

AGIW - Primo Progetto

Gruppo: AdR&SS

Angelo del Re 441476

Stefano Silvi 461993

Codice del progetto disponibile su GitHub:
<https://github.com/AdR21/Agiw2018>

Step 0

Il dominio principale assegnatoci è quello degli “Headphones”. Abbiamo scaricato le pagine html tramite uno script, verificando che quasi 500 domini fossero validi, ossia privi di errori. Lo stesso abbiamo fatto per il secondo dominio, quello delle “Tv”, comparando i risultati ottenuti tra i due.

Tipologia	Domini	Url	Tempo
Headphones	473	70.000	13h
Tv	836	100.000	15h

Step 1

Partendo dal totale dei domini ottenuti abbiamo effettuato una “pulizia” mediante l’eliminazione di tutti quelli che hanno link non validi o con errori, per esempio, “4xx” o “3xx”. Per le cuffie siamo passati da 473 a 46, invece per le tv da 836 a 47. A questo punto, manualmente, li abbiamo analizzati per ricavare soltanto quelli con le specifiche di nostro interesse, ossia pagine con un prodotto inerente al dominio, con descrizione delle sue specifiche. Per le cuffie il numero si è ridotto a 16, invece per le tv a 18. Dopo abbiamo eseguito lo script “Spexa” su entrambi i domini. I json ottenuti sono stati classificati in vuoti e in pieni: per gli headphones 400 vuoti e 244 pieni, per le tv 510 vuoti e 423 pieni.

Script utilizzato per ricavare domini e link validi:

```
1. import os
2. import json
3. import re
4.
5. url_file = "dexter_urls_category_headphone_2018-03-27_14:11:59.json"
6. url_json = open(url_file).read()
7. url_dictionary = json.loads(url_json)
8. os.chdir("headphone")
9. for current in os.listdir("."):
10.     with open(current + "/index.txt", "r") as index:
11.         for line in index:
12.             status = line.split('\t',1)[1]
13.             if re.match("3[0-9][0-9]|4[0-9][0-9]|5[0-9][0-9]|ConnectionError|Timeout",status):
14.                 url = line.split('\t',1)[0]
15.                 url_dictionary[current].remove(url)
16. os.chdir("../")
17. del(url_dictionary["www.calibex.co.uk"])
18. result = open("cleanedurls.json", "w")
19. json.dump(url_dictionary, result, indent=4, separators=(',', ': '))
20. result.close()
```

Risultati ottenuti con la “Spexa” al primo utilizzo e senza pulizia manuale dei domini:

Tipologia	Domini	Url	Tempo	Json tot	Json vuoti	Json pieni
Headphones	46	3.600	2h	1.100	827	273
Tv	47	4.500	2h22'	4.212	3.205	1.008

Risultati ottenuti con “Spexa” al secondo utilizzo e con pulizia manuale dei domini:

Tipologia	Domini	Url validi	Tempo	Json tot	Json vuoti	Json pieni
Headphones	15	644	27'30"	644	400	244
Tv	18	933	32'53"	930	510	423

Step 2

Lo scopo nello “Step 2” è di superare “Spexa” in termini di contenuti, ossia cercar di estrarre più informazioni rispetto allo script. Il nostro algoritmo è diviso in tre fasi principali. La prima consiste nell’estrarre una base di conoscenza da un dataset di pagine già pre-annotate. I dataset sono stati recuperati da <http://webdatacommons.org/productcorpus/>, che li mette a disposizione per entrambi i domini. Tramite lo script abbiamo estratto due json, uno per dominio, contenenti informazioni sulle specifiche dei prodotti.

Parte della base di conoscenza:

```

    "compatibility": [
      "Consumer Headphones, Studio/Live Monitoring Headphones",
      "Universal",
      "Consumer Headphones",
      "Studio/Live Monitoring Headphones"
    ],
    "connectivity_technology": [
      "3.5mm Jack",
      "Wired"
    ],
    "headphones_form_factor": [
      "In-Ear only",
      "Headband",
      "In-Ear Only",
      "In-Ear Monitors/IEMs",
      "Earbud (In Ear)",
      "Full-Size Headphones"
    ],
    "included_accessories": [
      "Carrying Case, Replacement Cable",
      "Carry Bag/Case, Extension Cable",
      "Carrying Case",
      "Carry Bag/Case",
      "Carrying Case, TRS 1/4\" Adapter"
    ],
  ],

```

Esempio di mapping sulle chiavi della base di conoscenza:

```

    ],
    "Compatibility": [
      "compatibility",
      "headphone_use"
    ],
    "Connectors": [
      "connectivity_technology"
    ],
    "Frequency Range": [
      "frequency_response"
    ],
    "EAN": [
      "product_code",
      "product_gtin"
    ],
  ],

```

Metodo “json2list” dello script “knoedbaseextractor”:

```

1. def json2list(self):
2.     with open(self._specs_file) as json_file:
3.         full_dictionary = json.load(json_file)
4.         res = []
5.         for elem in full_dictionary:
6.             new_elem = {
7.                 self._table_attributes: {},
8.                 self._list_attributes: {},
9.                 self._attributes_mapping: {}
10.            }
11.            for mapping in elem[self._attributes_mapping]:
12.                strings = mapping.split(":", 1)
13.                key = strings[0].strip()
14.                value = strings[1].strip()
15.                new_elem[self._attributes_mapping][key] = value
16.            mapping_copy = new_elem[self._attributes_mapping].copy()
17.            for item in elem[self._table_attributes]:
18.                strings = item.split(":", 1)
19.                if len(strings) == 2:
20.                    key = strings[0].strip()
21.                    value = strings[1].strip()
22.                    mapped_key = self.key_mapping(key, mapping_copy)
23.                    new_elem[self._table_attributes][mapped_key] = value
24.            for item in elem[self._list_attributes]:
25.                strings = item.split(":", 1)
26.                key = strings[0].strip()
27.                value = strings[1].strip()
28.                mapped_key = self.key_mapping(key, mapping_copy)
29.                new_elem[self._list_attributes][mapped_key] = value
30.            res.append(new_elem)
31.        return res

```

Nella seconda fase prendiamo il json contenente i link utilizzati nello “Step 1” e avviamo il nostro algoritmo per estrarre le informazioni. Quest’ultimo estrapola tutto ciò che è contenuto in tabelle e in liste. Per selezionarli utilizza la libreria python “Beautiful Soup”, specifica per file html e xml. Se in una pagina è presente il tag <h1> che, generalmente, contiene il nome del prodotto lo scrive su un file csv. Per tutte le tabelle prenderà le righe, ossia comprese tra tag <tr>. Se le righe contengono elementi del tipo <chiave, valore>, gli elementi vengono separati e scritti una riga per volta sul csv. Se, invece, contengono un valore unico vengono scritte così come sono. Lo stesso avviene per le liste.

Metodo “extract” dello script “csv_extractor”:

```

1. def extract(self):
2.     with open("cleanedurls_tv.json") as urls:
3.         sites = json.load(urls)
4.         print("%s" % sites)
5.         for domain in tqdm(sites):
6.             print(domain)
7.             values = sites[domain]
8.             if len(values) != 0:
9.                 if not os.path.exists(self._prefix + domain):
10.                    os.mkdir(self._prefix + domain)
11.                    i = 1
12.                    index_csv = open(self._prefix + domain + "/index_csv.txt", "w")
13.                    for site in tqdm(values):
14.                        print(site)

```

```

15.         try:
16.             req = Request(site, headers={'User-
Agent': "Magic Browser"})
17.             html = urlopen(req)
18.             bsObj = BeautifulSoup(html, "lxml")
19.             for script in bsObj(["script", "style"]):
20.                 script.decompose()
21.             csvFile = open(self._prefix + domain + "/" + str(i) + ".csv
", 'wt', newline='')
22.             writer = csv.writer(csvFile)
23.             csvRows = list()
24.             product_name = bsObj.find("h1")
25.             if product_name:
26.                 writer.writerow(["name:", product_name.get_text()])
27.             tables = bsObj.findAll("table")
28.             index_csv.write(site + "\t" + str(i) + ".csv\n")
29.             if tables:
30.                 for table in tables:
31.                     rows = table.findAll("tr")
32.                     for row in rows:
33.                         csvRow = list()
34.                         for cell in row.findAll(['th', 'td']):
35.                             text = ' '.join(
36.                                 cell.get_text().replace('\n', ' ').repla
ace('\t', ' ').strip().split())
37.                             if text:
38.                                 if ':' in text:
39.                                     strings = text.split(':', 1)
40.                                     elements = self.group(strings, 2)
41.                                     for el in elements:
42.                                         if el not in csvRows:
43.                                             csvRows.append(el)
44.                                 else:
45.                                     if len(text.split()) <= self._MAXIM
UM_TEXT_LENGTH:
46.                                         csvRow.append(text)
47.                                         full_text = cell.get_text().replace('.', '\
n').splitlines()
48.                                         for string in full_text:
49.                                             if 0 < len(string.split()) <= self._MAX
IMUM_TEXT_LENGTH and ':' in string:
50.                                                 strings = string.replace('\n', ' ')
51.                                                 .replace('\t', ' ').strip().split(':', 1)
52.                                                 elements = self.group(strings, 2)
53.                                                 for el in elements:
54.                                                     if el not in csvRows:
55.                                                         csvRows.append(el)
56.                                                 if len(csvRow) > 0:
57.                                                     if csvRow not in csvRows:
58.                                                         csvRows.append(csvRow)
59.             lists = bsObj.findAll("ul")
60.             if lists:
61.                 for lis in lists:
62.                     li = lis.findAll("li")
63.                     for row in li:
64.                         csvRow = list()
65.                         if row.get_text().strip():
66.                             text = ' '.join(
67.                                 row.get_text().replace('\n', ' ').repla
ce('\t', ' ').strip().split())
68.                             if ':' in text:
69.                                 strings = text.split(':', 1)
70.                                 elements = self.group(strings, 2)

```

```

70.                 for el in elements:
71.                     if el not in csvRows:
72.                         csvRows.append(el)
73.                     else:
74.                         if len(text.split()) <= self._MAXIMUM_T
EXT_LENGTH:
75.                             csvRow.append(text)
76.                         if len(csvRow) > 0:
77.                             if csvRow not in csvRows:
78.                                 csvRows.append(csvRow)
79.                 for r in csvRows:
80.                     writer.writerow(r)
81.                 i += 1
82.                 csvFile.close()
83.             except Exception as e:
84.                 print(e)
85.                 continue
86.             index_csv.close()

```

Nella terza fase leggiamo i csv prodotti precedentemente e ne estraiamo il contenuto rilevante. Quindi viene effettuata una pulizia e convertito il file in json. Per verificare che le informazioni siano valide o meno per il dominio corrente, sfruttiamo la base di conoscenza ottenuta dalla prima fase. Utilizzando la “cosine similarity” constatiamo se il valore nella riga del csv ha una buona corrispondenza nella base. Se ciò avviene, la riga viene considerata come buona e, quindi, memorizzata. Se invece l’elemento non è simile a niente, si controlla la sua posizione nel documento. Se è sufficientemente vicino a un dato già estratto e non contiene un numero troppo elevato di parole, viene preso. Se non vale tutto ciò che avviene prima, viene scartato.

Funzione “get_csv_information_as_dict” dello script “csv_cleaner”:

```

1. def get_csv_information_as_dict(self, csv_path):
2.     result = dict()
3.     data = list()
4.     with open(csv_path) as csv_specs:
5.         reader = csv.reader(csv_specs, delimiter=",")
6.         for line in reader:
7.             data.append(line)
8.         datas = self.chunks(data, math.ceil(len(data)/self.cpu_num))
9.         executor = ThreadPoolExecutor(max_workers=self.cpu_num)
10.        wait_for = [executor.submit(self.parallel_search, d.copy(), self.base.copy(
)) for d in datas]
11.        for future in as_completed(wait_for):
12.            result.update(future.result())
13.        executor.shutdown()
14.        return result

```

Estrae il file csv e legge le righe, dopodiché separa le righe in insiemi pari al numero di core disponibili sulla macchina. Dà in pasto ad ogni thread un sottoinsieme di righe. Attende che l’esecuzione dei thread termini e ricompone il risultato.

Funzione “parallel_search”:

```

1. def parallel_search(self, data, base):
2.     result = dict()
3.     to_be_evaluated = dict()
4.     for i, line in enumerate(data):
5.         if line:
6.             if line[0] == "name:":

```

```

7.         self.put_if_not_exists(result, "Name", line[1])
8.     elif len(line) >= 2:
9.         key = line[0]
10.        value = line[1]
11.        if len(key.split()) <= self._MAXIMUM_KEY_LENGTH:
12.            known = self.get_matching_key(key, base)
13.            if known:
14.                self.put_if_not_exists(result, key, value)
15.            else:
16.                to_be_evaluated[i] = [key, value]
17.        elif len(value.split()) <= self._MAXIMUM_VALUE_LENGTH:
18.            known = self.get_matching_value(value, base)
19.            if known:
20.                self.put_if_not_exists(result, key, value)
21.            else:
22.                to_be_evaluated[i] = [key, value]
23.    for index in to_be_evaluated:
24.        key = to_be_evaluated[index][0]
25.        value = to_be_evaluated[index][1]
26.        if (len(key.split()) <= self._MAXIMUM_KEY_LENGTH and
27.            len(value.split()) <= self._MAXIMUM_VALUE_LENGTH and
28.            self.search(data, result, index, self._ROUND)):
29.            self.put_if_not_exists(result, key, value)
30.    return result

```

All'inizio controlla se si ha a disposizione il nome del prodotto, se c'è lo memorizza. Se la riga contiene più di un singolo elemento, considera il primo come se fosse una chiave e i successivi come se fossero il valore associato a essa. Verifica se la chiave o il valore sono rilevanti nel dominio. Se almeno uno dei due lo è, li memorizza; altrimenti, li mette in attesa per una valutazione successiva. Se la ricerca va a buon fine si può considerare l'elemento come rilevante, altrimenti viene scartato. Nell'ultimo passaggio si cerca di fare una valutazione sugli elementi messi da parte. Per ogni coppia <chiave, valore> non ancora valutata, se la chiave non contiene più di dieci parole e il valore più di venti, sfrutta il metodo "search", il cui funzionamento verrà spiegato più avanti.

Funzioni "get_matching_key" e "get_matching_value":

```

1. def get_matching_key(self, key, base):
2.     for known in base["atts_map"]:
3.         if self.cosine_sim(key.lower(), known.lower())>=self._MIN_SIMILARITY:
4.             return known
5.     return ''
6. def get_matching_value(self, value, base):
7.     values = [item for sublist in base["table_atts"].values() for item in subli
8. st]
9.     values.extend([item for sublist in base["list_atts"].values() for item in s
10. ublist])
11.     for known in values:
12.         if self.cosine_sim(value.lower(), known.lower())>=self._MIN_SIMILARITY:
13.             return known
14.     return ''

```

Questi due metodi verificano se i valori nei csv sono simili a valori presenti nella base di conoscenza, se ciò si verifica, ritorna il primo valore con cui ha fatto "matching".

Funzione “cosine_sim”:

```
1. def cosine_sim(self, text1, text2):
2.     vectorizer = TfidfVectorizer()
3.     try:
4.         tfidf = vectorizer.fit_transform([text1, text2])
5.         return (tfidf * tfidf.T).A[0, 1]
6.     except ValueError as e:
7.         return 0.0
```

Metodo che calcola la “cosine similarity” tra due stringhe, per il calcolo utilizza la libreria “scikit-learn”.

Funzione “search”:

```
1. def search(self, data, extracted, index, around):
2.     for i in range(-around + index, around + index):
3.         try:
4.             if data[i][0] in extracted or data[i][1] in extracted.values():
5.                 return True
6.         except IndexError:
7.             continue
8.     return False
```

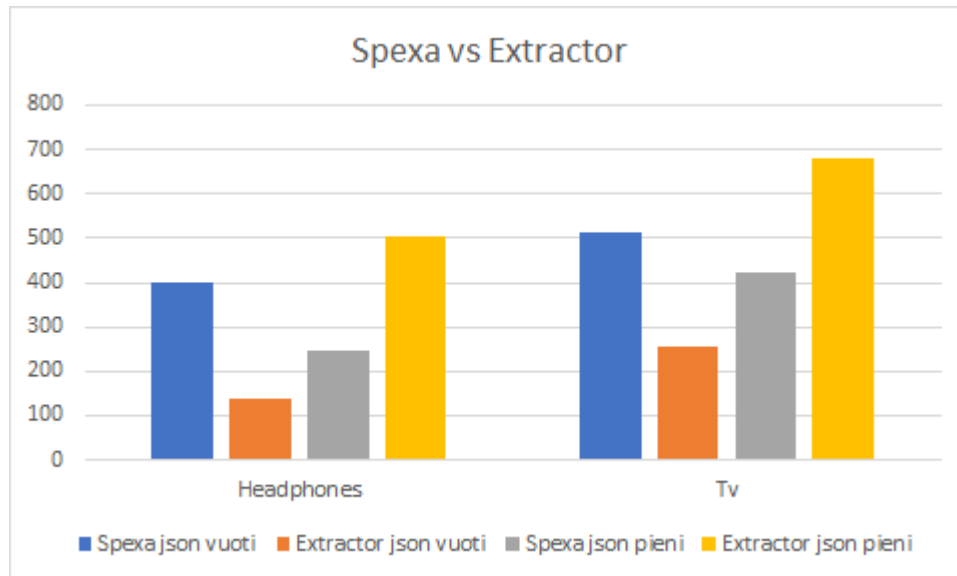
Metodo che prende in input le righe del csv, i dati estratti fino a quel momento, l’indice della riga che si vuole valutare e l’intorno entro il quale effettuare le ricerche. La funzione verifica se i dati contenuti in qualcuna delle righe intorno a quella specificata dall’indice sono stati estratti. Se ciò avviene restituisce “true” altrimenti “false”.

Lo script conclude con la scrittura, dei dati estratti, in un file json.

Tipologia	Domini	Url validi	Tempo	Json tot	Json vuoti	Json pieni
Headphones	15	644	49'27"	644	114	530
Tv	18	933	1h53'12"	933	252	681

“Spexa” vs “Extractor”

Tipologia	Spexa json vuoti	Extractor json vuoti	Spexa json pieni	Extractor json pieni
Headphones	400	114	244	530
Tv	510	252	420	681



Precision dominio degli “Headphones”

Spexa: $241(\text{json pieni e validi})/644=0,37$


Extractor: $530(\text{json pieni e validi})/644=0,82$

Precision dominio delle “TV”

Spexa: $159(\text{json pieni e validi})/933=0,17$

Extractor: $678(\text{json pieni e validi})/933=0,72$


Prestazioni simili



JVC HA-FX1X In-Ear only Headphones - Black

Out of stock | [Similar in Headphones](#)

★★★★☆

 [Ask Friends for feedback](#)

Mouseover to zoom or click to see larger image

Overview

Compare Prices

JVC HA-FX1X In-Ear only Headphones - Black

The JVC HA-FX1X Headphones are designed to provide an immersive listening experience. It features Extreme Deep Bass Ports and a 10 mm diaphragm with neodymium magnet for enhanced bass effects. This inner-ear headphone offers three sizes of silicone earpieces, its rubber protectors ensure durability. For a convenient user experience, it comes with a carrying case. Besides, this black headphone has a 3.5 mm gold-plated connector.

Product Details and Features

Miscellaneous	
Weight	5.2 gr
Features	
Impedance	16 ohms
Frequency Response	5 Hz - 23 kHz
Cable Length	1.2 m
Sensitivity	104 dB
Connectivity Type	Wired
Fit Design	In-Ear only
Use	Personal Audio
Driver Unit Size	10 mm
Cable Length (m)	1.2
Earpiece Design	Earbud (In Ear)
Features	Noise Isolating
Product Identifiers	
UPC	4975769390838
MPN	HA-FX1X-E
Other Features	
Brand	JVC

Spexa:

```
[
{
  "Use": "Personal Audio",
  "Cable Length (m)": "1.2",
  "Features": "Noise Isolating",
  "Weight": "5.2 gr",
  "MPN": "HA-FX1X-E",
  "Sensitivity": "104 dB",
  "Earpiece Design": "Earbud (In Ear)",
  "Frequency Response": "5 Hz - 23 kHz",
  "UPC": "4975769390838",
  "Brand": "JVC",
  "Fit Design": "In-Ear only",
  "Impedance": "16 ohms",
  "Cable Length": "1.2 m",
  "Connectivity Type": "Wired",
  "Driver Unit Size": "10 mm"
}
]
```

Extractor:

```
{
  "Name": "JVC HA-FX1X In-Ear only Headphones - Black",
  "Weight": "5.2 gr",
  "Impedance": "16 ohms",
  "Frequency Response": "5 Hz - 23 kHz",
  "Cable Length": "1.2 m",
  "Sensitivity": "104 dB",
  "Fit Design": "In-Ear only",
  "Use": "Personal Audio",
  "Cable Length (m)": "1.2",
  "Earpiece Design": "Earbud (In Ear)",
  "Features": "Noise Isolating",
  "UPC": "4975769390838",
  "MPN": "HA-FX1X-E",
  "Brand": "JVC",
  "Connectivity Type": "Wired",
  "Driver Unit Size": "10 mm"
}
```



Panasonic Smart Viera TX-L42E6B 42" 1080p HD LED LCD Internet TV

Out of stock | [Similar in Televisions](#)

★★★★★

3D Technology: 3D Not Supported

Type: Flat-Panel

Max. Resolution: 1080p

WiFi Connection: Built-in WiFi

Built-in Digital Tuner: Freeview HD

Smart TV Features: Skype Ready, Wireless LAN, Downloadable Apps, Internet Browsing, Internet Streaming, Interface Home Network, Streaming Internet Apps, Web Browser

[See more features](#)

[Ask Friends for feedback](#)

[Overview](#)

[Compare Prices](#)

Panasonic Smart Viera TX-L42E6B 42" 1080p HD LED LCD Internet T...

Product Details and Features

Miscellaneous

Exterior Colour Silver

Dimensions

Height 56.39 cm
Width 95.68 cm
Weight 13.48 Kg
Depth 4.88 cm

Audio Features

Audio Type Surround
Speaker Power 2 x 20 Watt
Number Of Speakers 2

Connectivity

Rear Output Connectors Optical Digital • Headphones
Connectors USB 2.0 • SCART

Technical Features

Refresh Rate 100 Hz
Power Source AC Adapter
Screen Type Flat Screen
Supported File Formats WAV • WMV • WMA • MP3 • JPEG • AAC • AVI
Viewing Angle 176°

Spexa:

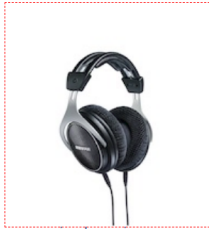
Extractor:

```
{
  "Contrast Ratio": "1000000:1",
  "Definition": "HDTV",
  "Number Of Speakers": "2",
  "Smart TV Features": "Internet Browsing",
  "Refresh Rate": "100 Hz",
  "Type": "Flat-Panel",
  "Display Resolution": "1920 x 1080 pixels",
  "Built-in Digital Tuner": "DVB-T • DVB-T2 • DVB-S2",
  "EAN": "4905524967739",
  "3D Technology": "3D Not Supported",
  "Max. Resolution": "1080p",
  "Display Technology": "LED LCD",
  "Viewing Angle": "89°",
  "Audio Type": "Surround",
  "MPN": "KDL42W706BSU",
  "Series": "W Series",
  "Product Line": "Bravia",
  "Screen Size": "42\"",
  "WiFi Connection": "Built-in WiFi"
}
```

```
{
  "3D Technology": "3D Not Supported",
  "Type": "Flat-Panel",
  "Max. Resolution": "1080p",
  "WiFi Connection": "Built-in WiFi",
  "Built-in Digital Tuner": "DVB-T \u2022 DVB-T2 \u2022 DVB-S2",
  "Smart TV Features": "Internet Browsing",
  "MPN": "KDL42W706BSU",
  "Product Line": "Bravia",
  "EAN": "4905524967739",
  "Name": "Sony Bravia KDL-42W706B 42\" 1080p HD LED LCD Internet TV",
  "Audio Type": "Surround",
  "Number Of Speakers": "2",
  "Refresh Rate": "100 Hz",
  "Viewing Angle": "89\u00b0",
  "Screen Size": "42\"",
  "Definition": "HDTV",
  "Display Technology": "LED LCD",
  "Display Resolution": "1920 x 1080 pixels",
  "Series": "W Series",
  "1000000": "1"
}
```

Quando Spexa fallisce

Shure SRH1540 Premium Closed-Back Headphones




SKU: SRH1540 MPN: SRH1540



The SRH1540 are Shure's new Premium Closed-Back headphones designed for studio use and audiophiles. Featuring an expansive soundstage for clear, extended highs and warm bass and developed with aluminum alloy and carbon fiber construction, the SRH1540s will provide years of pure listening enjoyment.

Save \$50 Now Until 6/30! Regularly \$499.00.

List Price: \$624.00
Sale Price: \$499.00
In Stock!

 **Add to Cart**
You can always remove it later

[view larger image](#)

 Like  Share 33 people like this. Be the first of your friends.

Product Description User Guides & Specifications Images

Shure SRH1540 Technical Specifications

Headphone Type: Closed-back, circumaural	Transducer Type: 40 mm Dynamic, Neodymium magnet
Frequency Range: 5 - 25,000 Hz	Sensitivity (@ 1 kHz): 99 dB/mW
Impedance (@ 1 kHz): 46 Ω	Cable Length/Type: 6 ft (1.83 m) dual-exit, detachable oxygen-free copper

Spexa: {}

Extractor:

```
{
  "Name": "Shure SRH1540 Premium Closed-Back Headphones",
  "Headphone Type": "Closed-back, circumaural",
  "Frequency Range": "5 - 25,000 Hz",
  "Sensitivity (@ 1 kHz)": "99 dB/mW",
  "Cable Length/Type": "6 ft (1.83 m) dual-exit, detachable oxygen-free copper",
  "Transducer Type": "40 mm Dynamic, Neodymium magnet",
  "Impedance (@ 1 kHz)": "46 \u03a9"
}
```

Quando Spexa sbaglia

American DJ HP550 Headphones

Black, comfortable, sturdy high powered headphones



Description

Reviews/Ratings

Features:

- Comfortable, sturdy high powered headphones
- Flexible housing design allows double or single ear monitoring; folds up for easy transport
- Replaceable earpads - No need to replace entire headphones when earpads wear and crack. Simply replace the earpad.
- Input: 3,5mm Mini plug with 1/4" adapter (straight and 90 Degrees)
- Includes Headphone bag and 2 1/4" jack plug adaptors (straight and 90 Degrees)
- Driver Unit: 57 mm diameter, Impedance: 64 ohm
- Sensitivity: 107dB (+/- 3dB) (S.P.L. at 1 KHz)
- Frequency Response: 50Hz to 30 KHz
- Magnet: NdFeB
- Plug: Standard 6,3 jack Stereo / Cord Length: 3 meters

Spexa:

```
{
  {
    "cart checkout my account login about us contact us": "800 562-6006",
    "cart": "checkoutmy accountloginabout uscontact us"
  }
}
```

Extractor:

```
{
  "Name": "American DJ HP550\u00a0Headphones",
  "Features": "",
  "Sensitivity": " 107dB (+/- 3dB) (S.P.L. at 1 KHz)",
  "Frequency Response": " 50Hz to 30 KHz",
  "Magnet": " NdFeB",
  "cart checkout my account login about us contact us": "cart",
  "cart": "checkout",
  "AV Products": "Systems Integration",
  "Driver Unit": " 57 mm diameter, Impedance: 64 ohm",
  "Plug": " Standard 6,3 jack Stereo / Cord Length: 3 meters",
  "Overall Rating": "",
  "Number of Reviews": "\u00a0",
  "E-mail this product page to a friend": "HP550 Headphones"
}
```