** | 54 | 25 | 9 | 6 | 3 | 2 | 1 | 0 |
| **Cumul** | 54 | 79 | 88 | 94 | 98 | 99 | 100 | 100 |

If the only PCs that are retained are those that explain up to 95% of the cumulative variation, say, then the original data reduces to a 4-dimensional subset; if only the PCs that individually explain more than 25% of the variation are retained, say, then the data reduces to a 2-dimensional subset; if only the PCs that lead into the first kink in the scree plot are retained, to a 3-dimensional subset (see Figure 23.11).

PCA is commonly-used, but often without regard to its inherent **limitations**, unfortunately:

- it is dependent on scaling, and so is not uniquely determined;
- with little domain expertise, it may be difficult to interpret the PCs;
- it is quite sensitive to outliers;
- the analysis goals are not always aligned with the principal components, and
- the data assumptions are not always met – in particular, does it always make sense that important data structures and data spread be correlated (the so-called **counting pancakes** problem), or that the components be **orthogonal**?

There are other methods to find the **principal manifolds** of a dataset, including UMAP, self-organizing maps, auto-encoders, curvilinear component analysis, manifold sculpting, kernel PCA, etc.

**Formalism** Because **K** is **positive semi-definite** ($\mathbf{K} \geq 0$), the eigenvalues $\lambda_i = s_i^2$ are non-negative and they can be ordered in a decreasing sequence

$$\mathbf{\Lambda} = \text{diag}(\lambda_1, \dots, \lambda_p), \quad \text{where } \lambda_1 \geq \cdots \lambda_p \geq 0$$

and $\mathbf{W} = [\mathbf{w}_1 | \cdots | \mathbf{w}_p]$.

If $k = \text{rank}(\mathbf{X})$, then there are $p - k$ "empty" principal component (corresponding to null eigenvalues) and $k$ "regular" principal components (corresponding to zero eigenvalues). We write $\mathbf{W}^* = [\mathbf{w}_1 | \cdots | \mathbf{w}_k]$ and $\mathbf{\Lambda}^* = \text{diag}(\lambda_1, \dots, \lambda_k)$. If $p - k \neq 0$, then the eigenvalue decomposition of **K** is

$$\mathbf{K} = \begin{bmatrix} \mathbf{W}^* & \mathbf{0} \end{bmatrix} \begin{bmatrix} \mathbf{\Lambda}^* & \mathbf{0} \\ \mathbf{0} & \mathbf{0} \end{bmatrix} \begin{bmatrix} (\mathbf{W}^*)^\top \\ \mathbf{0} \end{bmatrix} = \mathbf{W}\mathbf{\Lambda}\mathbf{W}^\top;$$

if **X** is of full rank, then $\mathbf{W}^* = \mathbf{W}$ and $\mathbf{\Lambda}^* = \mathbf{\Lambda}$.

The eigenvectors of $K$ (the $\mathbf{w}_j$) are the **singular vectors** of **X**: there exist $\mathbf{U}_{n\times n}$ and $\mathbf{\Sigma}_{n\times p}$ such that

$$\mathbf{X} = \mathbf{U}\mathbf{\Sigma}\mathbf{W}^\top,$$

where

$$\mathbf{U} = \begin{pmatrix} \mathbf{U}^* & \mathbf{0} \end{pmatrix} \quad \text{and} \quad \mathbf{\Sigma} = \begin{pmatrix} \text{diag}(s_i) \\ \mathbf{0} \end{pmatrix}.$$

If **X** is of full rank, then **W** is **orthonormal** and so represents a **rotation matrix**. As $\mathbf{W}^{-1} = \mathbf{W}^\top$, we must then have $\mathbf{XW} = \mathbf{U\Sigma}$, the **principal component decomposition of X**:

$$\mathbf{T}_{n\times p} = \mathbf{XW}, \quad \begin{bmatrix} \mathbf{t}_1 & \cdots & \mathbf{t}_p \end{bmatrix} = \begin{bmatrix} \mathbf{x}_1 & \cdots & \mathbf{x}_n \end{bmatrix}^\top \begin{bmatrix} \mathbf{w}_1 & \cdots & \mathbf{w}_p \end{bmatrix}.$$

The link between the principal components and the eigenvectors can be made explicit: the first principal component PC$_1$ is the loading $\mathbf{w}_1$ (with $\|\mathbf{w}_1\|_2 = 1$) which **maximizes the variance of the first column** of **T**:

$$\mathbf{w}_1 = \arg\max_{\|\mathbf{w}\|_2=1} \{\text{Var}(\mathbf{t}_1)\} = \arg\max_{\|\mathbf{w}\|_2=1} \left\{ \frac{1}{n-1} \sum_{i=1}^{n} (t_{1,i} - \overline{t_1})^2 \right\}.$$

Since

$$\overline{t_1} = \frac{1}{n}\sum_{j=1}^{n} \mathrm{E}[\mathbf{x}_j^\top \mathbf{w}_1] = \frac{1}{n}\sum_{j=1}^{n} \mathrm{E}\left[\sum_{i=1}^{p} x_{j,i} w_{i,1}\right]$$
$$= \frac{1}{n}\sum_{j=1}^{n}\sum_{i=1}^{p} w_{i,1} \underbrace{\mathrm{E}[x_{j,i}]}_{=0} = 0,$$

then $\mathrm{Var}(\mathbf{t}_1) = \frac{1}{n-1}(t_{1,1}^2 + \cdots + t_{n,1}^2)$ and the problem is equivalent to

$$\mathbf{w}_1 = \arg\max_{\|\mathbf{w}\|_2=1} \{t_{1,1}^2 + \cdots + t_{n,1}^2\},$$

By construction, $t_{i,1}^2 = (\mathbf{x}_i^\top \mathbf{w}_1)^2$ for all $i$, so

$$t_{1,1}^2 + \cdots + t_{n,1}^2 = (\mathbf{x}_1^\top \mathbf{w}_1)^2 + \cdots + (\mathbf{x}_n^\top \mathbf{w}_1)^2 = \|\mathbf{X}\mathbf{w}_1\|_2 = \mathbf{w}_1\mathbf{X}^\top\mathbf{X}\mathbf{w}_1.$$

Hence,

$$\mathbf{w}_1 = \arg\max_{\|\mathbf{w}\|_2=1} \{\mathbf{w}\mathbf{X}^\top\mathbf{X}\mathbf{w}\} = \arg\max_{\|\mathbf{w}\|_2=1} \{\mathbf{w}^\top\mathbf{K}\mathbf{w}\};$$

this is equivalent to finding the maximizer of $F(\mathbf{w}) = \mathbf{w}^\top\mathbf{K}\mathbf{w}$ subject to the constraint

$$G(\mathbf{w}) = 1 - \mathbf{w}^\top\mathbf{w} = 0.$$

We solve this problem by using the method of Lagarange multipliers; any optimizer $\mathbf{w}^*$ must be either:

1. a critical point of $F$, or
2. a solution of $\nabla F(\mathbf{w}) + \lambda\nabla G(\mathbf{w}) = \mathbf{0}$, $\lambda \neq 0$.

But $\nabla F(\mathbf{w}) = 2\mathbf{K}\mathbf{w}$ and $\nabla G(\mathbf{w}) = -2\mathbf{w}$; either $\mathbf{w}^* \in \ker(\mathbf{K})$ (case 1) or $2\mathbf{K}\mathbf{w}^* - 2\lambda^*\mathbf{w}^* = \mathbf{0}$ (case 2); either

$$\mathbf{K}\mathbf{w}^* = \mathbf{0} \quad \text{or} \quad (\mathbf{K} - \lambda^* I)\mathbf{w}^* = \mathbf{0},\ \lambda^* \neq 0.$$

In either case, $\lambda^* \geq 0$ is an eigenvalue of $K$, with associated eigenvector $\mathbf{w}^*$. There are at most $p$ distinct possibilities $\{(\lambda_j, \mathbf{w}_j)\}_{j=1}^{p}$, and for each of them

$$\mathbf{w}_j^\top\mathbf{K}\mathbf{w}_j = \mathbf{w}_j^\top\lambda_j\mathbf{w}_j = \lambda_j\mathbf{w}_j^\top\mathbf{w}_j = \lambda_j,$$

since $\mathbf{w}_j^\top\mathbf{w}_j = 1$.

Thus,

$$\arg\max_{\|\mathbf{w}\|_2=1} \{\mathrm{Var}(\mathbf{t}_1)\} = \arg\max_{\|\mathbf{w}\|_2=1} \{\lambda_j\} = \mathbf{w}_1 = \mathrm{PC}_1,$$

since $\lambda_1 \geq \lambda \geq 0$ for all eigenvalues $\lambda$ of $\mathbf{K}$.

A similar argument shows that $\mathbf{w}_j$, $j = 2, \ldots, p$, is the direction along which the variance is the $j$th highest, assuming that $\mathbf{w}_j$ is orthonormal to all the preceding $\mathbf{w}_\ell$, $\ell = 1, \ldots, j-1$, and that the variance is proportional to $\lambda_j$.

The process is repeated at most $p$ times, yielding $r$ non-trivial principal components $\mathrm{PC}_1, \ldots, \mathrm{PC}_r$, where $r \leq p$ is the rank($\mathbf{X}$). Thus, we see that the rotation matrix $\mathbf{W}$ that maximizes the variance sequentially in the columns of $\mathbf{T} = \mathbf{X}\mathbf{W}$ is the matrix of eigenvectors of $\mathbf{K} = \mathbf{X}^\top\mathbf{X}$.

We show how to implement principal component analysis in Sections 19.7.3 and 21.4.3 (in the Wine example).

### 23.2.4 The Manifold Hypothesis

**Manifold learning** involves mapping high-dimensional data to a lower dimensional manifold, such as mapping a set of points in $\mathbb{R}^3$ to a torus shape, which can then be **unfolded** (or embedded) into a 2D object.

Techniques for manifold learning are commonly-used because data is often (usually?) **sampled** from unknown and underlying sources which cannot be measured directly.

Learning a suitable "low-dimension" manifold from a higher-dimensional space is approximately as complicated as learning the sources (in a ML sense). This problem can also be re-cast as finding a set of **degrees of freedom** which can reproduce most of the variability in a dataset.

For instance, a set of multiple photographs of a 3D object taken from different positions but at the same distance from the object can be represented by two degrees of freedom: the **horizontal** and **vertical** angles from which the picture was taken.

As another example, consider a set of hand-written drawings of the digit "2" [291]. Each of these drawings can also be represented using a small number of degrees of freedom:

- the **ratio** of the length of the **lowest horizontal line** to the height of the hand-written **digit**;
- the **ratio** of the length of the **arch in the curve** at the top to the smallest horizontal distance from the **end point of the arch** to the **main vertical curve**;
- the **angle of rotation** of the digit as a whole with respect to some **baseline orientation**, etc.

These two scenarios are illustrated in Figure 23.12.

**Dimensionality reduction** and **manifold learning** are often used for one of three purposes:

- to **reduce the overall dimensionality** of the data while trying to **preserve the variance in the data**;
- to **display** high-dimensional datasets, or
- to **reduce the processing time** of supervised learning algorithms by lowering the dimensionality of the data.

PCA, for instance, provides a sequence of best linear approximations to high-dimensional observations (see previous section); the process has fantastic theoretical properties for computation and applications, but data is not always well-approximated by a fully linear process.

In this section, the focus is on non-linear dimensionality reduction methods, most of which are a variant of **kernel PCA**:

- LLE;
- Laplacian eigenmap;
- isomap;
- semidefinite embedding, and
- $t-$SNE.

By way of illustration, the different methods are applied to an "S"-shaped coloured 2D object living in 3D space (see Figure 23.15).

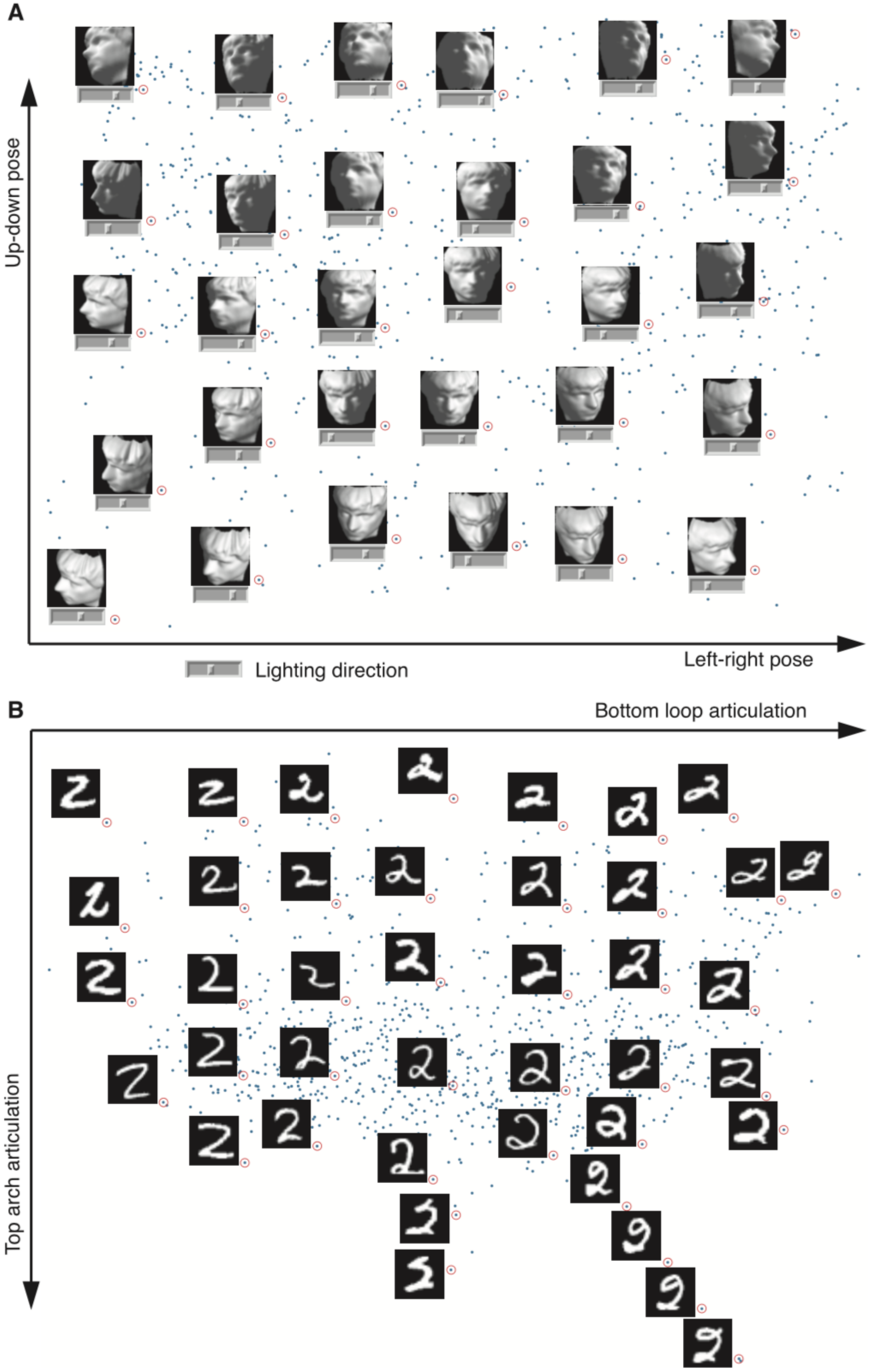


**Figure 23.12:** Plots showing degrees of freedom manifolds for images of faces (3D object) and handwritten digits [291].

**Kernel Principal Component Analysis** High-dimensional datasets often have a **non-linear nature**, in the sense that a linear PCA may only weakly capture/explain the variance across the entire dataset.

This is, in part, due to PCA relying on Euclidean distance as opposed to **geodesic distance** – the distance between two points *along* the manifold, that is to say, the distance that would be recorded if the high-dimensional object was first unrolled (see Figure 23.13).

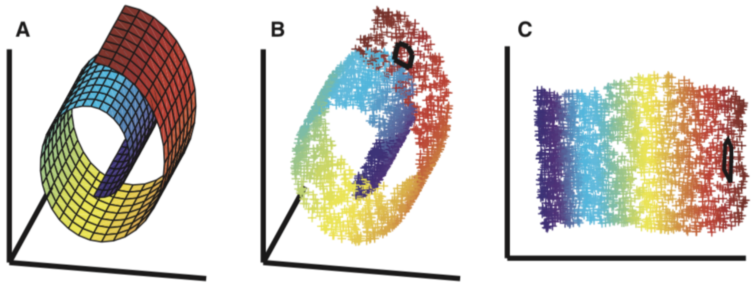


**Figure 23.13:** High-dimensional manifold unfolding; theoretical 2D manifold embedded in $\mathbb{R}^3$ (left), sample (middle), unfolding into $\mathbb{R}^2$ (right) [291].

Residents of Earth[19] are familiar with this concept: the Euclidean distance ("as the mole burrows") between Sao Paulo and Reykjavik is the length of the shortest tunnel joining the two cities, whereas the geodesic distance ("as the crow flies") is the arclength of the **great circle** through the two locations (see Figure 23.14).

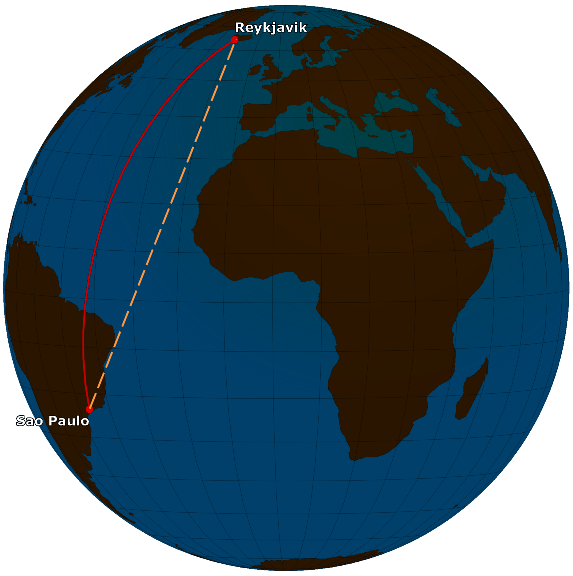


**Figure 23.14:** Geodesic (red, solid) and Euclidean (orange, dash) paths between Sao Paulo and Reykjavik, Great Circle Map ↗.

High-dimensional manifolds can be unfolded with the use of **transformations** $\Phi$ which map the input set of points

$$\{\mathbf{x}_1, \ldots, \mathbf{x}_n\} \subseteq \mathbb{R}^p$$

onto a **new set** of points

$$\{\Phi(\mathbf{x}_1), \ldots, \Phi(\mathbf{x}_n)\} \subseteq \mathbb{R}^m,$$

with $m \geq n$.

[19] Which we assume encompasses all of this work's readership. . .

If $\Phi$ is chosen so that $\sum_{i=1}^{n}\Phi(\mathbf{x}_i)=\mathbf{0}$ (i.e., the transformed data is also centered in $\mathbb{R}^m$), we can formulate the kernel PCA objective in $\mathbb{R}^p$ as a linear PCA objective in $\mathbb{R}^m$:

$$\min\left\{\sum_{i=1}^{n}\|\Phi(\mathbf{x}_i)-V_qV_q^{\top}\Phi(\mathbf{x}_i)\|^2\right\},$$

over the set of $m\times q$ matrices $V_q$ with orthonormal columns, where $q$ is the desired dimension of the manifold.[20]

In practice, it can be difficult to determine $\Phi$ **explicitly**; in many instances, it is inner-product-like quantities that are of interest to the analyst.

The problem can be bypassed by working with **positive-definite kernel functions** $K:\mathbb{R}^p\times\mathbb{R}^p\to\mathbb{R}_+$ which satisfy $K(\mathbf{x},\mathbf{y})=K(\mathbf{y},\mathbf{x})$ for all $\mathbf{x},\mathbf{y}\in\mathbb{R}^p$ and

$$\sum_{i=1}^{k}\sum_{j=1}^{k}c_ic_jK(\mathbf{x}_i,\mathbf{x}_j)\geq 0$$

for any integer $k$, coefficients $c_1,\ldots,c_k\in\mathbb{R}$ and vectors $\mathbf{x}_1,\ldots,\mathbf{x}_k\in\mathbb{R}^p$, with equality if and only if $c_1,\cdots,c_k=0$.[21]

Popular data analysis kernels include:

- **linear kernel** $K(\mathbf{x},\mathbf{y})=\mathbf{x}^{\top}\mathbf{y}$;
- **ploynomial kernel** $K(\mathbf{x},\mathbf{y})=(\mathbf{x}^{\top}\mathbf{y}+r)^k$, $n\in\mathbb{N}$, $r\geq 0$, and
- **Gaussian kernel** $K(\mathbf{x},\mathbf{y})=\exp\left\{\frac{-\|\mathbf{x}-\mathbf{y}\|}{2\sigma^2}\right\}$, $\sigma>0$.

Most dimension reduction algorithms can be re-expressed as some form of kernel PCA, as we will see shortly.

**Locally Linear Embedding** LLE is a manifold learning approach which addresses the problem of nonlinear dimension reduction by computing a low-dimensional, **neighbourhood-preserving embedding** of high-dimensional data.

The main assumption is that for any subset $\{\mathbf{x}_i\}\subseteq\mathbb{R}^p$ lying on some sufficiently well-behaved underlying $d-$dimensional manifold $\mathcal{M}$, each data point and its neighbours lie on a **locally linear patch** of $\mathcal{M}$. Using translations, rotations, and rescaling, the (high-dimensional) coordinates of each locally linear neighbourhood can be mapped to a set of $d-$dimensional global coordinates of $\mathcal{M}$.

This needs to be done in such a way that the relationships between neighbouring points are **preserved**. This can be achieved in 3 steps:

1. identify the **punctured neighbourhood** $N_i=\{i_1,\ldots,i_k\}$ of each data point $\mathbf{x}_i$ *via* $k$ nearest neighbours;[22]
2. find the weights $z_{i,j}$ that provide the best linear reconstruction of each $\mathbf{x}_i\in\mathbb{R}^p$ from their respective punctured neighbourhoods[23] , i.e., solve

$$\min_{\mathbf{W}}\left\{\sum_{i=1}^{n}\left\|\mathbf{x}_i-\sum_{j\in N_i}z_{i,j}\mathbf{x}_{N_i(j)}\right\|^2\right\},$$

where $\mathbf{Z}=\left(z_{i,j}\right)$ is an $n\times n$ matrix ($z_{i,j}=0$ if $j\notin N_i$), and

[20] This **error reconstruction** approach to PCA yields the same results as the **covariance** approach of the previous section [2].

[21] These kernels also appear in **support vector machines** (see Section 21.4.2).

[22] This could also be done by selecting all points within some fixed radius $\varepsilon$, but $k$ is not a constant anymore, and that complicates matters.

[23] Excluding $\mathbf{x}_i$ itself.

3. find the low-dimensional **embedding** vectors $\mathbf{y}_i \in \mathcal{M}(\subseteq \mathbb{R}^d)$ and neighbours $\mathbf{y}_{N_i(j)} \in \mathcal{M}$ for each $i$ which are best reconstructed by the weights determined in the previous step, i.e., solve

$$\min_{\mathbf{Y}} \left\{ \sum_{i=1}^{n} \left\| \mathbf{y}_i - \sum_{j \in N_i} w_{i,j} \mathbf{y}_{N_i(j)} \right\|^2 \right\} = \min_{\mathbf{Y}} \left\{ \mathrm{Tr}\left( \mathbf{Y}^\top \mathbf{Y} L \right) \right\},$$

where $L = (I - \mathbf{Z})^\top (I - \mathbf{Z})$ and $\mathbf{Y}$ is an $n \times d$ matrix.

If the global coordinates of the sampled points are **centered at the origin** and have **unit variance**,[24] it can be shown that $L$ has a null eigenvalue with associated eigenvector.

The $j^{\text{th}}$ column of $\mathbf{Y}$ is the eigenvector associated with the $j^{\text{th}}$ smallest non-zero eigenvalue of $L$ [292].

**Laplacian Eigenmap** LE is similar to LLE, except that the first step consists in constructing a **weighted graph** $\mathcal{G}$ with $n$ nodes (number of $p-$dimensional observations) and a set of edges connecting the neighbouring points.[25]

In practice, the edges' **weights** are determined either:

- by using the inverse exponential with respect to the Euclidean distance
$$w_{i,j} = \exp\left( -\frac{\|\mathbf{x}_i - \mathbf{x}_j\|^2}{s} \right),$$
for all $i, j$, for some parameter $s > 0$, or
- by setting $w_{i,j} = 1$, for all $i, j$.

The embedding map is then provided by the following objective

$$\min_{\mathbf{Y}} \left\{ \sum_{i=1}^{n} \sum_{j=1}^{n} w_{i,j} (\mathbf{y}_i - \mathbf{y}_j)^2 \right\} = \min_{\mathbf{Y}} \left\{ \mathrm{Tr}(\mathbf{Y} L \mathbf{Y}^\top) \right\},$$

subject to appropriate constraints, with the **Laplacian** $L = D - W$, where $D$ is the (diagonal) **degree matrix** of $\mathcal{G}$,[26] and $W$ its **weight matrix**.

The Laplacian eigenmap construction is identical to the LLE construction, save for their definition of $L$.

**Isomap** This approach follows the same steps as LLE except that it uses **geodesic distance** instead of Euclidean distance when looking for each point's neighbours.[27]

These neighbourhood relations are represented by a graph $\mathcal{G}$ in which each observation is connected to its neighbours *via* edges with weight $d_x(i, j)$ between neighbours.

The geodesic distances $d_{\mathcal{M}}(i, j)$ between all pairs of points on the manifold $\mathcal{M}$ are then estimated in the second step.

[24] Which can always be achieved with an appropriate set of restrictions.

[25] As with LLE, the edges of $\mathcal{G}$ can be obtained by finding the $k$ nearest neighbours of each node, or by selecting all points within some fixed radius $\varepsilon$.

[26] The sum of weights emanating from each node.

[27] As always, neighbourhoods can be selected with $k$NN or with a fixed $\varepsilon$.

**Semidefinite Embedding** SDE requires learning $K(\mathbf{x},\mathbf{z}) = \Phi(\mathbf{x})^\top\Phi(\mathbf{z})$ from the data before applying the kernel PCA transformation $\Phi$, which is achieved by formulating the problem of learning $K$ as an instance of **semidefinite programming**.

The distances and angles between observations and their neighbours are preserved under transformations by $\Phi$: $\|\Phi(\mathbf{x_i}) - \Phi(\mathbf{x}_j)\|^2 = \|\mathbf{x}_i - \mathbf{x}_j\|^2$, for all $\mathbf{x}_i, \mathbf{x}_j \in \mathbb{R}^p$.

In terms of the kernel matrix, this constraint can be written as

$$K(\mathbf{x}_i,\mathbf{x}_i) - 2K(\mathbf{x}_i,\mathbf{x}_j) + K(\mathbf{x}_j,\mathbf{x}_j) = \|\mathbf{x}_i - \mathbf{x}_j\|^2,$$

for all $\mathbf{x}_i, \mathbf{x}_j \in \mathbb{R}^p$.

By adding an objective function to maximize $\text{Tr}(K)$, that is, the variance of the observations in the learned feature space, SDE constructs a semidefinite program for learning the **kernel matrix**

$$K = \left(K_{i,j}\right)_{i,j=1}^n = \left(K(\mathbf{x}_i,\mathbf{x}_j)\right)_{i,j=1}^n,$$

from which we can proceed with kernel PCA.

**Unified Framework** All of the above algorithms (LLE, Laplacian Eigenmaps, Isomap, SDE) can be rewritten in the **kernel PCA** framework:

- in the case of LLE, if $\lambda_{\max}$ is the largest eigenvalue of

  $$L = (I - \mathbf{W})^\top(I - \mathbf{W}),$$

  then $K_{\text{LLE}} = \lambda_{\max} I - L$;
- with $L = D - W$, $D$ a (diagonal) degree matrix with $D_{i,i} = \sum_{j=1}^n W_{i,j}$, then the corresponding $K_{\text{LE}}$ is related to **commute times of diffusion** on the underlying graph, and
- with the Isomap element-wise squared geodesic distance matrix $\mathcal{D}^2$,

  $$K_{\text{Isomap}} = -\frac{1}{2}\left(I - \frac{1}{p}\mathbf{e}\mathbf{e}^\top\right)\mathcal{D}^2\left(I - \frac{1}{p}\mathbf{e}\mathbf{e}^\top\right),$$

  where $\mathbf{e}$ is a $p-$dimensional vector consisting solely of 1's (note that this kernel is not always positive semi-definite).

***t*−SNE** There are a few relatively new manifold learning techniques that do not fit neatly in the kernel PCA framework: Uniform Manifold Approximation and Projection (UMAP, Section 23.4.4) and $t-$**Distributed Stochastic Neighbour Embedding** ($t-$SNE).

For a dataset $\{\mathbf{x}_i\}_{i=1}^n \subseteq \mathbb{R}^p$, the latter involves calculating probabilities

$$p_{i,j} = \frac{1}{2n}\left\{\frac{\exp(-\|\mathbf{x}_i - \mathbf{x}_j\|^2/2\sigma_i^2)}{\sum_{k\neq i}\exp(-\|\mathbf{x}_i - \mathbf{x}_k\|^2/2\sigma_i^2)} + \frac{\exp(-\|\mathbf{x}_i - \mathbf{x}_j\|^2/2\sigma_j^2)}{\sum_{k\neq j}\exp(-\|\mathbf{x}_j - \mathbf{x}_k\|^2/2\sigma_j^2)}\right\},$$

which are proportional to the similarity of points in high-dimensional space $\mathbb{R}^p$ for all $i,j$, and $p_{i,i}$ is set to 0 for all $i$.[28] The bandwidths $\sigma_i$ are selected in such a way that they are smaller in denser data areas.

[28] The first component in the similarity metric measures how likely it is that $\mathbf{x}_i$ would choose $\mathbf{x}_j$ as its neighbour if neighbours were sampled from a Gaussian centered at $\mathbf{x}_i$, for all $i,j$.

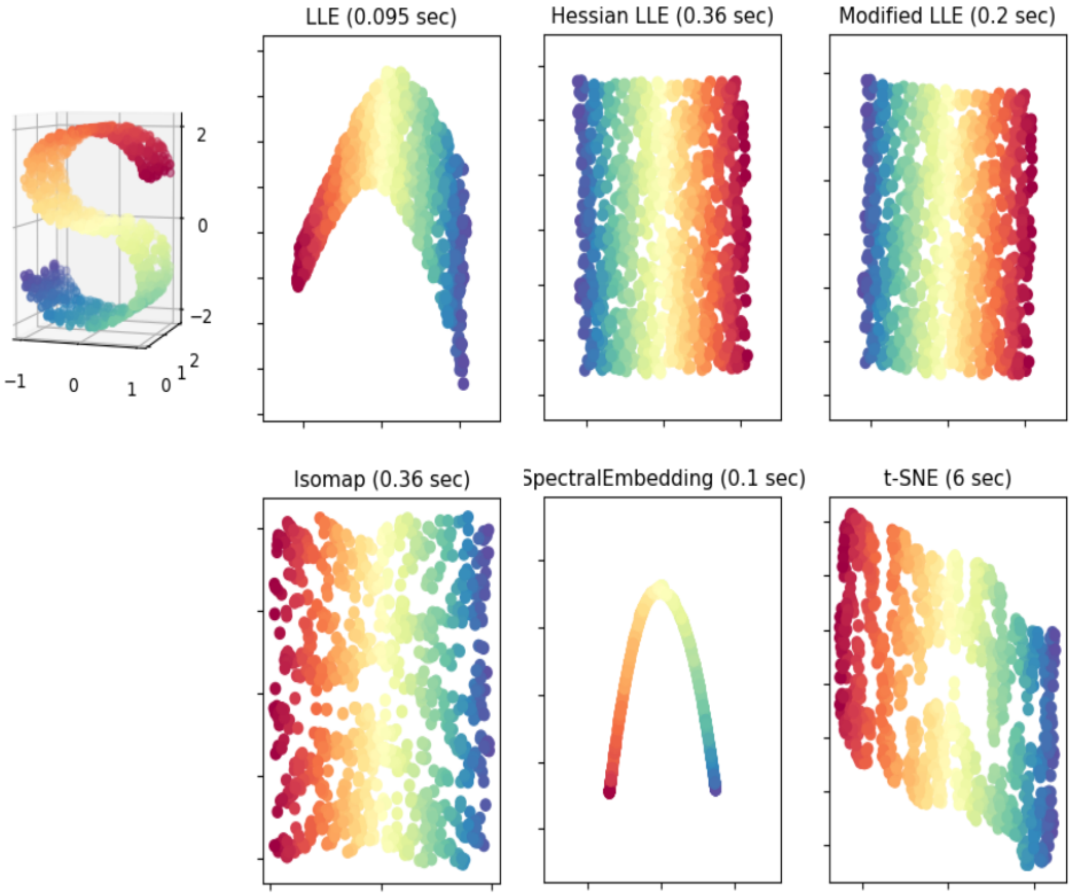


**Figure 23.15:** Comparison of manifold learning methods on an artificial dataset [294].

The lower-dimensional manifold $\{\mathbf{y}_i\}_{i=1}^n \subseteq \mathcal{M} \subseteq \mathbb{R}^d$ is selected in such a way as to preserve the similarities $p_{i,j}$ as much as possible; this can be achieved by building the (reduced) probabilities

$$q_{i,j} = \frac{(1 + \|\mathbf{y}_i - \mathbf{y}_j\|^2)^{-1}}{\sum_{k \neq i}(1 + \|\mathbf{y}_i - \mathbf{y}_k\|^2)^{-1}}$$

for all $i, j$ (note the asymmetry) and minimizing the **Kullback-Leibler divergence** of $Q$ from $P$:

$$\mathrm{KL}(P\|Q) = \sum_{i \neq j} p_{i,j} \log \frac{p_{i,j}}{q_{i,j}}$$

over possible coordinates $\{\mathbf{y}_i\}_{i=1}^n$ [293].

**MNIST Example** In [294], the methods above are used to learn $2D$ manifolds for the MNIST dataset [295], a database of handwritten digits. The results for 4 of those are shown in Figure 23.17. The analysis of optimal manifold learning methods remains fairly subjective, as it depends not only on the outcome, but also on how much computing power is used and how long it takes to obtain the mapping.

Naïvely, one would expect to see the coordinates in the reduced manifold congregate in 10 (or more) distinct groups; in that regard, $t-$SNE seems to perform admirably on MNIST.

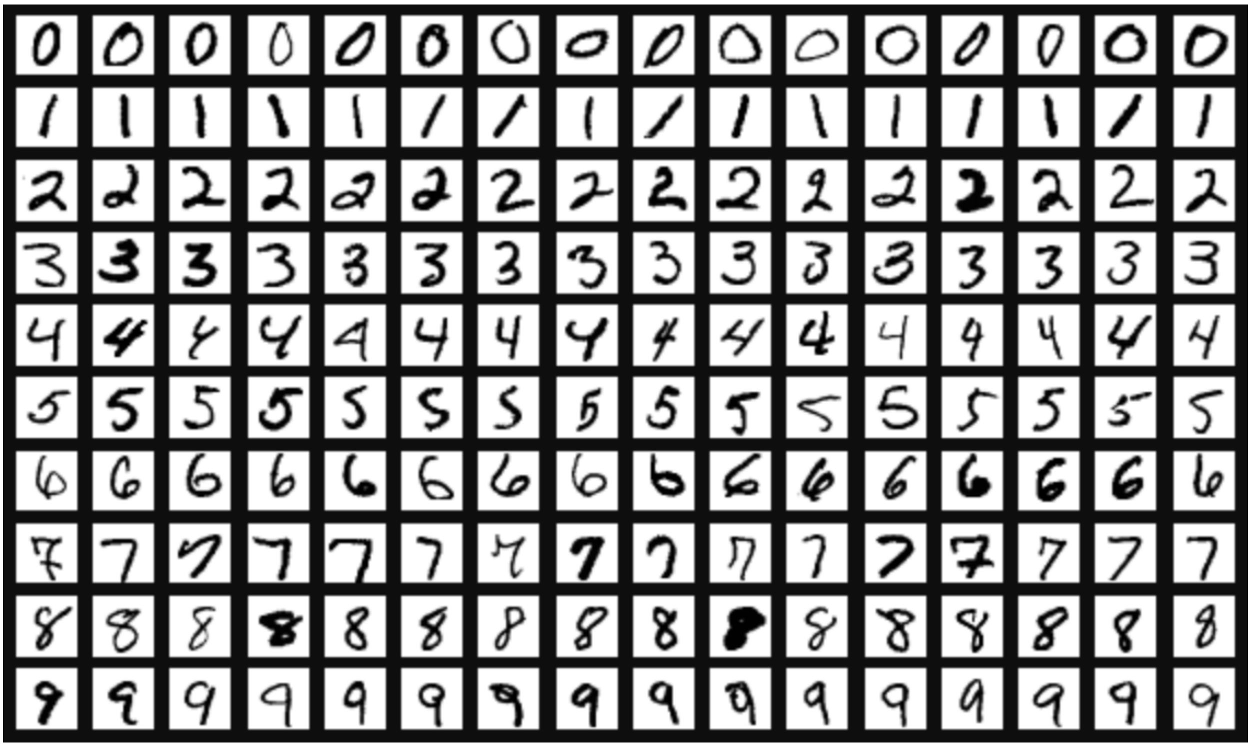


**Figure 23.16:** Sample of the MNIST dataset [294, 295].

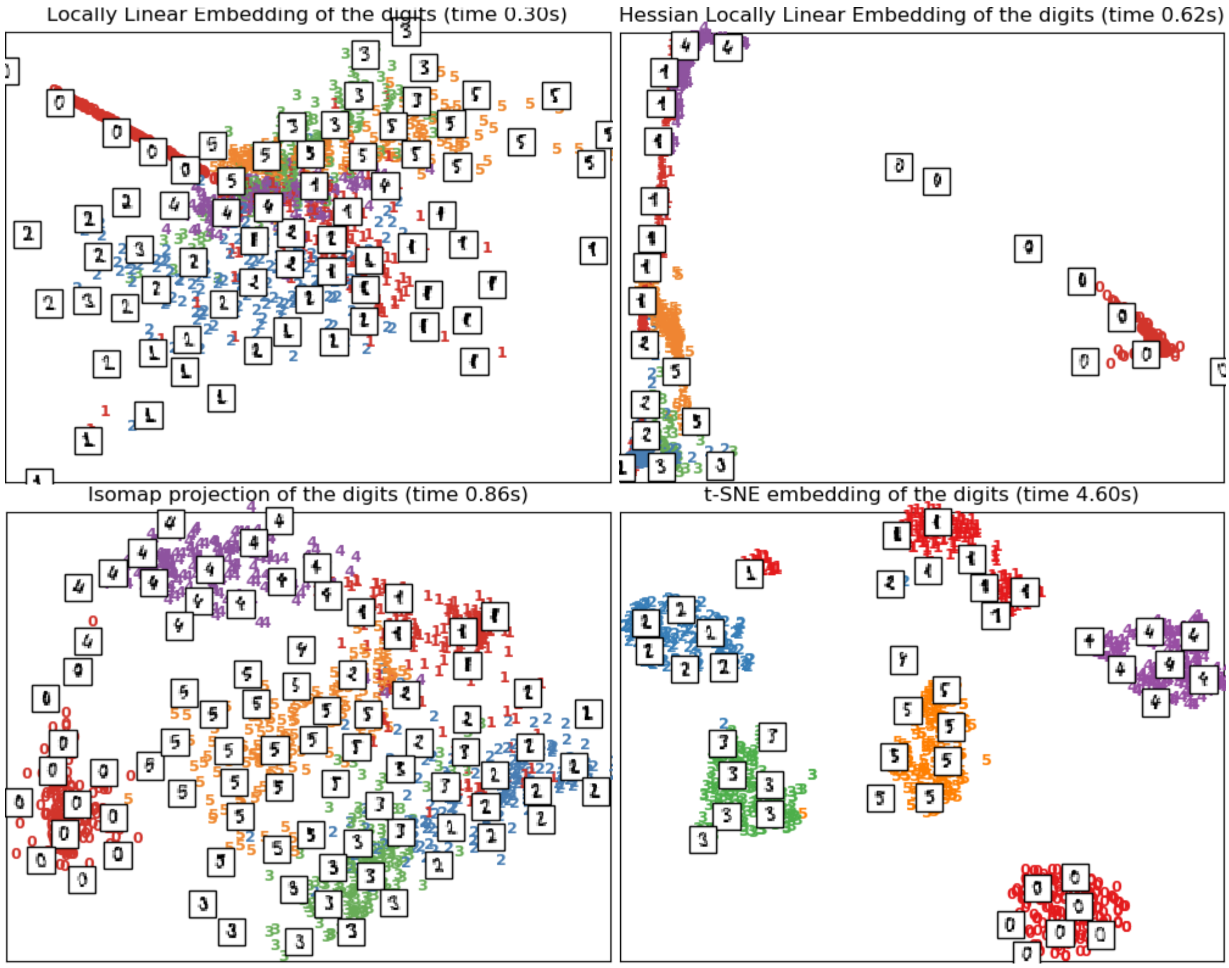


**Figure 23.17:** Manifold learning on the $0-5$ subset of MNIST: LLE, Hessian LLE, Isomap, $t$−SNE [294, 296].

## 23.3 Feature Selection

As seen in the previous section, dimension reduction methods can be used to learn low-dimensional manifolds for high-dimensional data, with the hope that the resulting loss in information content can be kept small. Unfortunately, this is not always feasible.

There is a non-technical, yet more problematic, issue with manifold learning techniques: the reduction often fails to provide an easily **interpretable** set of coordinates in the context of the original dataset.

For instance, in a dataset with the 4 features

$$X_1 = \text{Age},\ X_2 = \text{Height},\ X_3 = \text{Weight},\ \text{and}\ X_4 = \text{Rural}\ \in \{0,1\},$$

say, it is straightforward to **justify** a data-driven decision based on the rule $X_1 = \text{Age} > 25$, for example, but perhaps substantially harder to do so for a reduced rule such as

$$Y_2 = 3(\text{Age}-\overline{\text{Age}})-(\text{Height}-\overline{\text{Height}})+4(\text{Weight}-\overline{\text{Weight}})+\text{Rural} > 7,$$

even if there is nothing wrong with the rule from a technical perspective.

Furthermore, datasets often contain **irrelevant** and/or **redundant** features; identifying and removing these variables is a common data processing task. The motivations for doing so are varied, but usually fall into one of two categories:

- the modeling tools do not handle redundant variables well, due to **variance inflation** or similar issues, and
- as an attempt by analysts to overcome the **curse of dimensionality** or to avoid situations where the number of variables is larger than the number of observations.

In light of the comment above, the goal of **feature selection** is to remove (and not to transform or project) any attribute that adds noise and reduces the performance of a model, that is to say, to retain a subset of the most **relevant features**[29] , which can help create simpler models, decrease a statistical learner's training time, and reduce overfitting.

There are various feature selection methods, typically falling in one of three families – **filter** methods, **wrapper** methods, and **embedded** methods (the next two sections are inspired by [6]):

- **filter methods** focus on the relevance of the features, applying a specific **ranking metric** to each feature, individually. The variables that do not meet a preset benchmark[30] are then removed from consideration, yielding a subset of the most relevant features according to the selected ranking metric; different metrics, and different thresholds, might retain different relevant features;
- **wrapper methods** focus on the usefulness of each feature to the task of interest (usually classification or regression), but do not consider features individually; rather, they evaluate and compare the performance of different combinations of features in order to select the best-performing subset of features, and
- **embedded methods** are a combination of both, using implicit metrics to evaluate the performance of various subsets.

[29] This usually requires there to be a value to predict, against which the features can be evaluated for relevance; this is discussed further in Chapters 20 and 21.

[30] Either a threshold on the ranking or on the ranking metric value itself.

Feature selection methods can also be categorized as **unsupervised** or **supervised**:

- **unsupervised methods** determine the importance of features using only their values (with potential feature interactions), while
- **supervised methods** evaluate each feature's importance in relationship with the **target feature**.

Wrapper methods are typically supervised. Unsupervised filter methods search for **noisy** features and include the removal of **constant** variables, of **ID-like** variables (i.e. different on all observations), and of features with **low variability**.

### 23.3.1 Filter Methods

Filter methods evaluate features without resorting to the use of a classification/regression algorithm. These methods can either be

- **univariate**, where each feature is ranked independently, or
- **multivariate**, where features are considered jointly.

A filter criterion is chosen based on which metric suits the data or problem best.[31] The selected criterion is used to assign a score to, and rank, the features which are then retained or removed in order to yield a **relevant** subset of features.

Features whose score lies above (or below, as the case may be) some pre-selected threshold $\tau$ are retained (or removed); alternatively, features whose rank lies below (or above as the case may be) some pre-selected threshold $\nu$ are retained (or removed).

Such methods are advantageous in that they are computationally efficient. They also tend to be robust against overfitting as they do not incorporate the effects of the feature subset selection on classification/regression performance.

There are a number of commonly-used filter criteria, including the **Pearson correlation coefficient**, **information gain** (or mutual information), and **relief** [6].

Throughout, let $Y$ be the target variable (assuming that there is one), and $X_1, \ldots, X_p$ be the predictors.

**Pearson Correlation Coefficient** This value quantifies the linear relationship between two continuous variables [297].

For a predictor $X_i$, the **Pearson correlation coefficient** between $X_i$ and $Y$ is

$$\rho_i = \frac{\mathrm{Cov}(X_i, Y)}{\sigma_{X_i}\sigma_Y}.$$

Features for which $|\rho_i|$ is **large** (near 1) are linearly (or anti-) correlated with $Y$, those for which $|\rho_i| \approx 0$ are not linearly (nor anti-linearly) correlated with $Y$.[32]

Only those features with (relatively) strong linear (or anti-linear) correlation are retained.

[31] This can be quite difficult to determine.

[32] Which could mean that they are uncorrelated with $Y$, or that the correlation is not linear or anti-linear.

This correlation $\rho_i$ is only defined if **both** the predictor $X_i$ and the outcome $Y$ are **numerical**; there are alternatives for categorical $X_i$ and $Y$, or mixed categorical-numerical $X_i$ and $Y$ [298–300].

In order to get a better handle on what filter feature selection looks like in practice, consider the *Global Cities Index* dataset [301], which ranks prominent cities around the globe on a general scale of “Alpha”, “Beta”, “Gamma”, and “Sufficiency” (1, 2, 3, 4, respectively).

This dataset contains geographical, population, and economics data for 68 ranked cities.

```
globalcities <- data.frame(read.csv("globalcities.csv",
    stringsAsFactors = TRUE))
colnames(globalcities)[2:7] <- c("City.Area","Metro.Area",
    "City.Pop","Metro.Pop", "Ann.Pop.Growth", "GDPpc")
colnames(globalcities)[12:17] <- c("Higher.Ed.Insts",
    "Life.Exp.M","Life.Exp.F","Hospitals", "Museums",
    "Air.Quality")
str(globalcities)
```

```
’data.frame’:   68 obs. of  18 variables:
 $ Rating           : int  1 3 2 1 1 1 2 1 2 1 ...
 $ City.Area        : num  165 30.7 38.9 1569 102.6 ...
 $ Metro.Area       : num  807 25437 381 7762 3236 ...
 $ City.Pop         : num  0.76 3.54 0.66 5.72 1.62 ...
 $ Metro.Pop        : num  1.4 4.77 4.01 6.5 3.23 ...
 $ Ann.Pop.Growth   : num  0.01 0.26 0 0.03 0.01 0.04 0 0.01 0.01 0.01 ...
 $ GDPpc            : num  46 21.2 30.5 23.4 36.3 20.3 33.3 15.9 69.3 45.6 ...
 $ Unemployment.Rate: num  0.05 0.12 0.16 0.02 0.15 0.01 0.16 0.1 0.07 0.16 ...
 $ Poverty.Rate     : num  0.18 0.2 0.2 0 0.2 0.01 0.22 0.22 0.17 0.26 ...
 $ Major.Airports   : int  1 1 1 2 1 1 2 1 1 2 ...
 $ Major.Ports      : int  1 0 1 1 1 0 2 0 1 1 ...
 $ Higher.Ed.Insts  : int  23 10 21 37 8 89 30 19 35 25 ...
 $ Life.Exp.M       : num  76.3 75.3 78 69 79 79 82 74.6 74.8 77 ...
 $ Life.Exp.F       : num  80.8 80.8 83.7 74 85.2 83 88 79.7 81.1 82.6 ...
 $ Hospitals        : int  7 7 23 173 45 551 79 22 12 43 ...
 $ Museums          : int  68 36 47 27 69 156 170 76 30 25 ...
 $ Air.Quality      : int  24 46 41 54 35 121 26 77 17 28 ...
 $ Life.Expectancy  : num  78.5 78 80.8 71.5 82.1 ...
```

The R package FSelector contains feature selection tools, including various filter methods (such as chi-squared score, consistency, various entropy-based filters, etc.). Using its filtering functions, we extract the most relevant features to the ranking of a global city (we treat the Rating variable as a numerical response: is this justifiable?).

For instance, if we retain the 5 top predictors for **linear correlation** (Pearson’s correlation coefficient) with the response Rating, we obtain:

```
(lincor <- FSelector::linear.correlation(
    formula = ‘Rating‘ ~ ., data = globalcities))
```

```
                  attr_importance
City.Area            0.0007479646
Metro.Area           0.0055023564
City.Pop             0.1196632421
Metro.Pop            0.2030923952
Ann.Pop.Growth       0.1935336738
GDPpc                0.2090866065
Unemployment.Rate    0.2173999333
Poverty.Rate         0.0536585758
Major.Airports       0.2263265771
Major.Ports          0.0563507487
Higher.Ed.Insts      0.0547453393
Life.Exp.M           0.1302972404
Life.Exp.F           0.1412093767
Hospitals            0.1195832079
Museums              0.1553072283
Air.Quality          0.1382099362
Life.Expectancy      0.1380603739
```

```
(subset_lincor <- FSelector::cutoff.k(lincor, k = 5))
```

```
[1] "Major.Airports"    "Unemployment.Rate"    "GDPpc"
[4] "Metro.Pop"         "Ann.Pop.Growth"
```

According to the **linear correlation** feature selection method, the 5 “best’ ’ features that relate to a city’s global ranking are the number of major airports it has, its unemployment rate, its GDP per capita, its metropolitan population, and its annual population growth.[33]

**Mutual Information** **Information gain** is a popular **entropy-based** method of feature selection that measures the amount of dependence between features by quantifying the amount of mutual information between them. In general, this quantifies the **amount of information** obtained about a predictor $X_i$ by observing the target feature $Y$.

Mutual information can be expressed as

$$\mathrm{IG}(X_i;Y) = H(X_i) - H(X_i \mid Y),$$

where $H(X_i)$ is the **marginal entropy** of $X_i$ and $H(X_i \mid Y)$ is the **conditional entropy** of $X_i$ given $Y$ [302], an

$$H(X_i) = \mathrm{E}_{X_i}[-\log p(X_i)], \quad H(X_i \mid Y) = \mathrm{E}_{(X_i,Y)}[-\log p(X_i \mid Y)],$$

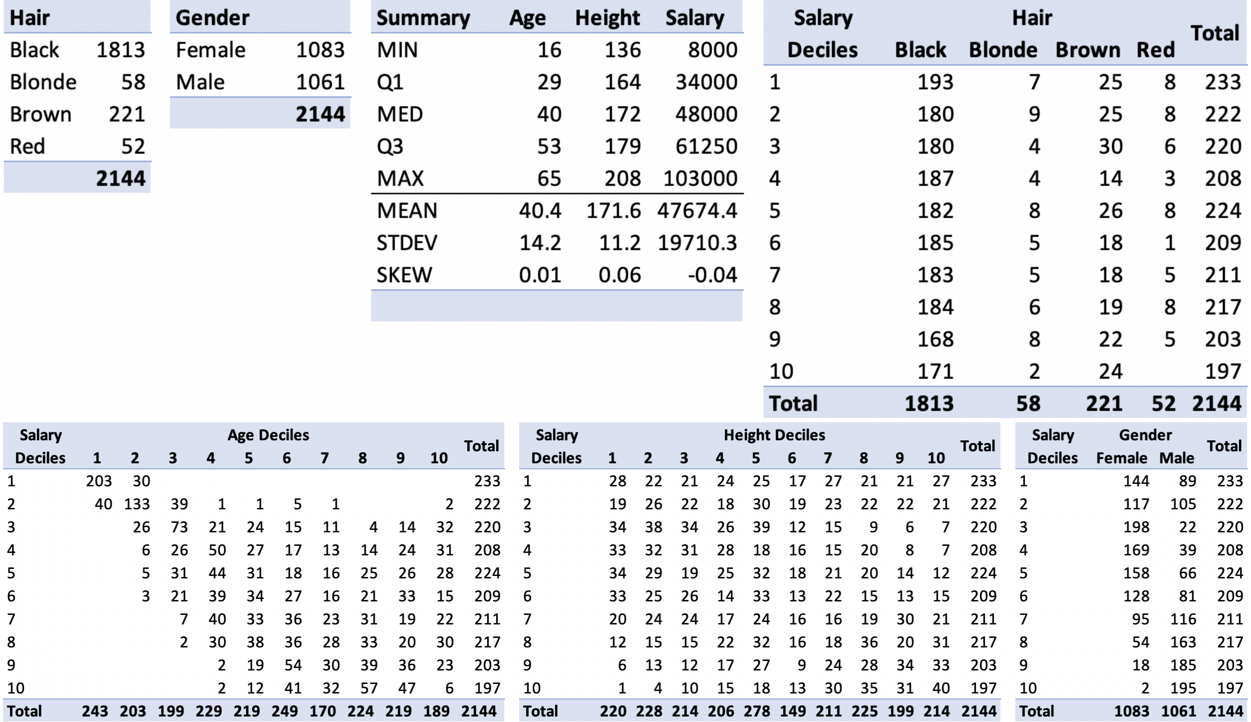
where $p(X_i)$ and $p(X_i \mid Y)$ are the probability density functions of the random variables $X_i$ and $X_i \mid Y$, respectively.

How is IG interpreted? Consider the following example: let $Y$ represent the salary of an individual (continuous), $X_1$ their hair colour (categorical), $X_2$ their age (continuous), $X_3$ their height (continuous), and $X_4$ their self-reported gender (categorical). A sample of $n = 2144$ individuals is found in `demo1.csv`.

[33] As filtering is a **pre-processing step**, proper analysis would also require building a model using this subset of features.

```
salary <- data.frame(read.csv("demo1.csv",
    stringsAsFactors = TRUE))
colnames(salary)[1] <- c("Hair")
```

Some summary statistics are shown below:



| Hair | |
|---|---|
| Black | 1813 |
| Blonde | 58 |
| Brown | 221 |
| Red | 52 |
| | **2144** |

| Gender | |
|---|---|
| Female | 1083 |
| Male | 1061 |
| | **2144** |

| Summary | Age | Height | Salary |
|---|---|---|---|
| MIN | 16 | 136 | 8000 |
| Q1 | 29 | 164 | 34000 |
| MED | 40 | 172 | 48000 |
| Q3 | 53 | 179 | 61250 |
| MAX | 65 | 208 | 103000 |
| MEAN | 40.4 | 171.6 | 47674.4 |
| STDEV | 14.2 | 11.2 | 19710.3 |
| SKEW | 0.01 | 0.06 | -0.04 |

| Salary | Hair | | | | Total |
|---|---|---|---|---|---|
| Deciles | Black | Blonde | Brown | Red | |
| 1 | 193 | 7 | 25 | 8 | 233 |
| 2 | 180 | 9 | 25 | 8 | 222 |
| 3 | 180 | 4 | 30 | 6 | 220 |
| 4 | 187 | 4 | 14 | 3 | 208 |
| 5 | 182 | 8 | 26 | 8 | 224 |
| 6 | 185 | 5 | 18 | 1 | 209 |
| 7 | 183 | 5 | 18 | 5 | 211 |
| 8 | 184 | 6 | 19 | 8 | 217 |
| 9 | 168 | 8 | 22 | 5 | 203 |
| 10 | 171 | 2 | 24 | | 197 |
| **Total** | **1813** | **58** | **221** | **52** | **2144** |

| Salary | Age Deciles | | | | | | | | | | Total |
|---|---|---|---|---|---|---|---|---|---|---|---|
| Deciles | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | |
| 1 | 203 | 30 | | | | | | | | | 233 |
| 2 | 40 | 133 | 39 | 1 | 1 | 5 | 1 | | | 2 | 222 |
| 3 | | 26 | 73 | 21 | 24 | 15 | 11 | 4 | 14 | 32 | 220 |
| 4 | | 6 | 26 | 50 | 27 | 17 | 13 | 14 | 24 | 31 | 208 |
| 5 | | 5 | 31 | 44 | 31 | 18 | 16 | 25 | 26 | 28 | 224 |
| 6 | | 3 | 21 | 39 | 34 | 27 | 16 | 21 | 33 | 15 | 209 |
| 7 | | | 7 | 40 | 33 | 36 | 23 | 31 | 19 | 22 | 211 |
| 8 | | | 2 | 30 | 38 | 36 | 28 | 33 | 20 | 30 | 217 |
| 9 | | | | 2 | 19 | 54 | 30 | 39 | 36 | 23 | 203 |
| 10 | | | | 2 | 12 | 41 | 32 | 57 | 47 | 6 | 197 |
| **Total** | **243** | **203** | **199** | **229** | **219** | **249** | **170** | **224** | **219** | **189** | **2144** |

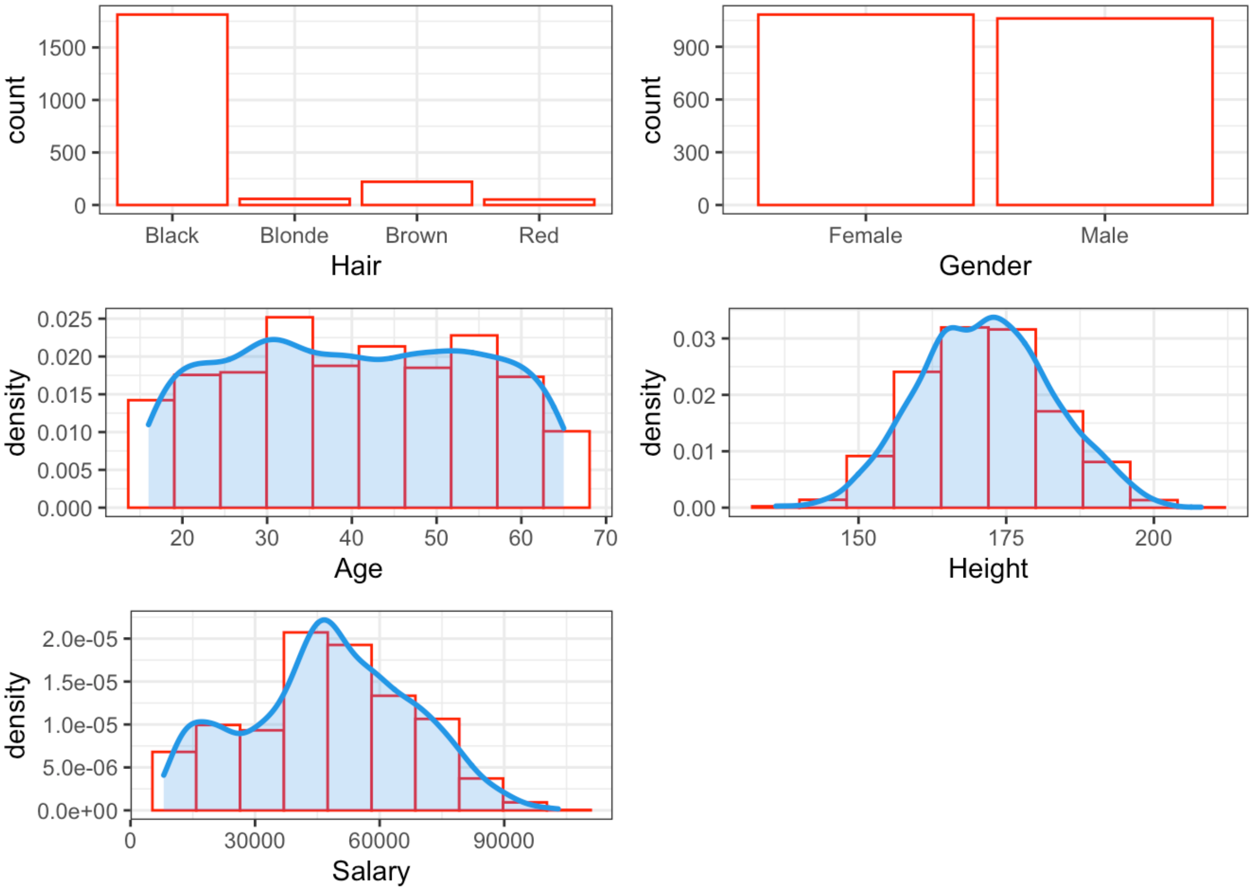
| Salary | Height Deciles | | | | | | | | | | Total |
|---|---|---|---|---|---|---|---|---|---|---|---|
| Deciles | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | |
| 1 | 28 | 22 | 21 | 24 | 25 | 17 | 27 | 21 | 21 | 27 | 233 |
| 2 | 19 | 26 | 22 | 18 | 30 | 19 | 23 | 22 | 22 | 21 | 222 |
| 3 | 34 | 38 | 34 | 26 | 39 | 12 | 15 | 9 | 6 | 7 | 220 |
| 4 | 33 | 32 | 31 | 28 | 18 | 16 | 15 | 20 | 8 | 7 | 208 |
| 5 | 34 | 29 | 19 | 25 | 32 | 18 | 21 | 20 | 14 | 12 | 224 |
| 6 | 33 | 25 | 26 | 14 | 33 | 13 | 22 | 15 | 13 | 15 | 209 |
| 7 | 20 | 24 | 24 | 17 | 24 | 16 | 16 | 19 | 30 | 21 | 211 |
| 8 | 12 | 15 | 15 | 22 | 32 | 16 | 18 | 36 | 20 | 31 | 217 |
| 9 | 6 | 13 | 12 | 17 | 27 | 9 | 24 | 28 | 34 | 33 | 203 |
| 10 | 1 | 4 | 10 | 15 | 18 | 13 | 30 | 35 | 31 | 40 | 197 |
| **Total** | **220** | **228** | **214** | **206** | **278** | **149** | **211** | **225** | **199** | **214** | **2144** |

| Salary | Gender | | Total |
|---|---|---|---|
| Deciles | Female | Male | |
| 1 | 144 | 89 | 233 |
| 2 | 117 | 105 | 222 |
| 3 | 198 | 22 | 220 |
| 4 | 169 | 39 | 208 |
| 5 | 158 | 66 | 224 |
| 6 | 128 | 81 | 209 |
| 7 | 95 | 116 | 211 |
| 8 | 54 | 163 | 217 |
| 9 | 18 | 185 | 203 |
| 10 | 2 | 195 | 197 |
| **Total** | **1083** | **1061** | **2144** |

In a general population, one would expect that the distribution of salaries, among others, is likely to be fairly haphazard, and it might be hard to explain why it has the shape that it does, specifically.

**Distributions of the predictors and the response**

```
library(ggplot2)
par(mfrow=c(3,2))
plot1 <- ggplot(salary, aes(x=Hair)) +
  geom_bar(color=’red’, fill=’white’) +
  theme_bw()

plot2 <- ggplot(salary, aes(x=Gender)) +
  geom_bar(color=’red’, fill=’white’) +
  theme_bw()

plot3 <- ggplot(salary, aes(x=Age)) +
  geom_histogram(aes(y=..density..),
  color=’red’, fill=’white’, bins=10) +
  geom_density(lwd = 1, colour = 4, fill = 4, alpha = 0.25) +
  theme_bw()

plot4 <- ggplot(salary, aes(x=Height)) +
  geom_histogram(aes(y=..density..),
  color='red', fill='white', bins=10) +
  geom_density(lwd = 1, colour = 4, fill = 4, alpha = 0.25) +
  theme_bw()

plot5 <- ggplot(salary, aes(x=Salary)) +
  geom_histogram(aes(y=..density..),
  color='red', fill='white', bins=10) +
  geom_density(lwd = 1, colour = 4, fill = 4, alpha = 0.25) +
  theme_bw()

gridExtra::grid.arrange(plot1, plot2, plot3, plot4, plot5, ncol=2)
```



Perhaps it could be explained by knowing the relationship between the salary and the other variables? It is this idea that forms the basis of mutual information feature selection.

Applying the definition, one sees that

$$
\begin{aligned}
H(X_1) &= -\sum_{\text{colour}} p(\text{colour}) \log p(\text{colour}) \\
H(X_2) &= -\int p(\text{age}) \log p(\text{age})\, d\text{age} \\
H(X_3) &= -\int p(\text{height}) \log p(\text{height})\, d\text{height} \\
H(X_4) &= -\sum_{\text{gender}} p(\text{gender}) \log p(\text{gender})
\end{aligned}
$$

$$
\begin{aligned}
H(X_1 \mid Y) &= -\int p(Y)\left\{\sum_{\text{colour}} p(\text{colour} \mid Y)\log p(\text{colour} \mid Y)\right\} dY \\
H(X_2 \mid Y) &= -\iint p(Y)p(\text{age} \mid Y)\log p(\text{age} \mid Y)\, d\text{age}\, dY \\
H(X_3 \mid Y) &= -\iint p(Y)p(\text{ht} \mid Y)\log p(\text{ht} \mid Y)\, d\text{ht}\, dY \\
H(X_4 \mid Y) &= -\int p(Y)\left\{\sum_{\text{gender}} p(\text{gender} \mid Y)\log p(\text{gender} \mid Y)\right\} dY
\end{aligned}
$$

If the theoretical distributions are known, the entropy integrals can be computed directly (or approximated using standard numerical integration methods).

Gender and hair colour can be fairly easily be modeled using multinomial distributions, but there is more uncertainty related to the numerical variables. A potential approach is to recode the continuous variables age, height, and salary as **decile variables** $a_d$, $h_d$, and $Y_d$ taking values $\{1,\ldots,10\}$ according to which decile of the original variable the observation falls (see decile breakdown above).

The integrals can then be replaced by sums:

$$
\begin{aligned}
H(X_1) &= -\sum_{\text{colour}} p(\text{colour})\log p(\text{colour}) \\
H(X_2) &\approx -\sum_{k=1}^{10} p(\text{a}_d = k)\log p(\text{a}_d = k) \\
H(X_3) &\approx -\sum_{k=1}^{10} p(\text{ht}_d = k)\log p(\text{ht}_d = k) \\
H(X_4) &= -\sum_{\text{gender}} p(\text{gender})\log p(\text{gender})
\end{aligned}
$$

$$
\begin{aligned}
H(X_1 \mid Y) &\approx -\sum_{j=1}^{10} p(Y_d = j)\sum_{c\in\text{colour}} p(c \mid Y_d = j)\log p(c \mid Y_d = j) \\
H(X_2 \mid Y) &\approx -\sum_{j=1}^{10} p(Y_d = j)\sum_{k=1}^{10} p(a_d = k \mid Y_d = j)\log p(a_d = k \mid Y_d = j) \\
H(X_3 \mid Y) &\approx -\sum_{j=1}^{10} p(Y_d = j)\sum_{k=1}^{10} p(h_d = k \mid Y_d = j)\log p(h_d = k \mid Y_d = j) \\
H(X_4 \mid Y) &\approx -\sum_{j=1}^{10} p(Y_d = k)\sum_{g\in\text{gender}} p(g \mid Y_d = j)\log p(g \mid Y_d = j)
\end{aligned}
$$

The results are shown below (using base 10 logarithms, and rounded out to the nearest hundredth):

| X | H(X) | H(X \| Y) | IG(X;Y) | Ratio |
|---|---|---|---|---|
| **Hair** | 0.24 | 0.24 | 0.00 | 0.00 |
| **Age** | 1.00 | 0.74 | 0.26 | 0.26 |
| **Height** | 1.00 | 0.96 | 0.03 | 0.03 |
| **Gender** | 0.30 | 0.22 | 0.08 | 0.26 |

The percentage decrease in entropy after having observed $Y$ is shown in the column "Ratio." Raw IG numbers would seem to suggest that Gender has a small link to Salary; the Ratio numbers suggest that this could be due to the way the Age and Height levels have been categorized (as deciles).

**Relief** This approach scores (numerical) features based on the identification of feature value differences between nearest-neighbour instance pairs.

If there is a feature value difference in a neighbouring instance pair **of the same class**, the score of the feature decreases; on the other hand, if there exists a feature value difference in a neighbouring instance pair with **different class values**, the feature score increases.

More specifically, let

$$D = \{(\mathbf{x}_i, y_i)\}_{i=1}^{n} \subset \mathbb{R}^p \times \{\pm 1\}$$

be a dataset where $\mathbf{x}_i$ is the $i$-th data sample and $y_n$ is its corresponding class label.

For each feature $j$ and observation $i$, two values are selected: the **near hit** $H(x_{i,j})$ is the value of $X_j$ which is nearest to $x_{i,j}$ among all instances in the same class as $\mathbf{x}_i$, while the **near miss** $M(x_{i,j})$ is the value of $X_j$ which is nearest to $x_{i,j}$ among all instances in the opposite class of $\mathbf{x}_i$.

The **Relief score** of the $j^{\text{th}}$ feature is

$$S_j^d = \frac{1}{n}\sum_{i=1}^{n}\left\{d(x_{i,j}, M(x_{i,j})) - d(x_{i,j}, H(x_{i,j}))\right\},$$

for some pre-selected distance $d : \mathbb{R} \times \mathbb{R} \to \mathbb{R}_0^+$.

A feature for which near-hits tend to be nearer to their instances than near-misses are (i.e., for which

$$d(x_{i,j}, M(x_{i,j})) > d(x_{i,j}, H(x_{i,j})),$$

on average) will yield **larger** Relief scores than those for which the opposite is true. Features are deemed **relevant** when their relief score is greater than some threshold $\tau$.

There are a variety of Relief-type measurements to accommodate potential feature interactions or multi-class problems[34] (ReliefF), but in general Relief is noise-tolerant and robust to interactions of attributes; its effectiveness decreases for small training sets, however [303].

The Relief algorithm is also implemented in the R package `CORElearn`, as are numerous other methods:

```
CORElearn::infoCore(what="attrEval") # Classification
```

```
[1] "ReliefFequalK"     "ReliefFexpRank"     "ReliefFbestK"
[4] "Relief"            "InfGain"            "GainRatio"
[7] "MDL"               "Gini"               "MyopicReliefF"
```

[34] For instance, for a $p-$distance $\delta$, set

$$H^\delta(x_{i,j}) = \arg\min_{\pi_j(\mathbf{z})}\{\delta(\mathbf{x}_i, \mathbf{z}) \mid \mathscr{C}(\mathbf{x}_i) = \mathscr{C}(\mathbf{z})\}$$

and

$$M^\delta(x_{i,j}) = \arg\min_{\pi_j(\mathbf{z})}\{\delta(\mathbf{x}_i, \mathbf{z}) \mid \mathscr{C}(\mathbf{x}_i) \neq \mathscr{C}(\mathbf{z})\}.$$

```
[10] "Accuracy"           "ReliefFmerit"       "ReliefFdistance"
[13] "ReliefFsqrDistance" "DKM"                "ReliefFexpC"
[16] "ReliefFavgC"        "ReliefFpe"          "ReliefFpa"
[19] "ReliefFsmp"         "GainRatioCost"      "DKMcost"
[22] "ReliefKukar"        "MDLsmp"             "ImpurityEuclid"
[25] "ImpurityHellinger"  "UniformDKM"         "UniformGini"
[28] "UniformInf"         "UniformAccuracy"    "EqualDKM"
[31] "EqualGini"          "EqualInf"           "EqualHellinger"
[34] "DistHellinger"      "DistAUC"            "DistAngle"
[37] "DistEuclid"
```

```
CORElearn::infoCore(what="attrEvalReg") # Regression
```

```
[1] "RReliefFequalK"      "RReliefFexpRank"     "RReliefFbestK"
[4] "RReliefFwithMSE"     "MSEofMean"           "MSEofModel"
[7] "MAEofModel"          "RReliefFdistance"    "RReliefFsqrDistance"
```

Again working on the *Global Cities Dataset*, we start by declaring the target variable Rating as a categorical variable.

```
globalcities.cat <- globalcities
globalcities.cat$Rating <- as.factor(globalcities.cat$Rating)
```

Now, let's evaluate the predictors relevance, using InfGain and ReliefFpe, say:

```
InfGain.wts <- CORElearn::attrEval(
    Rating ~ ., globalcities.cat, estimator="InfGain")
ReliefF.wts <- CORElearn::attrEval(
    Rating ~ ., globalcities.cat, estimator="ReliefFpe")
data.frame(InfGain.wts,ReliefF.wts)
```

```
                  InfGain.wts ReliefF.wts
City.Area          0.05898196 0.035227591
Metro.Area         0.10501737 0.020691027
City.Pop           0.10422894 0.050779784
Metro.Pop          0.13022307 0.006038070
Ann.Pop.Growth     0.04624721 0.005785050
GDPpc              0.12909419 0.038870952
Unemployment.Rate  0.08824268 0.069113116
Poverty.Rate       0.06966089 0.004854548
Major.Airports     0.03565266 0.015932195
Major.Ports        0.04175416 0.014967595
Higher.Ed.Insts    0.02868171 0.008031587
Life.Exp.M         0.09504723 0.119356004
Life.Exp.F         0.04937724 0.073946982
Hospitals          0.18080212 0.011268651
Museums            0.10732739 0.017808737
Air.Quality        0.05838298 0.057001715
Life.Expectancy    0.07730300 0.101021497
```

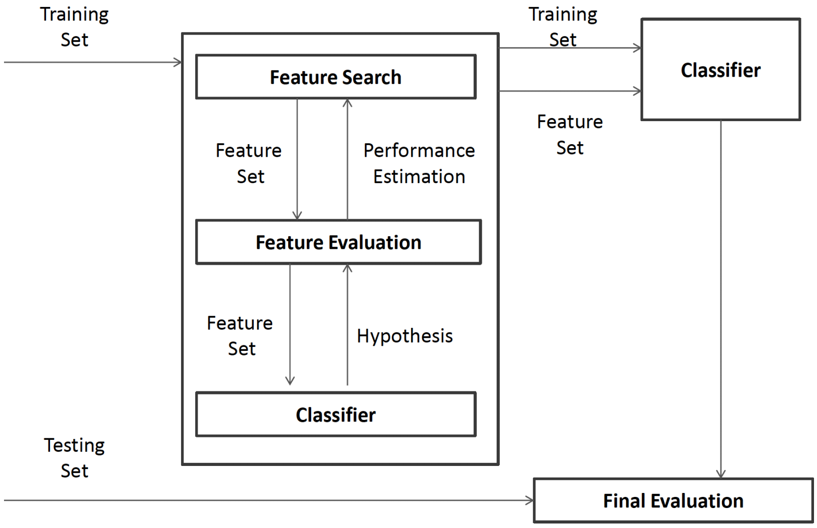


**Figure 23.18:** Feature selection process for wrapper methods in classification problems [6].

For classification tasks (categorical targets), the more relevant features are those for which the scores are **higher**.[35] How do the top-5 for each method compare in the previous example? Should this be surprising?

---

There is a multitude of other filter methods, including [4, 6]:

- **correlation metrics** (Kendall, Spearman, point-biserial, etc.);
- **entropy-based metrics** (gain ratio, symmetric uncertainty, etc.);
- **relief-type algorithms** (ReliefF, Relieved-F, etc.);
- $\chi^2-$test;
- ANOVA;
- Fisher Score;
- Gini Index;
- etc.

The list is by no means exhaustive, but it provides a fair idea of the various types of filter-based feature selection metrics.

### 23.3.2 Wrapper Methods

**Wrapper methods** offer a powerful way to address problem of variable selection. Wrapper methods evaluate the quality of subsets of features for predicting the target output under the selected predictive algorithm and select the optimal combination (for the given training set and algorithm).

In contrast to filter methods, wrapping methods are integrated directly into the classification or clustering process (see Figure 23.18 for an illustration of this process).

Wrapper methods treats feature selection as a **search problem** in which different subsets of features are explored. This process can be computationally expensive as the size of the search space increases exponentially with the number of predictors; even for modest problems an exhaustive search can quickly become impractical.

[35] 35: This is not necessarily the case for regression tasks, however – be sure to read the documentation for each method.

In general, wrapper methods iterate over the following steps, until an "optimal" set of features is identified:

- select a feature subset, and
- evaluate the performance of the selected feature subset.

The search ends once some desired quality is reached (such as adjusted $R^2$, accuracy, etc.). Various search methods are used to traverse the feature phase space and provide an approximate solution to the **optimal feature subset problem**, including: hill-climbing, best-first, and genetic algorithms.

Wrapper methods are not as efficient as filter methods and are not as robust against over-fitting. However, they are very effective at improving the model's performance due to their attempt to minimize the error rate.[36]

### 23.3.3 Subset Selection Methods

**Stepwise selection** is a form of *Occam's Razor*: at each step, a new feature is considered for inclusion or removal from the current features set based on some criterion ($F-$test, $t-$test, etc.). Greedy search methods such as backward elimination and forward selection have proven to be both quite robust against over-fitting and among the least computationally expensive wrapper methods.

**Backward elimination** begins with the full set of features and sequentially eliminates the least relevant ones until further removals increase the error rate of the predictive model above some utility threshold.

**Forward selection** works in reverse, beginning the search with an empty set of features and progressively adding relevant features to the ever growing set of predictors. In an ideal setting, model performance should be tested using **cross-validation**.

Stepwise selection methods are extremely common, but they have severe limitations (which are not usually addressed) [53, 304]:

- the tests are biased, since they are all based on the same data;
- the adjusted $R^2$ only takes into account the number of features in the final fit, and not the degrees of freedom that have been used in the entire model;
- if cross-validation is used, stepwise selection has to be repeated for each sub-model but that is not usually done, and
- it represents a classic example of $p$-hacking.

Consequently, the use of stepwise methods is **contra-indicated** in the machine learning context.

### 23.3.4 Regularization (Embedded) Methods

An interesting hybrid is provided by the **least absolute shrinkage and selection operator** (LASSO) and its variants, which are discussed in Section 20.2.4.

[36] Which unfortunately can also lead to the introduction of implicit bias in the problem [6].

### 23.3.5 Supervised and Unsupervised Feature Selection

While feature selection methods are usually categorised as filter, wrapper, or embedded, they can also be categorised as **supervised** or **unsupervised** methods. Whether a feature selection method is supervised or not boils down to whether the labels of objects/instances are incorporated into the feature reduction process (or not).

The methods that have been described in this section were all supervised.

In unsupervised methods, feature selection is carried out based on the characteristics of the attributes, without any reference to labels or a target variable. In particular, for **clustering problems** (see Section 22), only unsupervised feature selection methods can be used [5].

Unsupervised feature selection methods include:

- identifying ID-like predictors;
- identifying constant (or nearly constant) predictors;
- identifying predictors that are in a multicolinear relationship with other variables;
- identifying clusters of predictors, etc.

## 23.4 Advanced Topics

When used appropriately, the approaches to feature selection and dimension reduction methods presented in the last two sections provide a solid toolkit to help mitigate the effects of the curse of dimensionality.

However, they remain (for the most part) rather straightforward. The methods discussed in this section are decidedly more sophisticated, from a mathematical perspective; an increase in conceptual complexity can lead to insights that are out of reach of more direct approaches.

### 23.4.1 Singular Value Decomposition

From a database management perspective, it pays not to view datasets simply as flat file arrays; from an analytical perspective, however, viewing datasets as **matrices** allows analysts to use the full machinery of linear algebra and **matrix factorization** techniques, of which **singular value decomposition** (SVD) is a well-known component.[37]

As before, let $\{\mathbf{x}_i\}_{i=1}^{n} \subseteq \mathbb{R}^p$ and denote the **matrix of observations** by

$$\mathbf{X} = \begin{bmatrix} \mathbf{x}_1 \\ \mathbf{x}_2 \\ \vdots \\ \mathbf{x}_n \end{bmatrix} \in \mathbb{M}_{n,p}(\mathbb{R}) = \mathbb{R}^{n\times p}.$$

Let $d \geq \min p, n$. From a dimension reduction perspective, the goal of matrix factorization is to find two narrow matrices $\mathbf{W}_d \in \mathbb{R}^{n\times d}$ (the **cases**)

[37] Matrix factorization techniques have applications to other data analytic tasks; notably, they can be used to impute missing values and to build recommender systems.

and $\mathbf{C}_d \in \mathbb{R}^{p\times d}$ (the **concepts**) such that the product $\mathbf{W}_d\mathbf{C}_d^\top = \widetilde{\mathbf{X}}_d$ is the best rank $d$ approximation of $\mathbf{X}$, i.e.

$$\widetilde{\mathbf{X}}_d = \arg\min_{\mathbf{X}'}\{\|\mathbf{X}-\mathbf{X}'\|_F\}, \quad \text{with } \operatorname{rank}(\mathbf{X}') = d,$$

where the **Frobenius norm** $F$ of a matrix is

$$\|\mathbf{A}\|_F^2 = \sum_{i,j}|a_{i,j}|^2.$$

In a sense, $\widetilde{\mathbf{X}}_d$ is a **"smooth" representation** of $\mathbf{X}$; the dimension reduction takes place when $\mathbf{W}_d$ is used as a dense $d-$representation of $\mathbf{X}$.The link with the singular value decomposition of $\mathbf{X}$ can be made explicit: there exist orthonormal matrices $\mathbf{U} \in \mathbb{R}^{n\times n}$, $\mathbf{V} \in \mathbb{R}^{p\times p}$, and a diagonal matrix $\mathbf{\Sigma} \in \mathbb{R}^{n\times p}$ with $\sigma_{i,j} = 0$ if $i \neq j$ and $\sigma_{i,i} \geq \sigma_{i+1,i+1} \geq 0$ for all $i$,[38] such that

$$\mathbf{X} = \mathbf{U}\mathbf{\Sigma}\mathbf{V}^\top;$$

the decomposition is unique if and only if $n = p$.

Let $\mathbf{\Sigma}_d \in \mathbb{R}^{d\times d}$ be the matrix obtained by retaining the first $d$ rows and the the first $d$ columns of $\mathbf{\Sigma}$; $\mathbf{U}_d \in \mathbb{R}^{n\times d}$ be the matrix obtained by retaining the first $d$ columns of $\mathbf{U}$, and $\mathbf{V}_d^\top \in \mathbb{R}^{d\times p}$ be the matrix obtained by retaining the first $d$ rows of $\mathbf{V}^\top$.

Then

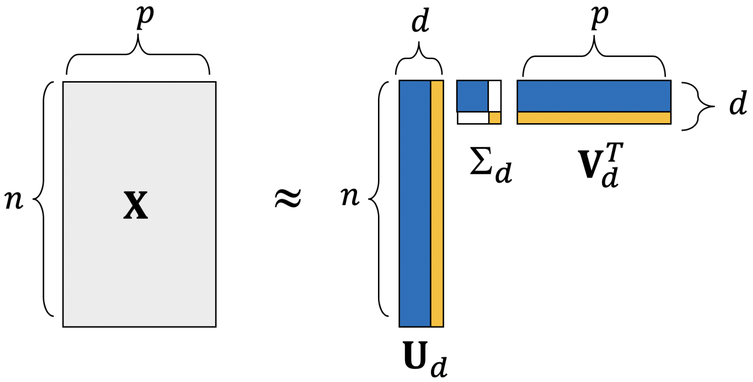
$$\widetilde{\mathbf{X}}_d = \underbrace{\mathbf{U}_d\mathbf{\Sigma}_d}_{\mathbf{W}_d}\mathbf{V}_d^\top,$$

and the $d$-dimensional rows of $\mathbf{W}_d$ are approximations of the $p-$dimensional rows of $\mathbf{X}$ in the sense that

$$\langle \mathbf{W}_d[i], \mathbf{W}_d[j]\rangle = \left\langle \widetilde{\mathbf{X}}_d[i], \widetilde{\mathbf{X}}_d[j]\right\rangle \approx \langle \mathbf{X}_d[i], \mathbf{X}_d[j]\rangle \text{ for all } i,j.$$

### Applications

1. One of the advantages of SVD is that it allows for substantial savings in data storage (modified from [204]):



Storing $\mathbf{X}$ requires $np$ saved entries, but an approximate version of the original requires only $d(n+p+d)$ saved entries; if $\mathbf{X}$ represents a 2000 × 2000 image (with 4 million entries) to be transmitted, say, a decent approximation can be sent *via* $d = 10$ using only 40100

[38] Each **singular value** is the principal square root of the corresponding eigenvalue of the covariance matrix $\mathbf{X}^\top\mathbf{X}$ (see Section 23.2.3).

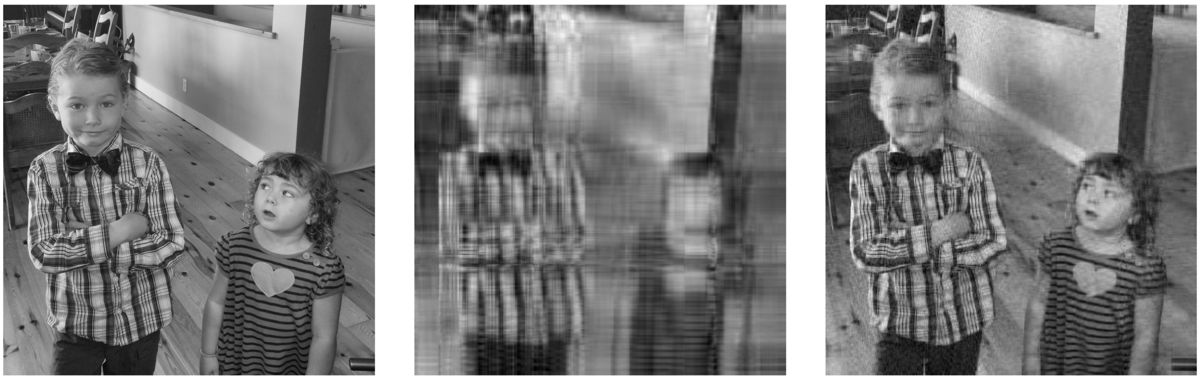


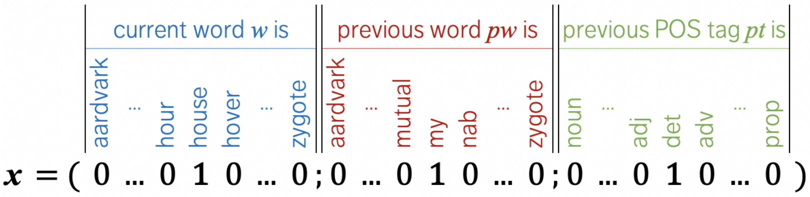
**Figure 23.19:** SVD image reconstruction: $d = 1400$ (left), $d = 10$ (middle), $d = 50$ (right); Llewellyn and Gwynneth Rayfield.

entries, roughly 1% of the original number of entries (see Figure **??** for an illustration).

2. SVD can also be used to learn **word embedding vectors**. In the traditional approach to text mining and natural language processing (NLP) (see Sections 29 and **??**), words and associated concepts are represented using **one-hot encoding**.[39]

For instance, if the task is to predict the part-of-speech (POS) tag of a word given its context in a sentence (current and previous word identities $w$ and $pw$, as well as the latter's part-of-speech (POS) tag $pt$), the input vector could be obtained by concatenation of the one-hot encoding of $w$, the one-hot encoding of $pw$, and the one-hot encoding of $pt$.

The input vector that would be fed into a classifier to predict the POS of the word "house" in the sentence fragment "my house", say, given that "my" has been tagged as a 'determiner' could be:



The sparsity of this vector is a major CoD liability: a reasonably restrictive vocabulary subset of English might contain $|V_W| \approx 20,000$ words, while the Penn Treebank project recognizes $\approx 40$ POS tags, which means that $\mathbf{x} \in \mathbb{R}^{40,040}$ (at least).

Another issue is that the one-hot encoding of words does not allow for meaningful similarity comparisons between words: in NLP, words are considered to be similar if they appear in similar sentence **contexts**.[40]

The terms "black" and "white" are similar in this framework as they both often occur immediately preceding the same noun (such as "car", "dress", etc.); human beings recognize that the similarity goes further than both of the terms being adjectives – they are both colours, and are often used as opposite poles on a variety of spectra. This similarity is impossible to detect from the one-hot encoding vectors, however, as all its word representations are exactly as far from one another.

[39] Sparse vectors whose entries are 0 or 1, based on the identity of the words and POS tags under consideration.

[40] "Ye shall know a word by the company it keeps", as the **distributional semantics** saying goes. The term "kumipwam" is not found in any English dictionary, but its probable meaning as "a small beach/sand lizard" could be inferred from its presence in sentences such as "Elowyn saw a tiny scaly kumipwam digging a hole on the beach". It is easy to come up with examples where the context is ambiguous, but on the whole the contextual approach has proven itself to be mostly reliable.

SVD proposes a single resolution to both of these issues. Let $\mathbf{M}^f \in \mathbb{R}^{|V_W|\times|V_C|}$ be the **word-context** matrix of the association measure $f$, derived from some appropriate corpus, that is, if $V_W = \{w_1, \dots, w_{|V_W|}\}$ and $V_C = \{c_1, \dots, c_{|V_C|}\}$ are the vocabulary and contexts of the corpus, respectively, then $M^f_{i,j} = f(w_i, c_j)$ for all $i, j$.

For instance, one could have

$$\begin{aligned} V_W &= \{\text{aardvark}, \dots, \text{zygote}\}\,, \\ V_C &= \{\dots, \text{word appearing before "cat"}, \dots\}, \end{aligned}$$

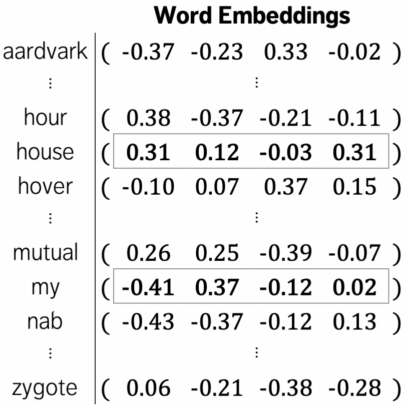
and $f$ given by **positive pointwise mutual information** for words and contexts in the corpus (the specifics of which are not germane to the discussion at hand; see [305] for details).

The SVD

$$\mathbf{M}^f \approx \mathbf{M}^f_d = \mathbf{U}_d \mathbf{\Sigma}_d \mathbf{V}^\top_d$$

yields $d-$dimensional word embedding vectors $\mathbf{U}_d\mathbf{\Sigma}_d$ which preserve the **context-similarity** property discussed above. The decomposition of the POS-context matrix, where words are replaced by POS tags, produces POS embedding vectors.

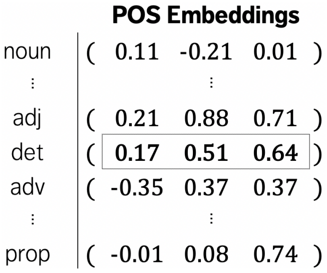
Perhaps a pre-calculated 4-dimensional word embedding of $V_W$ is:



**Word Embeddings**

| | |
|---|---|
| aardvark | ( -0.37 -0.23 0.33 -0.02 ) |
| ⋮ | ⋮ |
| hour | ( 0.38 -0.37 -0.21 -0.11 ) |
| house | ( **0.31 0.12 -0.03 0.31** ) |
| hover | ( -0.10 0.07 0.37 0.15 ) |
| ⋮ | ⋮ |
| mutual | ( 0.26 0.25 -0.39 -0.07 ) |
| my | ( **-0.41 0.37 -0.12 0.02** ) |
| nab | ( -0.43 -0.37 -0.12 0.13 ) |
| ⋮ | ⋮ |
| zygote | ( 0.06 -0.21 -0.38 -0.28 ) |

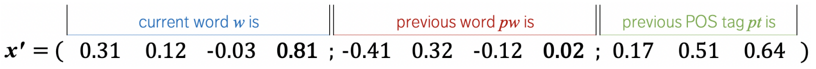
while a 3-dimensional POS embeddings could be:



**POS Embeddings**

| | |
|---|---|
| noun | ( 0.11 -0.21 0.01 ) |
| ⋮ | ⋮ |
| adj | ( 0.21 0.88 0.71 ) |
| det | ( **0.17 0.51 0.64** ) |
| adv | ( -0.35 0.37 0.37 ) |
| ⋮ | ⋮ |
| prop | ( -0.01 0.08 0.74 ) |

leading to a 11-dimensional representation $\mathbf{x}'$ of $\mathbf{x}$



$$\boldsymbol{x}' = (\overbrace{0.31 \quad 0.12 \quad -0.03 \quad \mathbf{0.81}}^{\text{current word } w \text{ is}} \;;\; \overbrace{-0.41 \quad 0.32 \quad -0.12 \quad \mathbf{0.02}}^{\text{previous word } pw \text{ is}} \;;\; \overbrace{0.17 \quad 0.51 \quad 0.64}^{\text{previous POS tag } pt \text{ is}})$$

which provides a **substantial reduction** in the dimension of the input space.

### 23.4.2 PCA Regression and Partial Least Squares

For $m = 1, \dots, M \leq p$, we let $z_m = \vec{X}^\top \boldsymbol{\phi}_m$ be linear combinations of the original predictors $\{X_1, \dots, X_p\}$.

If we are fitting $y = f(\mathbf{x}) = \mathrm{E}[Y \mid \vec{X} = \mathbf{x}]$ using OLS, we can also fit $y_i = \theta_0 + \mathbf{z}_i^\top \boldsymbol{\theta} + \varepsilon_i$, $i = 1, \dots, n$ using OLS. If the constants $\phi_{m,j}$ are selected **wisely**, then transforming the variables can yield a model that outperforms OLS regression – the predictions might be better than those obtained by fitting $y_i = \beta_0 + \mathbf{x}_i^\top \boldsymbol{\beta} + \varepsilon_i$, $i = 1, \dots, N$.

By definition, $\theta_0 = \beta_0$ and

$$\begin{aligned}
\mathbf{z}_i^\top \boldsymbol{\theta} &= \sum_{m=1}^{M} \theta_m z_{i,m} = \sum_{m=1}^{M} \theta_m \mathbf{x}_i^\top \boldsymbol{\phi} = \sum_{m=1}^{M} \theta_m \sum_{j=1}^{p} \phi_{m,j} x_{i,j} \\
&= \sum_{j=1}^{p} \sum_{m=1}^{M} \theta_m \phi_{m,j} x_{i,j} = \sum_{j=1}^{p} \beta_j x_{i,j} = \mathbf{x}_i^\top \boldsymbol{\beta},
\end{aligned}$$

where $\beta_j = \sum_{m=1}^{M} \theta_m \phi_{m,j}$, which is to say that the dimension reduction regression is a special case of the original linear regression model, with constrained coefficients $\beta_j$.

Such constraints can help with the bias-variance trade-off (when $p \gg n$, picking $M \ll p$ can reduce the variance of the fitted coefficients).

The challenge then is to find an appropriate way to pick the $\phi_{m,j}$. We will consider two approaches: **principal components** and **partial least squares**.

**Principal Components Regression** Let us assume that $M$ **principal components** $\{Z_1, \dots, Z_M\}$ have been retained (see Section 23.2.3), where

$$Z_i = \mathbf{w}_i^\top (X_1, \dots, X_p),$$

assuming that the eigenvectors $\mathbf{w}_i$ are ordered according to the corresponding eigenvalues ($\lambda_1 \geq \lambda_2 \geq \cdots \geq \lambda_p \geq 0$):

- the first principal component is the **normalized** (centered and scaled) linear combination of variables with the largest variance;
- the second principal component is the normalized linear combination of variables with the largest variance, subject to having no correlation with all previous components (the first);
- ...
- the $M$th principal component is the normalized linear combination of variables with the largest variance, subject to having no correlation with all previous components.

The regression function $f(\mathbf{x}) = \mathrm{E}[Y \mid \vec{X} = \mathbf{x}]$ is hopefully well approximated by the function $g(\mathbf{z}) = \mathrm{E}[Y \mid \vec{Z} = \mathbf{z}]$, i.e.,

$$\hat{y}_z = g(\mathbf{z}) = \gamma_0 + \gamma_1 z_1 + \cdots + \gamma_M z_M$$

should compare "reasonably well" to

$$\hat{y}_z = f(\mathbf{z}) = \beta_0 + \beta_1 x_1 + \cdots + \beta_p x_p.$$

The main challenge is to determine the optimal $M$. If $M$ is too small, we run the risk of having a model with high squared bias and low variance (**underfitting**); if $M$ is too large, not only we we not achieve much in the way of dimension reduction, but we might produce a model with low squared bias and high variance (**overfitting**).

Any method that allows for the estimation of $\text{MSE}_{\text{Te}}$ (such as cross-validation) could be used to select $M$, but there are other approaches as well (again, see Section 23.2.3).

**Partial Least Squares** In **principal component regression** (PCR), the identified **directions** (linear combinations) that best represent the predictors $\{X_1, \ldots, X_p\}$ are determined in an **unsupervised** manner: the response $Y$ plays no role in determining the principal components.

As such, there is no guarantee that the directions that best explain the predictors are also the best directions to use to predict the response. The framework for **partial least squares** is the same as that for PCR, except that the directions $Z_i$ are selected both to explain the predictors and to be related to the response $Y$.

As in PCR, we start by **normalizing** (centering and scaling) the predictor part of the training set Tr. The **first direction** $Z_1$ is computed using the OLS coefficient estimates of

$$Y_i = \phi^1_{0,j} + \phi_{1,j} X_{i,j} + \gamma_i, \quad j = 1, \ldots, p, i = 1, \ldots, N.$$

Note that each $\phi_{1,j}$ is proportional to $\rho_{X_j,Y}$ and that the direction

$$Z_1 = \sum_{j=1}^{p} \phi_{1,j} X_j = \boldsymbol{\phi}_1^\top \vec{X}$$

places the highest weights on the predictors that are most strongly linked to the response. Now, we run an OLS regression of $Y$ using $Z_1$ as a predictor:

$$Y_i = \psi_0 + \psi_1 z_{1,i} + \varepsilon_i, \quad i = 1, \ldots, N$$

and let $\varepsilon_i = Y_i - \psi_0 - \psi_1 z_{1,i}$ be the component of the data not "explained" by $Z_1$.

The **second direction** $Z_2$ is computed using the OLS coefficient estimates of

$$\varepsilon_i = \phi^2_{0,j} + \phi_{2,j} X_{i,j} + \gamma_i, \quad j = 1, \ldots, p, i = 1, \ldots, N.$$

Note that each $\phi_{2,j}$ is proportional to $\rho_{X_j,\varepsilon}$ and that the direction

$$Z_2 = \sum_{j=1}^{p} \phi_{2,j} X_j = \boldsymbol{\phi}_2^\top \vec{X}$$

places higher weights on the predictors that are most strongly linked to the first residual (which is to say, the component that does not explain $Z_1$). The process continues in the same way, building directions $Z_3, \ldots, Z_p$ that are strongly linked, in sequence, to the preceding residuals; as the chain starts with the response $Y$, the directions do take into account both the related response and the predictor structure.[41]

[41] The problem of selecting $M$ is tackled as it is in PCA regression.

**Summary** Due to the **bias-variance trade-off** (see Chapters 19 and 21), we must often strike the right balance in terms of model complexity, which is usually measured in terms of the number of parameters that must be estimated from Tr.

While this allows us to compare completely different models with one another, it also suggests that models that use fewer predictors as inputs are not as complex as those that use the full set of predictors. The full models are not necessarily the ones that perform best (in term of **Te error**), thanks to the **curse of dimensionality**.

Thankfully, predictor subset selection methods can be used to select the best model: while the cross-validation approach is strongly encouraged, other approaches (including shrinkage, feature selection, dimension reduction) could also prove competitive.

### 23.4.3 Spectral Feature Selection

Text mining tasks often give rise to datasets which are likely to be affected by the CoD; the problem also occurs when dealing with-high resolution images, with each of the millions of pixels it contains viewed as a feature.[42]

**Spectral feature selection** attempts to identify "good" or "useful" training features in such datasets by measuring their relevance to a given task *via* spectral analysis of the data.

**General Formulation for Feature Selection** Let $\mathbf{X} \in \mathbb{R}^{n\times p}$ be a data set with $p$ features and $n$ observations. The problem of $\ell-$feature selection, with $1 \leq \ell \leq p$, can be formulated as an optimization problem [306]:

$$\begin{aligned} &\max_{\mathbf{W}} r(\hat{\mathbf{X}}) \\ \text{s.t. } &\hat{\mathbf{X}} = \mathbf{XW}, \quad \mathbf{W} \in \{0,1\}^{p\times\ell} \\ &\mathbf{W}^{\top}\mathbf{1}_{p\times 1} = \mathbf{1}_{\ell\times 1}, \quad \|\mathbf{W}\mathbf{1}_{\ell\times 1}\|_0 = \ell \end{aligned}$$

The **score function** $r(\cdot)$ is the objective which evaluates the relevance of the features in $\hat{\mathbf{X}}$, the data set containing only the features selected by the **selection matrix W** with entries either 0 or 1. To ensure that only the original feature are selected (and not a linear combination of features), the problem stipulates that each column of **W** contains only one 1 ($\mathbf{W}^{\top}\mathbf{1}_{p\times 1} = \mathbf{1}_{\ell\times 1}$); to ensure that exactly $\ell$ rows contain one 1, the constraint $\|\mathbf{W}\mathbf{1}_{\ell\times 1}\|_0 = l$ is added.

The selected features are often represented by

$$\hat{\mathbf{X}} = \mathbf{XW} = (f_{i_1}, \ldots, f_{i_\ell}) \quad \text{with } \{i_1, \ldots, i_\ell\} \subset \{1, \ldots, p\}.$$

If $r(\cdot)$ does not evaluate features independently, this optimization problem is NP-hard. To make to problem easier to solve, the features are assumed to be **independent** of one another.[43] In that case, the objective function reduces to

$$\max_{\mathbf{W}} r(\hat{\mathbf{X}}) = \max_{\mathbf{W}} \left( r(f_{i_1}) + \cdots + r(f_{i_l}) \right);$$

[42] Such images contain millions of pixels, if not more.

[43] Or that their interactions are negligible.

the optimization problem can then be solved by selecting the $\ell$ features with the largest individual scores. The link with **spectral analysis** will now be explored.[44]

**Similarity Matrix** Let $s_{i,j}$ denote the **pairwise similarity** between observations $\mathbf{x}_i$ and $\mathbf{x}_j$. If class labels $y(\mathbf{x}) \in \{1, \dots, K\}$ are known for all instances $\mathbf{x}$, the following function can be used

$$s_{i,j} = \begin{cases} \frac{1}{n_k}, & y(\mathbf{x}_i) = y(\mathbf{x}_j) = k \\ 0, & \text{otherwise} \end{cases}$$

where $n_k$ is the number of observations with class label $k$.

If class labels are not available, a popular similarity function is the **Gaussian radial basis function** (RBF) kernel, given by

$$s_{i,j} = \exp\left(-\frac{\|x_i - x_j\|^2}{2\sigma^2}\right),$$

where $\sigma$ is a parameter that is used to control the Gaussian's width.[45] For a given $s_{i,j}$ and $n$ observations, the **similarity matrix** $S$ is an $n \times n$ matrix containing the observations' pairwise similarities, $S(i, j) = s_{i,j}$, $i, j = 1, \dots, n$.

By convention, $\text{diag}(S) = \mathbf{0}$. Other similarity functions include the following kernels [307]:

1. **linear** – $s_{i,j} = \mathbf{x}_i^\top \mathbf{x}_j + c$, $c \in \mathbb{R}$;
2. **polynomial** – $s_{i,j} = (\alpha \mathbf{x}_i^\top \mathbf{x}_j + c)^d$, $\alpha, c \in \mathbb{R}$,[46] $d \in \mathbb{N}$, and
3. **cosine** – $s_{i,j} = \frac{\mathbf{x}_j^\top \mathbf{x}_i}{\|\mathbf{x}_i\|\|\mathbf{x}_j\|}$, which measures the similarity of 2 vectors by determining the angle between them.[47]

**Similarity Graph** For each similarity matrix $S$, one can construct a **weighted graph** $G(V, E)$ in which each observation corresponds to a node and the associated pairwise similarities correspond to the respective edge weights; $V$ is the set of all **vertices** (nodes) and $E$ the set of all graph **edges**. As an example, consider the simple dataset:

$$\mathbf{X} = \begin{bmatrix} 1 & 0 & 3 & 6 & 7 & 6 \\ 1 & 2 & 2 & 4 & 4 & 8 \end{bmatrix}^\top;$$

for validation purposes, we will assume that the first three belong to one group, the last three, to another. The scatter plot is obtained below:

```
from matplotlib import ticker, cm
import numpy as np
import matplotlib.pyplot as plt
from scipy.spatial.distance import pdist, squareform, cdist
import networkx as nx
from numpy.linalg import eigh,norm
import random # for replicability
```

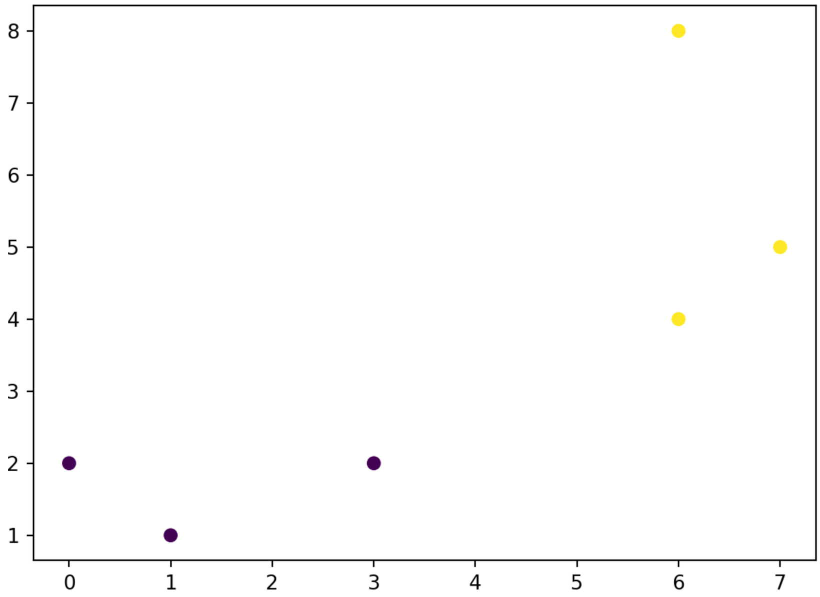
[44] We have encountered some of these concepts in Section 22.4.2.

[45] One can think of this as the "reach" of each point.

[46] For image processing, this kernel is often used with $\alpha = c = 1$.

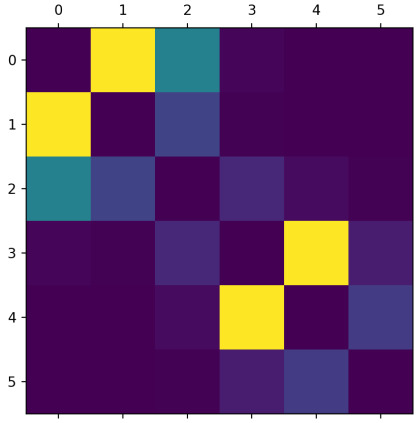
[47] It is often used in high-dimensional applications such as text mining.

```
# Data
X = np.array([[1,1,1],[0,2,1],[3,2,1],[6,4,1],[7,5,1],[6,8,1]])
Y = np.array([0,0,0,1,1,1])
# Plot
plt.scatter(X[:,0],X[:,1], c=Y)
```



Next, we compute the similarity (adjacency matrix). We use the Gaussian RBF kernel with $\sigma = 0.5$, and `pdist()` from `scipy.spatial.distance` which computes the pairwise distance of each row from an input data matrix. `pdist()` return a vector, which allows a speed boost for the following computation, but it needs to be converted back to a matrix for the next steps.[48]

```
S = squareform(np.exp(-pdist(X))/((0.5)**2))
plt.matshow(S)
plt.show()
```



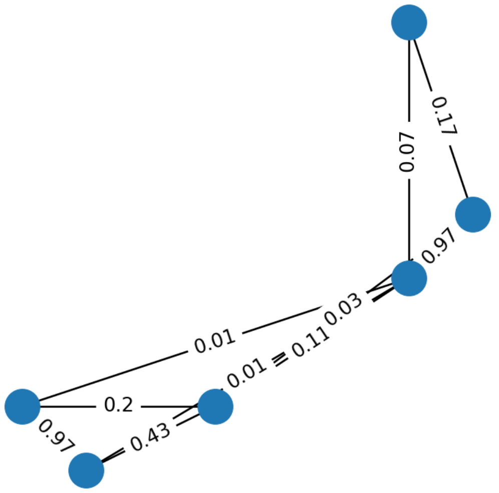
[48] This is done *via* `squareform()`, also from `scipy.spatial.distance`.

or

$$S = \begin{bmatrix} 0 & 0.972 & 0.428 & 0.012 & 0.003 & 0.001 \\ 0.972 & 0 & 0.199 & 0.007 & 0.002 & 0.001 \\ 0.428 & 0.199 & 0 & 0.109 & 0.027 & 0.005 \\ 0.012 & 0.007 & 0.109 & 0 & 0.972 & 0.073 \\ 0.003 & 0.002 & 0.027 & 0.972 & 0 & 0.169 \\ 0.001 & 0.001 & 0.005 & 0.073 & 0.169 & 0 \end{bmatrix},$$

and the resulting graph $G$ is shown below:

```
G = nx.from_numpy_matrix(S)

# Add the weight (similarity) attribute to the graph
pos = {i : [X[i,0],5*X[i,1]] for i in range(6)}
labels = nx.get_edge_attributes(G,'weight')

# round the similarity (for display)
labels = {k: round(v,2) for k, v in labels.items()}

# Create the edge list and labels. Remove null edges
edge_list = [k for k, v in labels.items() if v != 0]
labels = {k: v for k, v in labels.items() if v != 0}

# Draw it using the edge and labels list, at the right position
nx.draw_networkx_nodes(G,pos)
nx.draw_networkx_edges(G,pos, edgelist=edge_list)
nx.draw_networkx_edge_labels(G,pos, edge_labels=labels)
plt.axis('off')
plt.show()
```



**Laplacian Matrix of a Graph** The similarity matrix $S$ is also known as the **adjacency matrix** $A$ of the graph $G$,[49] from which the **degree matrix** $D$ can be constructed:

$$D(i,j) = \begin{cases} d_{i,i} = \sum_{k=1}^{n} a_{i,k}, & i = j \\ 0, & \text{otherwise} \end{cases}$$

[49] In certain formulations, the entries of the adjacency matrix $A$ are instead defined to take on the value 1 or 0, depending as to whether the similiarity between the corresponding observations is greater than (or smaller than) some pre-determined threshold $\tau$.

By definition, $D$ is diagonal; the element $d_{i,i}$ can be viewed as an estimate of the **density** around $\mathbf{x}_i$; as $a_{i,k}(= s_{i,k)}$ is a measure of similarity between $\mathbf{x}_i$ and $\mathbf{x}_k$, the larger $a_{i,k}$ is, the more similar the two observations are.

A large value of $d_{i,i}$ indicates the presence of one or more observations "near" $\mathbf{x}_i$; conversely, a small value of $d_{i,i}$ suggests that $\mathbf{x}_i$ is isolated.

The **Laplacian** and **normalized Laplacian** matrices are defined as

$$L = D - A \quad \text{and} \quad \mathscr{L} = D^{-1/2}LD^{-1/2},$$

respectively. Since $D$ is diagonal, $D^{-1/2} = \operatorname{diag}\left(d_{i,i}^{-1/2}\right)$.

It can be shown that $L$ and $\mathscr{L}$ are both positive semi-definite matrices. By construction, the smallest eigenvalue of $L$ is 0, with associated eigenvector $\mathbf{1}$, since

$$L\mathbf{1} = (D - A)\mathbf{1} = D\mathbf{1} - A\mathbf{1} \tag{23.1}$$

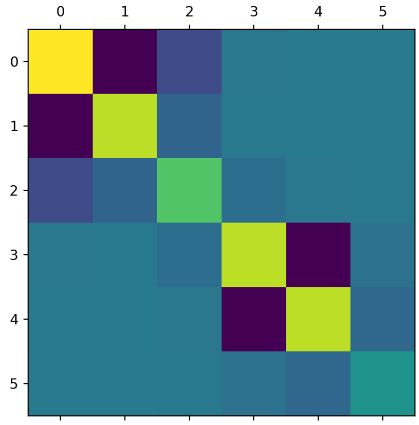
$$= \begin{pmatrix} d_{1,1} \\ \vdots \\ d_{n,n} \end{pmatrix} - \begin{pmatrix} \sum_{k=1}^{n} a_{1,k} \\ \vdots \\ \sum_{k=1}^{n} a_{n,k} \end{pmatrix} = \begin{pmatrix} d_{1,1} - \sum_{k=1}^{n} a_{1,k} \\ \vdots \\ d_{n,n} - \sum_{k=1}^{n} a_{n,k} \end{pmatrix} = \begin{pmatrix} 0 \\ \vdots \\ 0 \end{pmatrix} = 0 \cdot \mathbf{1}. \tag{23.2}$$

For $\mathscr{L}$, the corresponding eigenpair is 0 and $\operatorname{diag}(D^{1/2})$ (the proof is similar).

We can compute the degree matrix $D$ and the Laplacian $L$ for the toy example above. For the degree matrix, we use the method `sum()` from `numpy` with the argument `axis=1` to sum over the columns and `diag()` to convert the result into a diagonal matrix.

The Laplacian is simply $L = D - S$.

```
rowsum = S.sum(axis=1)
D = np.diag(rowsum)
L = D - S
plt.matshow(L)
plt.show()
```



**Eigenvectors as Cluster Separators** The eigenvectors of the Laplacian matrices have some very useful properties relating to features selection. If $\boldsymbol{\xi} \in \mathbb{R}^n$ is an eigenvector of $L$ or $\mathscr{L}$, then $\boldsymbol{\xi}$ can be viewed as a function that assigns a value to each observation in $\mathbf{X}$.

This point-of-view can prove quite useful, as the following simple example from [306] shows. Let $\mathbf{X}$ be constructed of three two-dimensional Gaussians, each with unit variance (and no covariance) but with different means. We start by generating 30 observations for each of the 3 mechanisms, and re-shuffle them into a "random" dataset.

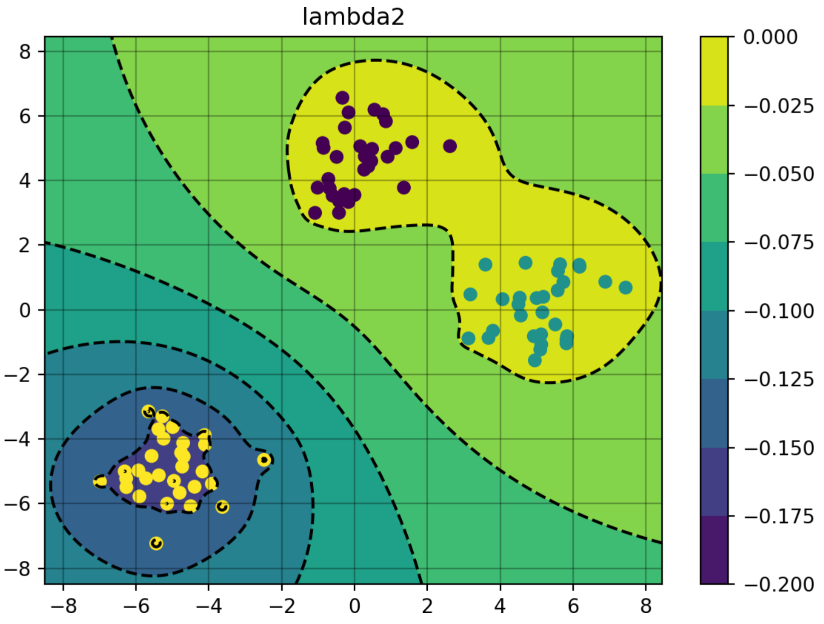
```
random.seed(1234) # for replicability

mean1 = [0,5]
cov1 = [[1,0],[0,1]]
X1 = np.random.multivariate_normal(mean1,cov1,30)
mean2 = [5,0]
cov2 = [[1,0],[0,1]]
X2 = np.random.multivariate_normal(mean2,cov2,30)
mean3 = [-5,-5]
cov3 = [[1,0],[0,1]]
X3 = np.random.multivariate_normal(mean3,cov3,30)

plt.scatter(X1[:,0],X1[:,1])
plt.scatter(X2[:,0],X2[:,1])
plt.scatter(X3[:,0],X3[:,1])

X = np.concatenate((X1,X2,X3), axis=0)
Y = np.array([0] * 30 + [1] * 30 + [2] * 30).reshape((90,1))

df = np.concatenate((X,Y), axis = 1)
np.random.shuffle(df)
X,Y = df[:,:2],df[:,2]
```

We can then compute the similarity (using Guassian RBF with $\sigma = 1$), adjacency, degree and Laplacian matrix. Using `eigh()` from `numpy` we can find the eigenvalue and eigenvectors of $L$. This function returns a vector of all the eigenvalues of $L$ and a matrix of all its eigenvectors as columns. According to the convention, the eigenspace is sorted so that the eigenvalues satisfy $0 = \lambda_1 \leq \lambda_2 \leq \lambda_3 \leq \ldots$

```
.exp(-1 * pdist(X, 'sqeuclidean')))
is=1)
m)


 of L (vs: eigenvalue vector, es: eigenvector matrix)


alue
sort()

t]
```

We show properties of $L$'s spectrum by providing the **contour plot** of the second eigenvector/eigenvalue pair.[50]

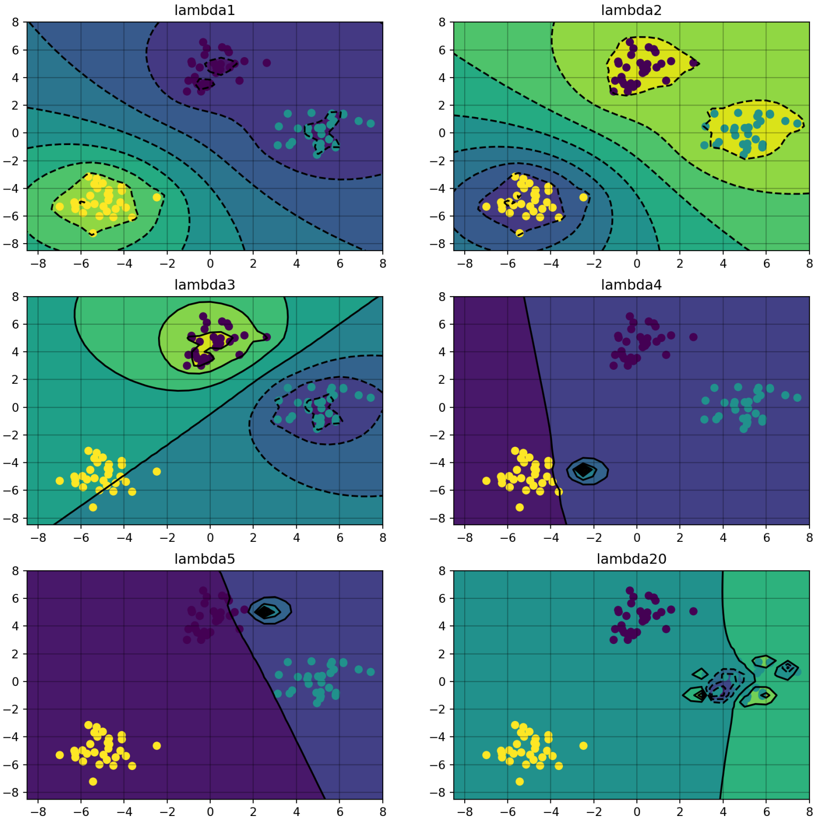
```
def gen_z(lam = 1):
  """ Compute the meshgrid for z
  """
  for i in range(len(x)):
    for j in range(len(y)):
      dist = cdist([[x_[i,j],y_[i,j]]],X)[0]
      z[i,j] = np.average(es[:,lam], weights= 1/dist)

# Setup the meshgrid
x = np.arange(-8.5,8.5,0.05)
y =  np.arange(-8.5,8.5,0.05)
x_, y_ = np.meshgrid(x,y, indexing='ij')
z = 0*x_

# Index of the eigenvalue to be plotted (0 is the first, etc)
l = 1

# Compute the meshgrid with the gen_z function
gen_z(l)

# Setup and plot
fig,ax = plt.subplots()
cs = ax.contourf(x_, y_, z)
ax.contour(cs,colors='k')
ax.grid(c='k', ls='-', alpha=0.3)
fig.colorbar(cs)
ax.scatter(X[:,0],X[:,1], c=Y)
ax.set_title('lambda{0}'.format(l+1))
plt.show()
```



[50] Remember, the eigenvectors act as functions in this viewpoint. For a given eigenvector $\lambda_j$, the contour value at each point $\mathbf{x}_i$ is the value of the associated eigenvector $\boldsymbol{\xi}_j$ in the $i^{\text{th}}$ position, namely $\xi_{j,i}$. For any point $\mathbf{x}$ not in the dataset, the contour value is given by averaging the $\xi_{j,k}$ of the observations $\mathbf{x}_k$ near $\mathbf{x}$, inversely weighted by the distances $\|\mathbf{x}_k - \mathbf{x}\|$.

The next block accomplishes the same tasks, but it does so for potentially more than one eigenvector/eigenvalue pair, and , and aranges the plots in a grid. The contour plot of the ranked eigenvectors $\xi_1, \xi_2, \xi_3, \xi_4, \xi_5$ and $\xi_{20}$, corresponding to the eigenvalues $\lambda_1 \leq \lambda_2 \leq \lambda_3 \leq \lambda_4 \leq \lambda_5 \leq \lambda_{20}$ are computed and displayed below.

```
x = np.arange(-8.5,8.5,0.5)
y =  np.arange(-8.5,8.5,0.5)
x_, y_ = np.meshgrid(x,y, indexing=’ij’)
z = 0*x_

# List of index of the eig to be plotted
ls = [0,1,2,3,4,19]
grid_length = 3
grid_width = 2

plt.figure(figsize=(12,12))
for i in range(len(ls)):
  ax = plt.subplot(grid_length,grid_width,i+1)
  z = 0*x_
  gen_z(ls[i])
  cs = ax.contourf(x_, y_, z)
  ax.contour(cs,colors=’k’)
  ax.grid(c=’k’, ls=’-’, alpha=0.3)
  ax.scatter(X[:,0],X[:,1], c=Y)
  ax.set_title(’lambda{0}’.format(ls[i]+1))
```



From those plots, it seems as though the first eigenvector does a better job at capturing the cluster structure in the data, while larger values tends to capture more of the sub-cluster structure. One thing to note is that it might appear that $\lambda_1$ is as good (or better) as $\lambda_2$ and $\lambda_3$ to separate the groups, but a closer look at the scale of the contour plot of $\lambda_1$ shows that its values have a miniscule range.

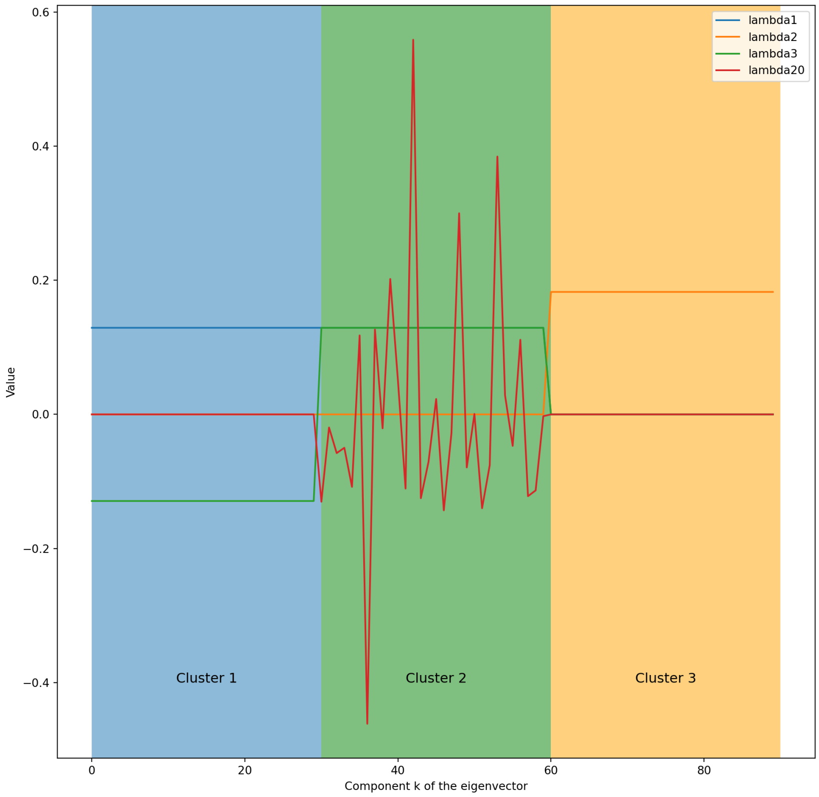
The fact that there is any variation at all is due to floating point errors in the practical computation of the eigenvalue $\lambda_1$ and the eigenvector $\xi_1$; as seen previously, these should be exactly 0 and $\mathbf{1}$, respectively.

At any rate, this process shows how the eigenpairs of the Laplacian matrix contains information about the structure of $\mathbf{X}$.

In spectral graph theory, the eigenvalues of the Laplacian measure the **smoothness** of the eigenvectors. An eigenvector is said to be **smooth** if it assigns similar values to points that are near one another.

In the previous example, assume that the data is **unshuffled**, that is, the first $k_1$ points are in the same cluster, the next $k_2$ are also in the same, albeit different, cluster, and so on. The next plot shows the smoothness of the eigenvector over each cluster; colour is added to emphasize the cluster limits.

```
# Recompute everything without the shuffling and sorting part
X = np.concatenate((X1,X2,X3), axis=0)
Y = np.array([0] * 30 + [1] * 30 + [2] * 30).reshape((90,1))

A = squareform(np.exp(-1 * pdist(X, 'sqeuclidean')))
rowsum = A.sum(axis=1)
D = np.diag(rowsum)
L = D - A
vs,es = eigh(L)

# List of index of selected eigenvalue.
ls = [0,1,2,19]

# Plot a line for each index in ls
for l in ls:
  plt.plot(es[:,l], label = 'lambda{0}'.format(l+1))

# Color the cluster limits
plt.axvspan(0,30, alpha = 0.5)
plt.text(15,-0.4, 'Cluster 1', fontsize=12)
plt.axvspan(30,60, alpha = 0.5, facecolor='g')
plt.text(45,-0.4, 'Cluster 2', fontsize=12)
plt.axvspan(60,90, alpha = 0.5, facecolor='orange')
plt.text(75,-0.4, 'Cluster 3', fontsize=12)

plt.legend()
plt.ylabel('Value')
plt.xlabel('Component k of the eigenvector')
plt.show()
```



Both $\lambda_2$ and $\lambda_3$ are fairly smooth, as they seem to be piece-wise constant on each cluster, whereas $\lambda_{20}$ is all over the place on cluster 1 and constant on the rest of the data. As discussed previously $\lambda_1$ is constant over the entirety of the dataset, marking it as maximally smooth but not very useful from the perspective of differentiating data structure.[51]

Indeed, let $\mathbf{x} \in \mathbb{R}^n$. then

$$\begin{aligned}\mathbf{x}^\top L\mathbf{x} = \mathbf{x}^\top D\mathbf{x} - \mathbf{x}^\top A\mathbf{x} &= \sum_{i=1}^{n} d_i x_i^2 - \sum_{i,j=1}^{n} a_{i,j} x_i x_j \\ &= \frac{1}{2}\left(\sum_{i=1}^{n} d_i x_i^2 - 2\sum_{i,j=1}^{n} a_{ij} x_i x_j + \sum_{j=1}^{n} d_j x_j^2\right) \\ &= \frac{1}{2}\sum_{i,j=1}^{n} a_{ij}(x_i - x_j)^2\end{aligned}$$

If $\mathbf{x} = \boldsymbol{\xi}$ is a normalized eigenvector of $L$, then $\boldsymbol{\xi}^\top L\boldsymbol{\xi} = \lambda\boldsymbol{\xi}^\top\boldsymbol{\xi} = \lambda$, thus

$$\lambda = \boldsymbol{\xi}^\top L\boldsymbol{\xi} = \frac{1}{2}\sum_{i,j=1}^{n} a_{ij}(\xi_i - \xi_j)^2.$$

Instinctively, if the eigenvector component does not vary a lot for observations that are near one another, one would expect the corresponding eigenvalue to be small; this result illustrates why the small magnitude of the eigenvalue is a good measure of the smoothness of its associated eigenvector.

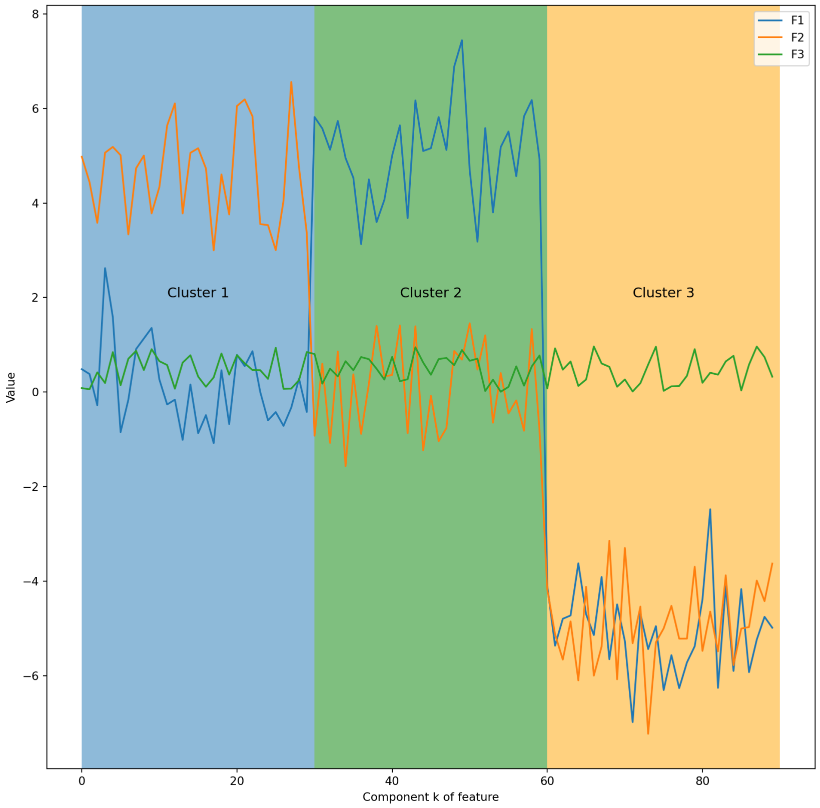
[51] As a reminder, the eigenvalues themselves are ordered in increasing sequence: for the current example,

$$\lambda_1 = 0 \le \lambda_2 = 1.30 \times 10^{-2} \le \lambda_3 = 3.94 \times 10^{-2} \le \cdots \lambda_{20} = 2.95 \le \cdots$$

**Feature Ranking** We can use the above discussion as a basis for feature selection. If $\mathbf{x}$ is not en eigenvector of $L$, the value $\mathbf{x}^\top L\mathbf{x}$ can also be seen as a measure of how much $\mathbf{x}$ varies locally. This can be used to measure how meaningful a feature $\mathbf{f} \in \mathbb{R}^n$ is.

In the current example, the only two features are the Euclidean coordinates of the observations: $\mathbf{f}_1$ and $\mathbf{f}_2$. We also add a **useless** feature $\mathbf{f}_3$ to the dataset (distributed uniformly across the clusters).

```
# Add a random feature
U1 = np.random.uniform(size=(90,1))
X = np.concatenate((X,U1), axis=1)
f = [0,1,2]

for i in f:
  plt.plot(X[:,i], label = 'F{0}'.format(i+1))

plt.axvspan(0,30, alpha = 0.5)
plt.text(15,2, 'Cluster 1', fontsize=12)
plt.axvspan(30,60, alpha = 0.5, facecolor='g')
plt.text(45,2, 'Cluster 2', fontsize=12)
plt.axvspan(60,90, alpha = 0.5, facecolor='orange')
plt.text(75,2, 'Cluster 3', fontsize=12)

plt.legend()
plt.ylabel('Value')
plt.xlabel('Component k of feature')
plt.show()
```



The plot shows that the third feature is not able to distinguish between the clusters. However, we also have:

```
for i in [0,1,2]:
  f = X[:,i]
  print("F{0}".format(i+1), "->",f.T.dot(f.dot(L)))
```

```
F1 -> 105.35092482283625
F2 -> 110.6011478717457
F3 -> 46.90787692252817
```

Thus

$$\mathbf{f}_1^\top L\mathbf{f}_1 = 94.3, \quad \mathbf{f}_2^\top L\mathbf{f}_2 = 102.5, \quad \mathbf{f}_3^\top L\mathbf{f}_3 = 41.7;$$

by the previous assumption relating the magnitude of $\xi^\top L\xi$ to the smoothness of $\xi$, this would seem to indicate that $\mathbf{f}_3$ is a "better" feature than the other two.

The problem is that the value of $\mathbf{f}_i^\top L\mathbf{f}_i$ is affected by the respective norms of $\mathbf{f}_i$ and $L$. This need to be addressed.

The relation between $L$ and $\mathscr{L}$ yields

$$\mathbf{f}_i^\top L\mathbf{f}_i = \mathbf{f}_i^\top D^{1/2}\mathscr{L}D^{1/2}\mathbf{f}_i = (D^{1/2}\mathbf{f}_i)^T\mathscr{L}(D^{1/2}\mathbf{f}_i).$$

Set $\tilde{\mathbf{f}}_i = (D^{1/2}\mathbf{f}_i)$ and $\hat{\mathbf{f}}_i = \tilde{\mathbf{f}}_i/\|\tilde{\mathbf{f}}_i\|$. The **feature score metric** $\varphi_1$ is a normalized version of the smoothness measure:

$$\varphi_1(\mathbf{f}_i) = \hat{\mathbf{f}}_i^\top\mathscr{L}\hat{\mathbf{f}}_i, \quad i = 1, \ldots, p.$$

For $\varphi_1$, smaller values are better. The scoring function can also be defined using the spectral decomposition of $\mathscr{L}$.

Suppose that $(\lambda_k, \xi_k)$, $1 \leq k \leq n$ are **eigenpairs** of $\mathscr{L}$ and let $\alpha_k = \hat{\mathbf{f}}_i^\top\xi_k$, for a given $i$. Then

$$\varphi_1(\mathbf{f}_i) = \sum_{k=1}^{n}\alpha_k^2\lambda_k, \quad \text{with } \sum_{k=1}^{n}\alpha_k^2 = 1.$$

Indeed, let $\mathscr{L} = \mathbf{U}\mathbf{\Sigma}\mathbf{U}^\top$ be the eigen-decomposition of $\mathscr{L}$. By construction, $\mathbf{U} = [\xi_1|\xi_2|\cdots|\xi_n]$ and $\mathbf{\Sigma} = \text{diag}(\lambda_k)$, so that

$$\begin{aligned}\varphi_1(\mathbf{f}_i) &= \hat{\mathbf{f}}_i^\top\mathscr{L}\hat{\mathbf{f}}_i = \hat{\mathbf{f}}_i^\top\mathbf{U}\mathbf{\Sigma}\mathbf{U}^\top\hat{\mathbf{f}}_i \\ &= (\alpha_1, \ldots, \alpha_n)\mathbf{\Sigma}(\alpha_1, \ldots, \alpha_n)^\top = \sum_{k=1}^{n}\alpha_k^2\lambda_k.\end{aligned}$$

This representation allows for a better comprehension of the $\varphi_1$ score; $\alpha_k$ is the cosine of the angle between the normalized feature $\hat{\mathbf{f}}_i$ and eigenvector $\xi_k$. If a feature aligns with "good" eigenvectors (i.e., those with small eigenvalues), its $\varphi_1$ score will also be small.

The larger $\alpha_1^2$ is, the smaller $\sum_{k=2}^{n}\alpha_k^2$ is; this is problematic because, in such cases, a small value of $\varphi_1$ indicates smoothness but not separability.

To overcome this issue, $\varphi_1$ can be normalized by $\sum_{k=2}^{n} \alpha_k^2$, which yields a new scoring function:

$$\varphi_2(\mathbf{f}_i) = \frac{\sum_{k=1}^{n} \alpha_k^2 \lambda_k}{\sum_{k=2}^{n} \alpha_k^2} = \frac{\hat{\mathbf{f}}_i^\top \mathscr{L} \hat{\mathbf{f}}_i}{1 - \left(\hat{\mathbf{f}}_i^\top \xi_1\right)}$$

A small value for $\varphi_2$ once again indicates that a feature closely aligns with "good" eigenvectors.

Another ranking feature is closely related to the other two. According to spectral clustering, the first $k$ non-trivial eigenvectors form an optimal set of **soft cluster indicators** that separate the graph $G$ into $k$ connected parts. Therefore, we define $\varphi_3$ as

$$\varphi_3(\mathbf{f}_i, k) = \sum_{j=2}^{k} (2 - \lambda_j)\alpha_j^2.$$

Contrary to the other scoring functions, $\varphi_3$ assigns larger value to feature that are more relevant. It also prioritizes the leading eigenvectors, which helps to reduce noise. Using this ranking function requires a number of categories or clusters $k$ to be selected (depending on the nature of the ultimate task at hand); if this value is unknown, $k$ becomes a hyper-parameter to be tuned.

The feature score metrics are implemented as below:

```
D_sq = np.sqrt(D)
L_norm = D_sq.dot(L).dot(D_sq)

def score_1(i):
  f_tilde = D_sq.dot(X[:,i])
  f_hat = f_tilde / norm(f_tilde)
  return f_hat.dot(L_norm).dot(f_hat)

def score_2(i):
  f_tilde = D_sq.dot(X[:,i])
  f_hat = f_tilde / norm(f_tilde)
  phi_1 = f_hat.dot(L_norm).dot(f_hat)
  return phi_1 / (1 - (f_hat.dot(es[:,0])**2))

def score_3(i,k):
  f_tilde = D_sq.dot(X[:,i])
  f_hat = f_tilde / norm(f_tilde)
  alpha = f_hat.dot(es)
  temp = (2 - vs[1:k]) * (alpha[1:k])**2
  return np.sum(temp)

from tabulate import tabulate

n_feature = X.shape[1]
results = {’phi_1’:[], ’phi_2’:[], ’phi_3’: []}
k = 3
```

```
for i in range(n_feature):
  results[’phi_1’].append(score_1(i))
  results[’phi_2’].append(score_2(i))
  results[’phi_3’].append(score_3(i,k))

print(tabulate(results,
                      headers="keys",
                      showindex=True,
                      tablefmt="simple",
                      numalign="left"))
```

```
    phi_1    phi_2    phi_3
--  -------  -------  --------
0   2.5516   3.28365  1.37706
1   2.71268  3.47861  1.40086
2   17.0148  31.8369  0.433994
```

This make more sense, as the pattern is similar to the pattern obtained for the eigenvalues: $\mathbf{f}_1, \mathbf{f}_2$, being able to differentiate the clusters, have smaller $\varphi_1$ scores than $\mathbf{f}_3$. Returning to the current example, while the score of the useless feature 3, $\varphi_1(\mathbf{f}_3)$ is larger than the other scores, it is still small when compared to the eigenvalues of $L$. This is due to the fact the $\mathbf{f}_3$ and $\boldsymbol{\xi}_1$ are nearly co-linear.

Computing $\varphi_2$ for our three features yields a larger distinction between the real features and the random, useless one than with $\varphi_1$.

**Regularization** There is one glaring problem with the ranking functions that have been defined previously: they all assume the existence of a gap between subsets of "large" and "small" eigenvalues. For clearly separated data, that is to be expected; but in noisy data, this gap may be negligible, which leads to an increase in the score value of poor features [308].

This issue can be tackled by applying a **spectral matrix function** $\gamma(\cdot)$ to the Laplacian $\mathscr{L}$, replacing the original eigenvalues by **regularized eigenvalues** as follows:

$$\gamma(\mathscr{L}) = \sum_{j=1}^{n} \gamma(\lambda_j)\xi_j\xi_j^{\top}.$$

In order for this to work properly, $\gamma$ needs to be (strictly) increasing. Examples of such regularization functions include:

| $\gamma(\lambda)$ | **(name)** |
|---|---|
| $1 + \sigma^2\lambda$ | (regularized Laplacian) |
| $\exp(\sigma^2/2\lambda)$ | (diffusion Process) |
| $\lambda^{\nu},\ \nu \geq 2$ | (high-order polynomial) |
| $(a - \lambda)^{-p},\ a \geq 2$ | ($p$-step random walk) |
| $(\cos \lambda\pi/4)^{-1}$ | (inverse cosine) |

The ranking function $\varphi_1, \varphi_2, \varphi_3$ can be regularized *via*

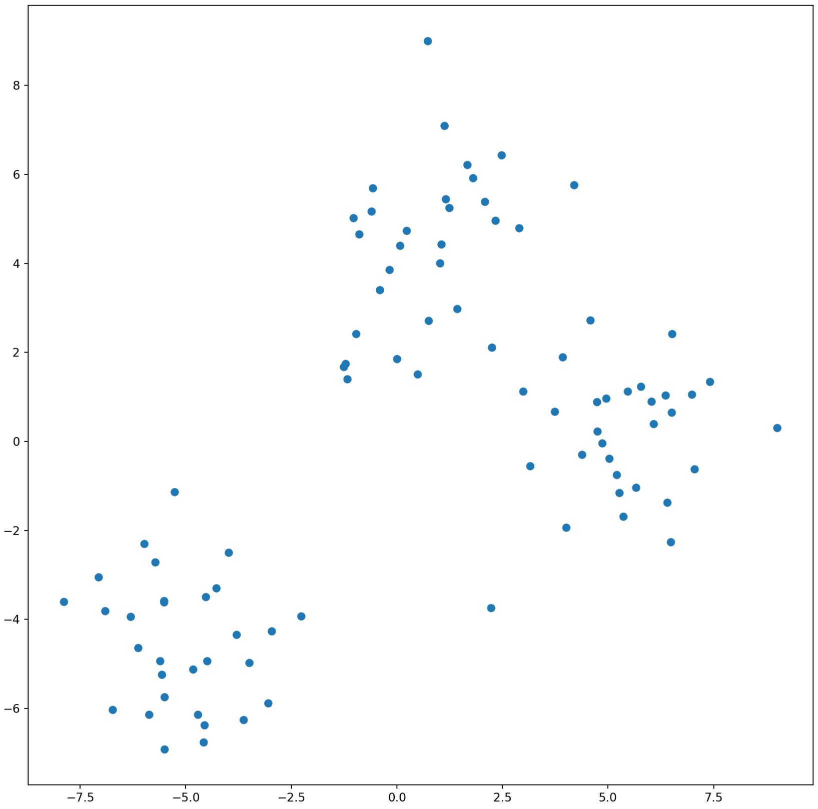
$$\hat{\varphi}_1(\mathbf{f}_i) = \sum_{k=1}^{n} \alpha_k^2 \gamma(\lambda_k)$$

$$\hat{\varphi}_2(\mathbf{f}_i) = \frac{\hat{\mathbf{f}}_i^\top \gamma(\mathscr{L}) \hat{\mathbf{f}}_i}{1 - \left(\hat{\mathbf{f}}_i^\top \xi_1\right)}$$

$$\hat{\varphi}_3(\mathbf{f}_i) = \sum_{j=2}^{k} (\gamma(2) - \gamma(\lambda_j)) \alpha_j^2$$

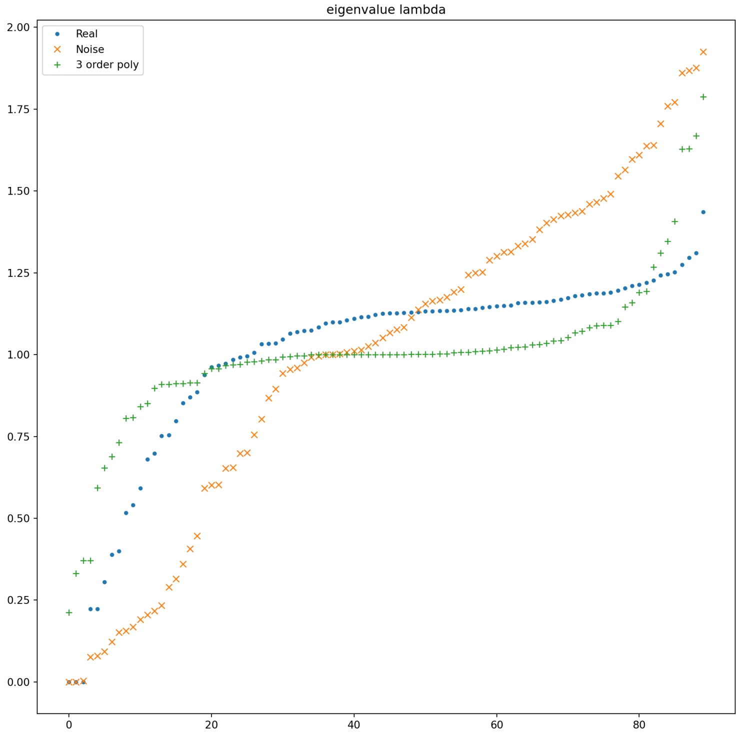
To illustrate how this regularization process can help reduce noise (still using the framework from the previous example), $\mathbf{X}$ was contaminated with random values from a normal distribution with a variance of 1.5.

```
random.seed(5678) # for replicability
noise = np.random.normal(0, 1.3, 90*3).reshape(90,3)
X_noise = X + noise
plt.scatter(X_noise[:,0], X_noise[:,1])
plt.show()
```



We plot the components of the eigenvalues of three normalized Laplacians: one from the original data, one from the noisy data, and one from the noisy data with a 3rd order polynomial regularization.

```
def isqrt(x):
  if x == 0:
    return 0
  else:
    return x**(-0.5)

v_isqrt = np.vectorize(isqrt)

def find_eig(X, k = lambda x:x):
  A = squareform(np.exp(-1 * pdist(X, ’sqeuclidean’)))
  rowsum = A.sum(axis=1)
  D = np.diag(rowsum)
  L = D - A
  D_is = v_isqrt(D)
  L_norm = k(D_is.dot(L).dot(D_is))
  vs,es = eigh(L_norm)
  arg_sort = vs.argsort()
  vs = vs[arg_sort]
  es = es[:,arg_sort]
  return vs,es, L_norm

vs,es,L = find_eig(X)
vs_n,es_n,L_noise = find_eig(X_noise)
vs_k,es_k, L_k = find_eig(X_noise, k = lambda x:x**3)

plt.plot(vs, ’.’, label = ’Real’)
plt.plot(vs_n, ’x’, label = ’Noise’)
plt.plot(vs_k, "+", label = ’3 order poly’)
plt.title(’eigenvalue lambda’)
plt.legend()
```

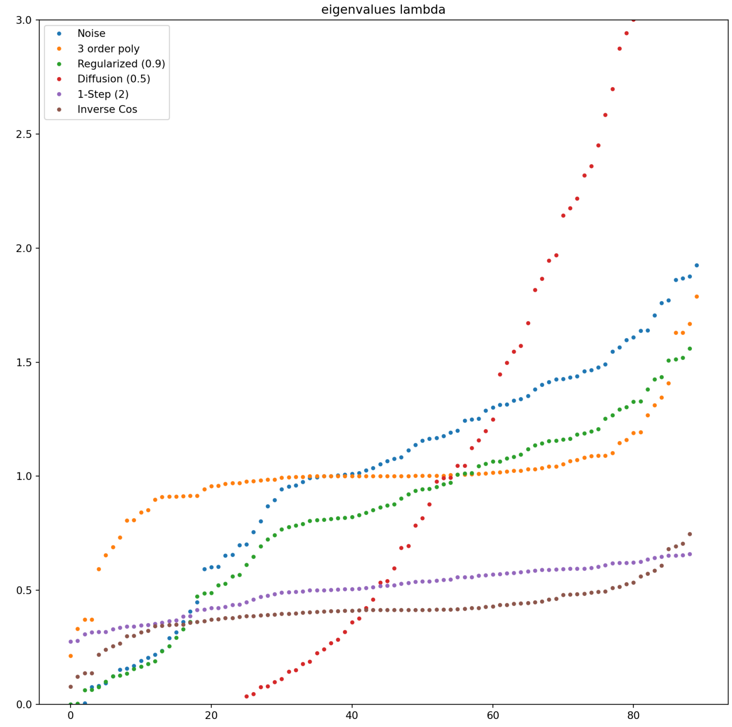


The preceding plot shows the effect of noise on $\mathscr{L}$'s: it tends to linearize the eigenvalues, and this provides much support to the poorer eigenvectors. The eigenvalues of the noisy Laplacian have been regularized using the standard cubic $\gamma(\lambda) = \lambda^3$; the distinction between the first eigenvalues and the rest is clear.

We can compare different kernels:

- **regularized Laplacian** – $\gamma(\lambda) = 1 + (0.9)^2\lambda$
- **high-order polynomial** – $\gamma(\lambda) = \lambda^3$
- **diffusion process** – $\gamma(\lambda) = \exp((0.3)^2/2\lambda)$
- $p$**–step random walk** – $\gamma(\lambda) = (2 - \lambda)^{-1}$
- **inverse cosine** – $\gamma(\lambda) = \cos(\lambda\pi/4)^{-1}$

```
vs_n,es_n,_ = find_eig(X_noise)
vs_reg, es_reg,_ = find_eig(X_noise, k = lambda x:1 + (0.9)**2 * x)
vs_k3,es_k3,_ = find_eig(X_noise, k = lambda x:x**3)
vs_diff, es_diff,_ = find_eig(X_noise, k = lambda x: np.exp((0.3)**2/(2*
vs_1step, es_1step,_ = find_eig(X_noise, k = lambda x: (2-x)**(-1))
vs_cos, es_cos, _ = find_eig(X_noise, k = lambda x: np.cos(x * (3.1416/4

plt.plot(vs_n, '.', label = 'Noise')
plt.plot(vs_k, '.', label = '3 order poly')
plt.plot(vs_reg, '.', label = 'Regularized (0.9)')
plt.plot(vs_diff, '.', label = 'Diffusion (0.5)')
plt.plot(vs_1step, '.', label = '1-Step (2)')
plt.plot(vs_cos, '.', label = 'Inverse Cos')

plt.ylim(0,3)
plt.title('eigenvalues lambda')
plt.legend()
plt.show()
```



The choice of a specific regularization function depends on the context and the goals of the data analysis task; for large datasets, considerations of ease of computation may also form part of the selection strategy.

**Spectral Feature Selection with SPEC** The remarks from the previous subsections can be combined to create a feature selection framework called **SPEC** [309]:

1. using a specified similarity function $s$, construct a similarity matrix $S$ of the data $\mathbf{X}$ (optionally with labels $\mathbf{Y}$);
2. construct the graph $G$ of the data;
3. extract the normalized Laplacian $\mathscr{L}$ from this graph;
4. compute the eigenpairs (eigenvalues and eigenvectors) of $\mathscr{L}$;
5. select a regularization function $\gamma(\cdot)$;
6. for each feature $\mathbf{f}_i$, $i = 1, \ldots, p$, compute the relevance $\hat{\varphi}(\mathbf{f}_i)$, where $\hat{\varphi} \in \{\hat{\varphi}_1, \hat{\varphi}_2, \hat{\varphi}_3\}$, and
7. return the features in descending order of relevance.

In order for SPEC to provide "good" results, proper choices for the similarity, ranking, and regularization functions are needed. Among other considerations, the similarity matrix should reflect the true relationships between the observations.

Furthermore, if the data is **noisy**, it might be helpful to opt for $\hat{\varphi} = \hat{\varphi}_3$ and/or $\gamma(\lambda) = \lambda^\nu$, $\nu \geq 2$. When the gap between the small and the large eigenvalues is wide, $\hat{\varphi} = \hat{\varphi}_2$ or $\hat{\varphi} = \hat{\varphi}_3$ usually provide **good choices**, although $\hat{\varphi}_2$ has been shown to be more robust [306].

### 23.4.4 Uniform Manifold Approximation and Projection

The feature selection and dimension reduction landscape is in flux, and there are more recent (and sophisticated) developments that are generating a lot of interest. Case in point, consider **Uniform Manifold Approximation and Projection** (UMAP) methods.

**Dimensionality Reduction and UMAP** A mountain is a 3-dimensional object.[52] And the surface of a mountain range is 2-dimensional – it can be represented with a flat map – even though the surface, and the map for that matter, still exist in 3-dimensional space.[53]

What does it mean to say that a shape is $q$-dimensional for some $q$? What is a **shape**, even?

Shapes could be lines, cubes, spheres, polyhedrons, or more complicated things. In geometry, the customary way to represent a shape is *via* a set of points $S \subseteq \mathbb{R}^p$.

A circle is the set of points whose distance to a fixed point (the centre) is **exactly equal** to the radius $r$, say, whereas a disk is the set of points whose distance to the centre is **at most** the radius.

In the mountain example, $p = 3$ for both the mountains $S_m$ and the mountain surface $S_s$. So the question is, when is the **(effective) dimension** of a set $S$ less than $p$, and how is that dimension calculated?

[52] When we consider the world at a low resolution, at least.

[53] In the parlance of the field, we say that the surface is **embedded** in $\mathbb{R}^3$.

It turns out that there are multiple definitions of the **dimension** $q$ of a set $S \subseteq \mathbb{R}^p$ [310]:

- the smallest $q$ for which $S$ is $q$-dimensional manifold;
- how many nontrivial $n^{\text{th}}$ homology groups of $S$ there are;
- how the "size" of the set scales as the coordinates scale.

A **$q$-dimensional manifold** is a set where each small region is approximately the same as a small region of $\mathbb{R}^q$. For instance, if a small piece of a (stretchable) sphere is cut out with a cookie cutter, it could theoretically be bent so that it looks like it came from a flat plane, without changing its "essential" shape.

Dimensionality reduction is more than just a matter of selecting a definition and computing $q$, however. Indeed, any dataset $\mathbf{X}$ is necessarily finite and is thus, by definition, actually $0-$dimensional; the object of interest is the shape that the data **would** form if there were infinitely many available data points, or, in other words, the **support of the distribution** generating the data.

Furthermore, any dataset is probably noisy and may only **approximately lie** in a lower-dimensional shape.

Lastly, it is not clear how to build an algorithm that would, for example, determine what all the **homology groups** of some set $S$ are. The problem is quite thorny. Let $X \subseteq \mathbb{R}^p$ be a finite set of points. A **dimensionality reducer** for $\mathbf{X}$ is a function $f_{\mathbf{X}} : \mathbf{X} \to \mathbb{R}^q$, where $q < p$, which satisfies certain properties that imply that $f_{\mathbf{X}}(\mathbf{X})$ has similar structure to $\mathbf{X}$.[54]

Various dimensionality reducers were discussed in Section 23.2; they each differ based on the relationship between $\mathbf{X}$ and $f_{\mathbf{X}}(\mathbf{X})$.

For instance, in PCA, the dataset $\mathbf{X}$ is first translated so that its points (or at least its "principal components") lie in a linear subspace. Then $q$ unit-length linear basis elements are chosen to span a subspace, projection onto which yields an affine map $f$ from $\mathbf{X}$ to $\mathbb{R}^q$ that preserves Euclidean distances between points (a rigid transformation), assuming that the non-principal dimensions are ignored.

PCA seems reasonable but what if a rigid transformation down to $\mathbb{R}^q$ is not possible? As an example, consider the **swiss roll** of Figure 23.13, which is a loosely rolled up rectangle in 3-dimensional space. What can be preserved when we "reduce" this space? Only the local structure? The global structure?

UMAP is a dimension reduction method that attempts to approximately preserve **both the local and global structure**. It can be especially useful for visualization purposes, i.e., reducing to $q = 3$ or fewer dimensions. While the semantics of UMAP can be stated in terms of graph layouts, the method was derived from abstract topological assumptions. For a full treatment and mathematical properties, see [311].

Note that UMAP works best when the data $\mathbf{X}$ is evenly distributed on its support $\mathcal{S}$. In this way, the points of $\mathbf{X}$ "cover" $\mathcal{S}$ and UMAP can determine where the true gaps or holes in $S$ are.

[54] In the remainder of this section, the subscript is dropped. Note that $q$ is assumed, not found by the process.

**UMAP Semantics** Let the (scaled) data be denoted by $\mathbf{X} = \{\mathbf{x}_1, \ldots, \mathbf{x}_n\}$, where $\mathbf{x}_i \in \mathbb{R}^p$ for all $i$; let $d : \mathbf{X} \times \mathbf{X} \longrightarrow \mathbb{R}_{\geq 0}$ be a distance function, and let $k \geq 1$ be an integer.

Consider a **directed graph** graph $D = (V, E, A)$, with

- vertices $V(D) = \mathbf{X}$;
- edges $E(D)$ consisting of the ordered pairs $(\mathbf{x}_i, \mathbf{x}_j)$ such that $\mathbf{x}_j$ is one of the $k$ nearest neighbours of $\mathbf{x}_i$ according to $d$;
- weight function $W : E(D) \longrightarrow \mathbb{R}$ such that

  $$w(\mathbf{x}_i, \mathbf{x}_{i,j}) = \exp\left(\frac{-\max(0, d(\mathbf{x}_i, \mathbf{x}_{i,j}) - \rho_i)}{\sigma_i}\right),$$

  where $\mathbf{x}_{i,1}, \ldots, \mathbf{x}_{i,k}$ are the $k$ nearest neighbours of $\mathbf{x}_i$ according to $d$, $\rho_i$ is the minimum nonzero distance from $\mathbf{x}_i$ to any of its neighbours, and $\sigma_i$ is the unique real solution of

  $$\sum_{j=1}^{k} \exp\left(\frac{-\max(0, d(x_i, x_{i,j}) - \rho_i)}{\sigma_i}\right) = \log_2(k),$$

  and
- $A$ is the weighted adjacency matrix of $D$ with vertex ordering $\mathbf{x}_1, \ldots, \mathbf{x}_n$.

Define a symmetric matrix

$$B = A + A^\top - A \circ A^\top,$$

where $\circ$ is **Hadamard's component-wise product**.

The graph $G = (V, W, B)$ has the same vertex set, the same vertex ordering, and the same edge set as $D$, but its edge weights are given by $B$. Since $B$ is symmetric, $G$ can be considered to be undirected.

UMAP returns the (reduced) points $f(\mathbf{x}_1), \ldots, f(\mathbf{x}_n) \in \mathbb{R}^q$ by finding the position of each vertex in a **force directed graph layout**, which is defined *via* a graph, an attractive force function defined on edges, and a repulsive force function defined on all pairs of vertices.

Both force functions produce **force values** with a direction and magnitude based on the pair of vertices and their respective positions in $\mathbb{R}^q$.

To compute the **layout**, initial positions in $\mathbb{R}^q$ are chosen for each vertex, and an iterative process of translating points based on their attractive and repulsive forces is carried out until a convergence criterion is met. In UMAP, the attractive force between vertices $\mathbf{x}_i, \mathbf{x}_j$ at positions $y_i, y_j \in \mathbb{R}^q$, respectively, is

$$\frac{-2ab\|y_i - y_j\|_2^{2(b-1)}}{1 + \|y_i - y_j\|_2^2} w(x_i, x_j)(y_i - y_j),$$

where $a$ and $b$ are parameters, and the repulsive force is

$$\frac{b(1 - w(x_i, x_j))(y_i - y_j)}{(0.001 + \|y_i - y_j\|_2^2)(1 + \|y_i - y_j\|_2^2)}.$$

There are a number of important **free parameters** to select, namely

- $k$: nearest neighbor neighborhood count;
- $q$: target dimension;
- $d$: distance function, e.g. Euclidean metric.

The UMAP documentation states,

> low values of $k$ will force UMAP to concentrate on very local structure (potentially to the detriment of the big picture), while large values will push UMAP to look at larger neighborhoods of each point . . . losing fine detail structure for the sake of getting the broader structure of the data [311].

The user may set these parameters to appropriate values for the dataset. The choice of a distance metric plays the same role as in clustering, where closer pairs of points are considered to be more similar than farther pairs. There is also a minimum distance value used within the force directed layout algorithm which says how close together the positions may be.

**Example** We compare various dimensionality reducers for a number of datasets (adapted from the UMAP documentation).[55] Let us start by installing the required Python modules.

```
import warningswarnings.filterwarnings(’ignore’)
import numpy as np
import matplotlib.pyplot as plt
from sklearn import datasets, decomposition, manifold, preprocessing
from colorsys import hsv_to_rgb
import umap.umap_ as umap
```

We will plot the two-dimensional reduction of five different datasets; the points will be coloured to get a sense of which points got sent where (otherwise we would only know the shape of the reduced dataset, but have no way to tell how the local structure is affected by the reducers).

We use $t-$SNE, Isomap, MDS (multidimensional scaling), PCA, and UMAP. The 5 datasets are:

- a set consisting of 4 distinct 10-dimensional Gaussian distributions;
- the digit classification dataset;
- the wine characteristics dataset (essentially 1D);
- a 2D rectangle rolled up in 3D space (colours indicate the position along the unrolled rectangle), and
- points on the 2D surface of a 3D sphere (which is not homeomorphic to $\mathbb{R}^2$)); even with the north pole removed, the stereographic projection would map points close to the north pole arbitrarily far from the origin (colour hue indicates the angle around the equator and darkness indicates distance from south pole).

```
blobs, blob_labels = datasets.make_blobs(
    n_samples=500, n_features=10, centers=4
  )
```

[55] Careful: the correct Python package to install is umap-learn, not umap.

```
digits = datasets.load_digits(n_class=10)

wine = datasets.load_wine()

swissroll, swissroll_labels = datasets.make_swiss_roll(
    n_samples=1000, noise=0.1
  )

sphere = np.random.normal(size=(600, 3))

# scale points to have same distance from origin
sphere = preprocessing.normalize(sphere)

# compute colours in HSV format
sphere_hsv = np.array([
        (
          (np.arctan2(c[1], c[0]) + np.pi) / (2 * np.pi),
          np.abs(c[2]), min((c[2] + 1.1), 1.0),
        )
      for c in sphere
    ])

# convert colours to RGB format
sphere_colors = np.array([hsv_to_rgb(*c) for c in sphere_hsv])
```

Next we set parameters for the reducer algorithms. For UMAP, we set `min_dist` to 0.3 which will spread out the points to a noticable degree. We also set the number $k$ of nearest neighbours to 30.

The choices for the other reduces are shown in the code block.

In practice, we would typically set these parameters on a dataset-by-dataset basis, but for illustration purposes, we do not need to fine tune the choices.

```
reducers = [
    (manifold.TSNE, {"perplexity": 50}),
    (manifold.Isomap, {"n_neighbors": 30}),
    (manifold.MDS, {}),
    (decomposition.PCA, {}),
    (umap.UMAP, {"n_neighbors": 30, "min_dist": 0.3}),
]

test_data = [
    (blobs, blob_labels),
    (digits.data, digits.target),
    (wine.data, wine.target),
    (swissroll, swissroll_labels),
    (sphere, sphere_colors),
]

dataset_names = ["Blobs", "Digits", "Wine", "Swiss Roll",
    "Sphere"]
```

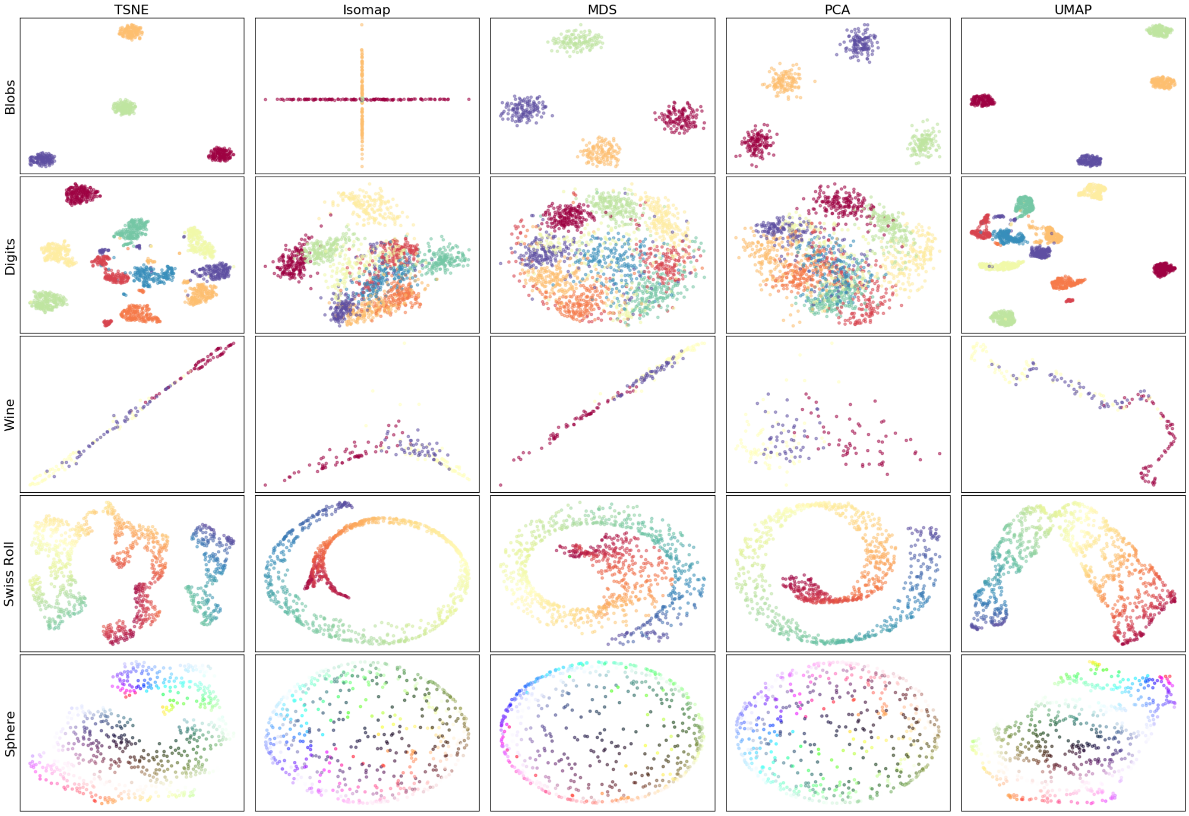
Now, we compute the 2D reductions for every reducer-dataset pair (this step is time-consuming):

```
reductions_and_labels = [(reducer(n_components=2,
      **args).fit_transform(data),
      labels)
    for data, labels in test_data
    for reducer, args in reducers
  ]
```

And we display the results:

```
n_rows = len(test_data)
n_cols = len(reducers)
fig = plt.figure(figsize=(16, 16))

fig.subplots_adjust(left=.02, right=.98, bottom=.001,
  top=.96, wspace=.05, hspace=.02)

ax_index = 1
ax_list = []

for reduction, labels in reductions_and_labels:
    ax = fig.add_subplot(n_rows, n_cols, ax_index)
    if isinstance(labels[0], tuple):
        # if labels are colours, use them
        ax.scatter(*reduction.T, s=10, c=labels, alpha=0.5)
    else:
        # otherwise, use "spectral" map from labels to colours
        ax.scatter(*reduction.T, s=10, c=labels,
            cmap="Spectral", alpha=0.5)
    ax_list.append(ax)
    ax_index += 1

plt.setp(ax_list, xticks=[], yticks=[])

for i in np.arange(n_rows) * n_cols:
    ax_list[i].set_ylabel(dataset_names[i // n_cols], size=16)

for i in range(n_cols):
    ax_list[i].set_xlabel(repr(reducers[i][0]()).split("(")[0],
        size=16)
    ax_list[i].xaxis.set_label_position("top")

fig.show()
```

Notice that Isomap removes exactly the wrong dimension in the swiss roll; $t$−SNE (and MDS to some extent) reduces the wine data down to one dimension (its true dimensionality) even though it was only asked for a reduction to two dimensions. UMAP manages to give the most sensible output for the swiss roll.



## 23.5 Exercises

Consider the datasets

- `GlobalCitiesPBI.csv`
- `2016collisionsfinal.csv`
- `polls_us_election_2016.csv`
- `HR_2016_Census_simple.xlsx`
- `UniversalBank.csv`.

and/or any other datasets of interest (as long as they have a sufficiently large number of predictors).

1. Establish 2-3 questions that you could try to answer with each dataset.
2. Based on the questions obtained in 1, provide 3-5 subsets of features that would do a good job of representing each dataset (use some of the methods described in this module, or other methods as needed).
3. Learn 3-5 reduced manifolds for each dataset (use some of the methods described in this module, or other methods as needed).
4. How would you validate your results?