DTSA 5510 Unsupervised Algorithms in Machine Learning Final Project

Location of this project: https://github.com/NikoKuu/unsupervised-algorithms-in-ml-final

Project Topic

This work is the final project for DTSA 5510 Unsupervised Algorithms in Machine Learning course. The objective is to demonstrate how to use unsupervised learning methods, including data cleaning and eploratory data analysis (EDA).

A dataset of Amazon webstore item descriptions by Pavlo Mospan was selected. Kaggle is hosting the data set:

Amazon Advertisements, Pavlo Mospan 2019.

Available at: https://www.kaggle.com/datasets/sachsene/amazons-advertisements/data.

The plan is to clean up the data, perform basic Exploratory Data Analysis (EDA), vectorize the text descriptions of each item and use unsupervised learning methods to **categorize the data**. Non-negative Matrix Factorization and k-means are used as unsupervised learning algorithms. The actual categories of each item are known so the effectiveness of the unsupervised categorization can be gaged and supervised learning methods can also be used as a comparison to unsupervised.

Data

The data set consists of 525 csv-files (total of 280MB) in different levels of categories, totaling more than three million items with description. There are about 20 main categories and up to two additional nested categories. The main focus of this work is to cluster the items based on the description text mainly focusing on the most detailed category.

Some of the text descriptions are very short and frankly impossible to categorize, such as a one-word text of the product part number. The text data also has a lot of numerical values for product dimensions or package quantity. A few methods to handle the numerical values were experimented.

Each csv-file has only column named 'ad' and the first row is the header. Rest of the rows are item description text each item in the category. Category name is extracted from the filename. These files have csv file extensions but they should not treated such since some of the text have commas in them.

Load most of the required modules

In [1]:
```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
import os
import re
import string
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.decomposition import NMF
from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay, accuracy_scor
%matplotlib inline
```

Note on the number of categories

Accuracy measurement will be conducted to verify the performance of the model(s). However, large number of categories makes label permutation to find optimal labels infeasible.

$permutations = \frac{n!}{(n-r)!}$, where n is the total number of elements and r is the number of elements chosen.

In this work, $n = r$, and if we had $n = 20$, we would have $20! \approx 2.43e18$ permutations.

Data Cleaning

The default categorization is cleaned by making the following changes and corrections:

- Move items in "home kitchen" category to main category "home" and subcategory "kitchen".
  - There already were some items in the home/kitchen -category.
- Move items in main category industrial/scientific category to common main category "industrial scientific".
  - There already were items in the "industrial scientific" -category.
- "Computers" moved under "electronics" main category
- "Smart home" moved under "electronics" main category
- Fix typo in 'sports'
  - No effect on the categories

Also, measurement units (such as inch, mm and pack), digits and puncuation are removed. The code below also does additional cleaning however some of it is not necessary since the TFIDF vectorizer can do it.

Loading the data

In [2]:
```
class amz_data:
    def __init__(self):
```

```
        self.id = 0
        self.df = pd.DataFrame()
        self.df_cat = pd.DataFrame(columns=['full', 'main', 'sub', 'subsub', 'botto
        self.placeholder = ''#'<NUM>'

    def read_data(self, basedir):
         # https://docs.python.org/3.9/library/os.html?highlight=os%20walk
        for root, dirs, files in os.walk(basedir):
            #print(files)
            for file in files:
                path = os.path.join(root, file)
                self.add_to_dataframe(path)

    def add_to_dataframe(self, path):
        # print(path)
        # Too long path name. Needed to add the prefix for long path names.
        with open(u'\\\\?\\' + os.path.abspath(path), 'r', encoding="utf8") as f:
            # lines = f.readlines()
            lines = f.read().splitlines() # without the \n in the end
        df = pd.DataFrame(lines[1:], columns=['description'])
        df['id'] = self.id # Assign a unique id
        df['word_count'] = df.apply(self.count_words, axis=1)
        df['description'] = df.apply(self.preprocess_text, axis=1)
        categories = list(self.get_category(os.path.basename(path)))
        df['full'] = categories[0]
        df['main'] =  categories[1]
        df['sub'] =  categories[2]
        df['subsub'] =  categories[3]
        df['bottom'] =  categories[4]
        self.df = pd.concat([self.df, df], ignore_index=True)
        # self.df_cat = pd.concat([self.df_cat, [self.id, self.get_category(os.path
        self.df_cat.loc[len(self.df_cat)] = list(categories)
        self.id = self.id + 1

    def get_category(self, file_name):
        if 'home-kitchen' in file_name: # Home kitchen -exception
            file_name = file_name.replace('home-kitchen', 'home_kitchen')
        if 'industrial_scientific' in file_name: # Industrial scientific -exception
            file_name = file_name.replace('industrial_scientific-tests-measurements
            file_name = file_name.replace('industrial_scientific', 'industrial-scie
        if '_computers' in file_name: # Computers -exception
            file_name = file_name.replace('_computers', '_electronics_computers')
        if '_smart-home' in file_name: # Smart home -exception
            file_name = file_name.replace('_smart-home', '_electronics_smart-home')
        if 'fan-shop' in file_name: # Fan shop -correction
            file_name = file_name.replace('fan-shop', 'sports-outdoors_fan-shop')
        if 'aports' in file_name: # aports -correction
            file_name = file_name.replace('aports', 'sports')
        category = file_name.split('.')[0]
        cat_list = category.split('_')
        main = cat_list[1]
        if len(cat_list) >= 4:
            sub = cat_list[2]
        else:
            sub = None
        if len(cat_list) >= 5:
```

```
            subsub = cat_list[3]
        else:
            subsub = None
        bottom = cat_list[-1]
        return category.replace('amazon_',''), main, sub, subsub, bottom

    def count_words(self, row):
        words = len(row['description'].split(' '))
        return words

    def replace_numbers(self, row):
        text = re.sub(r'\d+', self.placeholder, row['description'])
        return text

    def preprocess_text(self, row):
        # Remove not helpful words
        text = row['description']
        text = text.lower().replace('mm','')
        text = text.replace('inches','')
        text = text.replace('inch','')
        text = text.replace('pack','')
        text = text.replace('pcs','')
        text = text.replace('pieces','')
        text = text.strip()
        pattern = r'\b[a-zA-Z]\b' # Replace single-character words
        text = re.sub(pattern, '', text)
        translator = str.maketrans('', '', string.punctuation + string.digits)
        return text.translate(translator)
```

Option 1. Load the data from local disk

In [3]:
```
amz = amz_data()
amz.read_data(r'..\Amazon Ads\data')
```

Option 2. Load the data from Kaggle

In [ ]:
```
import kagglehub

# Download latest version
path = kagglehub.dataset_download("sachsene/amazons-advertisements")

amz = amz_data()
amz.read_data(os.path.join(path, 'scrapped_data', 'scrapped_data'))
```

Dataset info

General info and four first and last entries are printed out below to show the general format of the data.

The dataframe has over three million rows. However, it was found that there are over 300,000 duplicated entries (identical rows). These duplicates had the same category. In addition to that, there are 600,000 duplicate descriptions with different category. Since the

models are trying to categorize each item into one category, these multicategory entries are also removed. Total of ~950,000 items were removed as duplicates.

In [4]:
```
print(amz.df.info(),'\n')
print(amz.df.head(4),'\n')
print(amz.df.tail(4),'\n')
print('\nNumber of nan values:\n', np.sum(amz.df.isna(), axis=0))
print('\nNumber of duplicated entries:', np.sum(amz.df.duplicated()))
print('Number of duplicated items:', np.sum(amz.df.duplicated(subset='description')
print('Drop duplicates...')
amz.df.drop_duplicates(inplace=True, subset='description')
print('Number of duplicated items:', np.sum(amz.df.duplicated(subset='description')
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 3318260 entries, 0 to 3318259
Data columns (total 8 columns):
 #   Column       Dtype 
---  ------       ----- 
 0   description  object
 1   id           int64 
 2   word_count   int64 
 3   full         object
 4   main         object
 5   sub          object
 6   subsub       object
 7   bottom       object
dtypes: int64(2), object(6)
memory usage: 202.5+ MB
None
```

	description	id	word_count	full	main	sub	subsub	bottom
0	hisense k ultra hd smart led tv he	0	10	electronics_smart-home	electronics	None	None	smart-home
1	vizio k ultra hd smart led tv pe	0	11	electronics_smart-home	electronics	None	None	smart-home
2	sony xbrxeseries class hdr uhd smart led tv	0	8	electronics_smart-home	electronics	None	None	smart-home
3	tivo bolt vox tb dvr streaming media player k...	0	17	electronics_smart-home	electronics	None	None	smart-home

	description	id	word_count	full	main	sub	subsub	bottom
3318256	whimsical watches women t schnauzer black leat...	524	11	women-fashion_watches_wrist	women-fashion	watches	None	wrist
3318257	whimsical watches women t saint bernard black ...	524	12	women-fashion_watches_wrist	women-fashion	watches	None	wrist
3318258	whimsical watches women n schnauzer black leat...	524	11	women-fashion_watches_wrist	women-fashion	watches	None	wrist
3318259	whimsical watches women t pixie bob cat black ...	524	13	women-fashion_watches_wrist	women-fashion	watches	None	wrist

```
Number of nan values:
 description         0
id                   0
word_count           0
full                 0
main                 0
sub             956581
subsub         3116914
bottom               0
dtype: int64

Number of duplicated entries: 356103
Number of duplicated items: 951386
```

```
Drop duplicates...
Number of duplicated items: 0
```

Total of 439 item level categories were found and those are grouped into 17 main categories.

In [5]:
```
print(amz.df_cat.info(), '\n')
print('Any duplicated categories:', np.any(amz.df_cat.duplicated()), '\n')
print('Number of full categories:', len(pd.unique(amz.df_cat['full'])))
print('Number of main categories:', len(pd.unique(amz.df_cat['main'])))
print('Number of sub-categories:', len(pd.unique(amz.df_cat['sub'])))
print('Number of lower sub-categories:', len(pd.unique(amz.df_cat['subsub'])))
print('Number of most detailed categories:', len(pd.unique(amz.df_cat['bottom'])))
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 525 entries, 0 to 524
Data columns (total 5 columns):
 #   Column  Non-Null Count  Dtype 
---  ------  --------------  ----- 
 0   full    525 non-null    object
 1   main    525 non-null    object
 2   sub     390 non-null    object
 3   subsub  48 non-null     object
 4   bottom  525 non-null    object
dtypes: object(5)
memory usage: 24.6+ KB
None

Any duplicated categories: False

Number of full categories: 525
Number of main categories: 17
Number of sub-categories: 29
Number of lower sub-categories: 12
Number of most detailed categories: 439
```

Exploratory Data Analysis

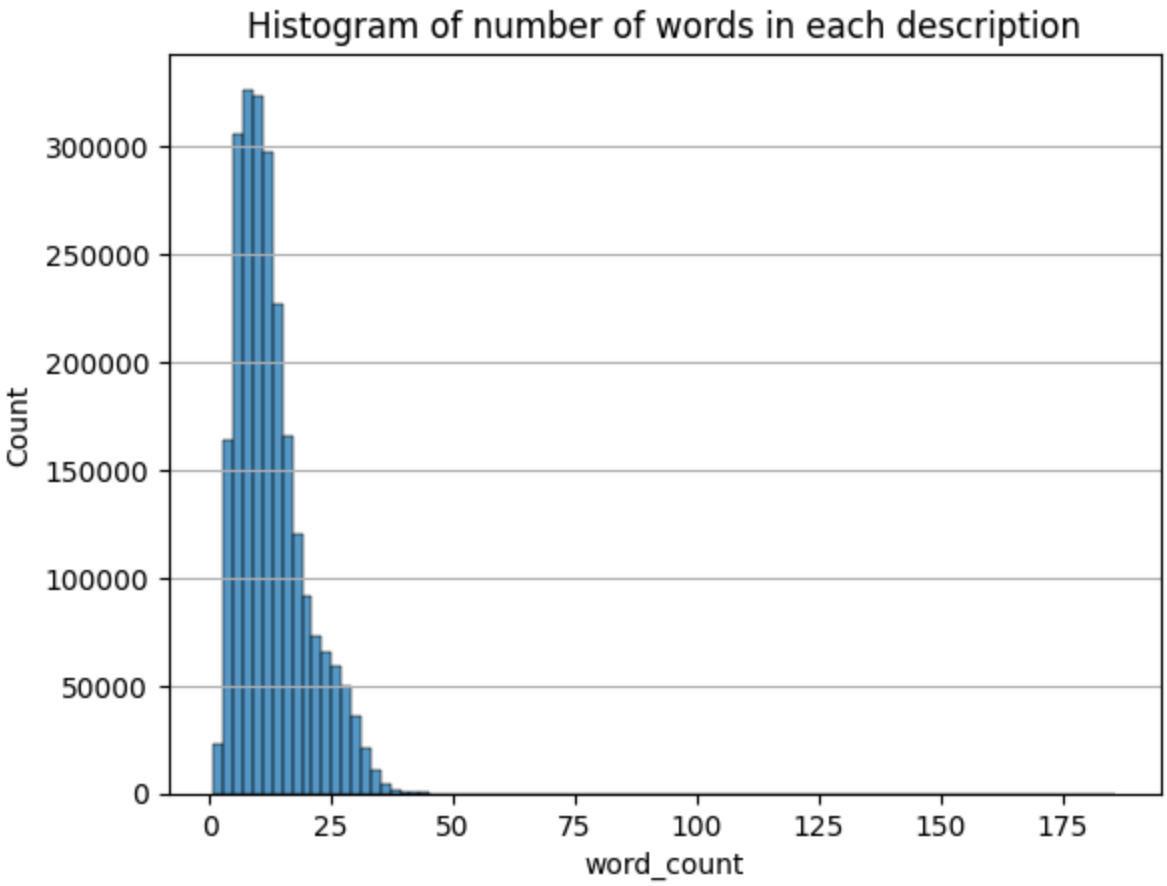
Below a histogram of approximate word counts is plotted. The distribution looks reasonable.

Also the shortest and the longest desriptions are found showing that the shortest has only one word and the longest has 186 words in them. Mean number of words is 13 with standard deviation of 7.

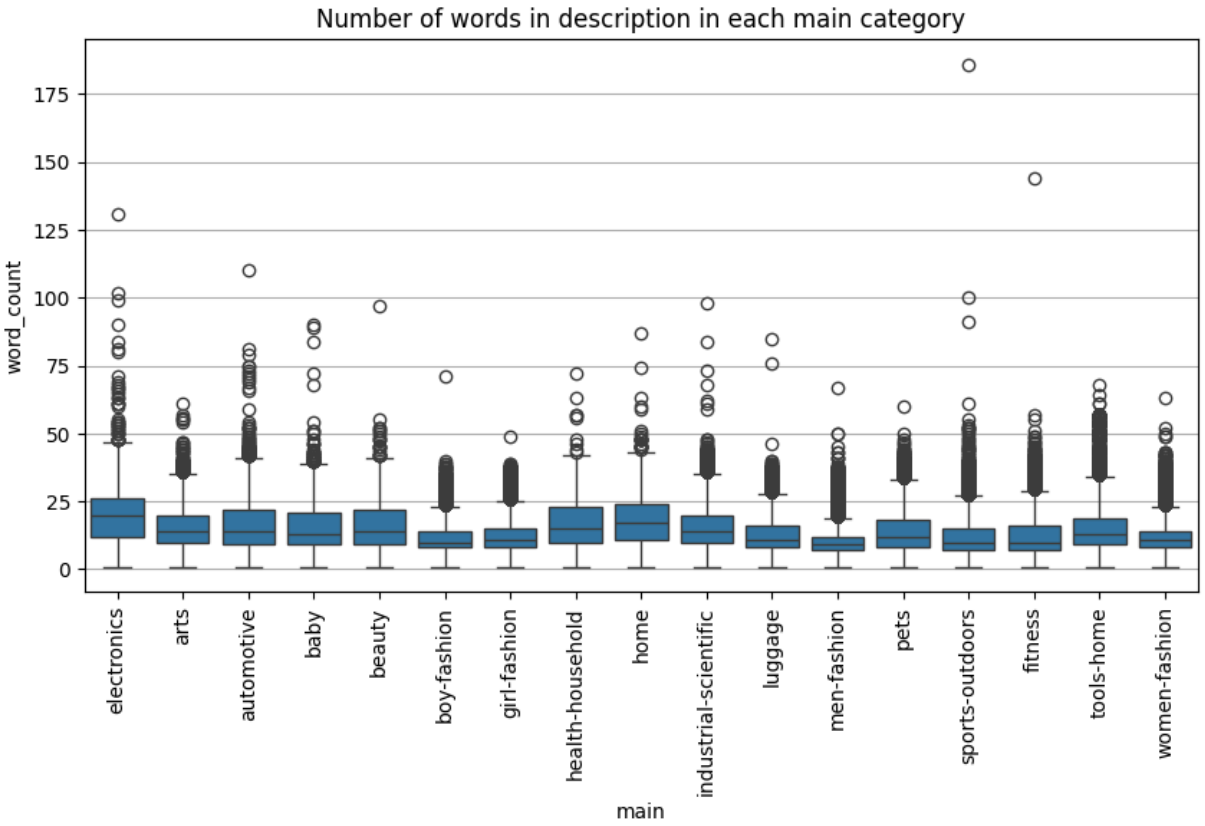
Lastly, the box plot shows how many words each main category has. There does not seem to be significant difference in the description length between the main categories.

In [17]:
```
sns.histplot(amz.df, x='word_count', binwidth=2)
plt.grid(axis='y')
plt.title('Histogram of number of words in each description')
plt.show()
print('Maximum number of words:', max(amz.df['word_count']))
print('Minimum number of words:', min(amz.df['word_count']))
```

```
print('Mean number of words:', round(np.mean(amz.df['word_count'])))
print('Standard deviation of number of words:', round(np.std(amz.df['word_count'])))
print('Entry with the minimum amount of words:\n', amz.df.iloc[np.argmin(amz.df['wo

fig = plt.figure(figsize=(10, 5))
sns.boxplot(amz.df, x='main', y='word_count')
plt.xticks(rotation=90, fontsize = 10)
plt.title('Number of words in description in each main category')
plt.grid(axis='y')
plt.show()
```




```
Maximum number of words: 186
Minimum number of words: 1
Mean number of words: 13
Standard deviation of number of words: 7

Entry with the minimum amount of words:
 description                   bxppwjz
id                                  0
word_count                          1
full           electronics_smart-home
main                       electronics
sub                               None
subsub                            None
bottom                     smart-home
Name: 799, dtype: object
```



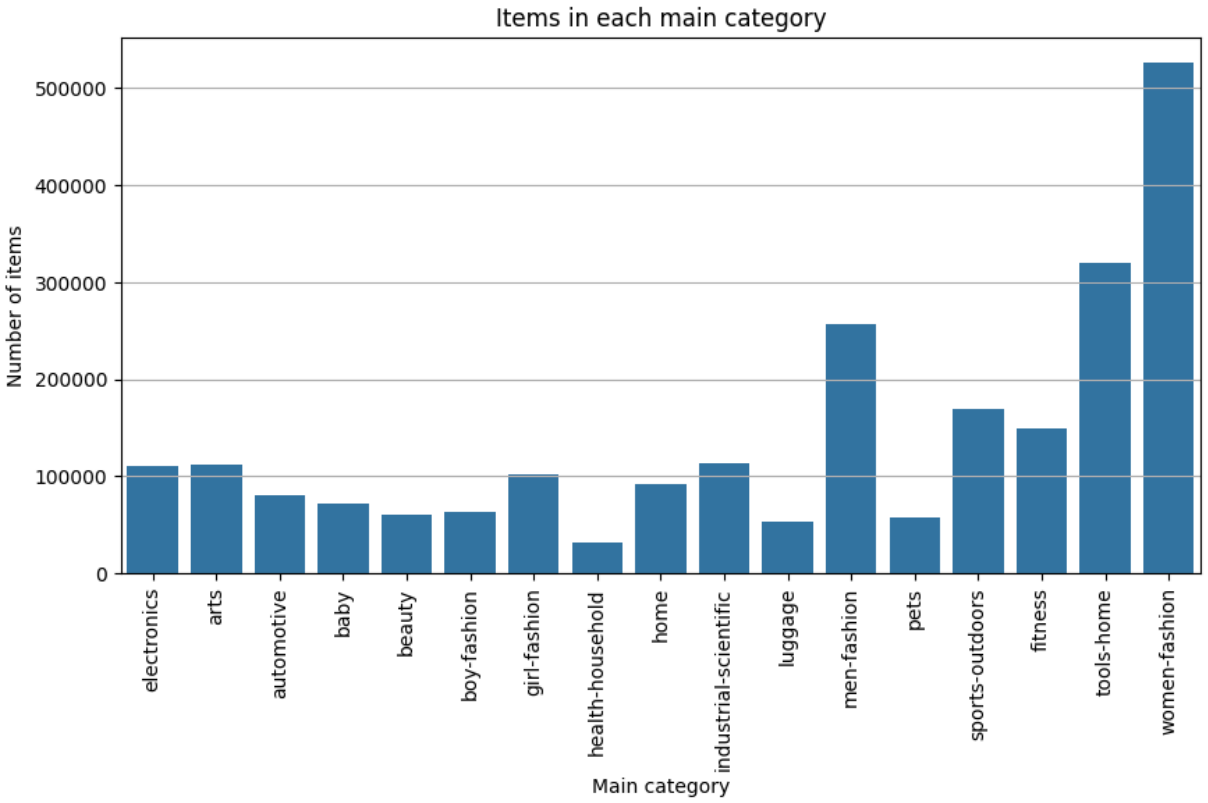
Below, first the number of items in each main category is plotted. Then then number of items in each detailed category is plotted.

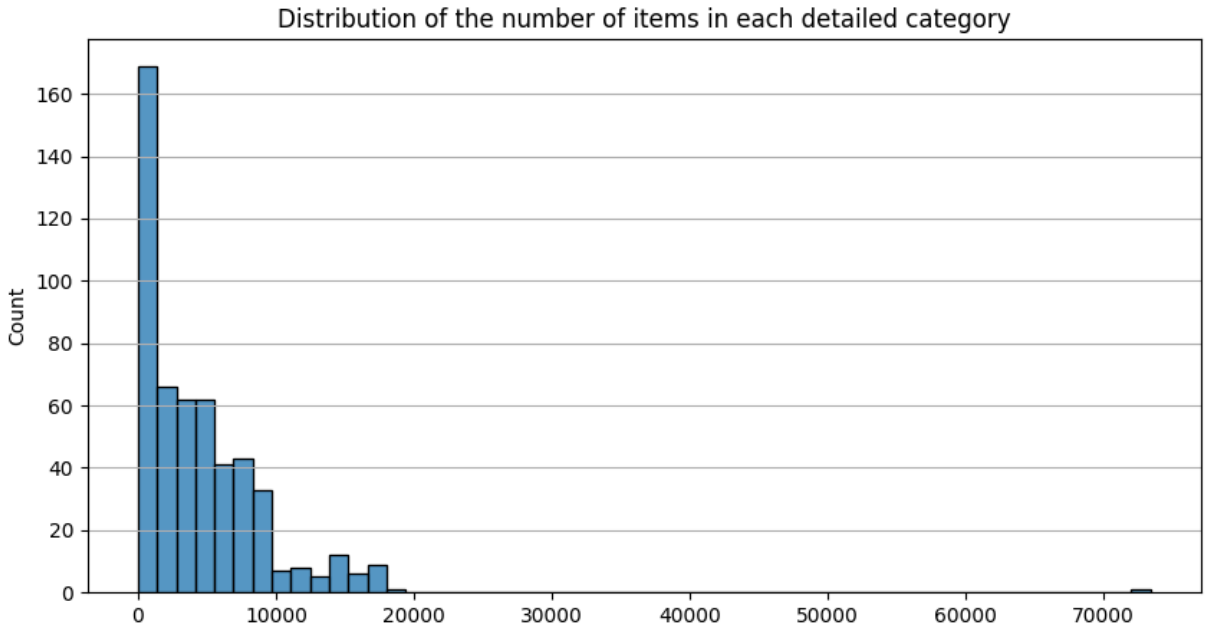
Women's fashion is the largest main category with over 500,000 items and health household has the least amount of items.

In detailed categories, fan shop is extremely large category (73,000 items) compared to the others and some categories have only 6 items. Learning from very small categories may not work well.

In [38]:
```
fig = plt.figure(figsize=(10, 5))
sns.countplot(amz.df, x='main', stat='count')
plt.title('Items in each main category')
plt.xlabel('Main category')
plt.ylabel('Number of items')
plt.xticks(rotation=90, fontsize = 10)
plt.grid(axis='y')
plt.show()

fig = plt.figure(figsize=(10, 5))
sizes = amz.df.groupby('full').size()
sns.histplot(sizes)
plt.title('Distribution of the number of items in each detailed category')
plt.grid(axis='y')
plt.show()

print('Maximum number of items:', max(sizes))
print('Minimum number of items:', min(sizes))
```

```
print('Mean number of items:', round(np.mean(sizes)))
print('Standard deviation of number of items:', round(np.std(sizes)), '\n')

print('Item count of the five largest categories:\n', sizes.sort_values(ascending=F
print('Item count of the five smallest categories:\n', sizes.sort_values(ascending=
```





```
Maximum number of items: 73372
Minimum number of items: 6
Mean number of items: 4508
Standard deviation of number of items: 5090

Item count of the five largest categories:
 full
sports-outdoors_fan-shop         73372
girl-fashion_jewelry_earrings    18113
women-fashion_jewelry_body       17916
men-fashion_access_caps-hats     17700
luggage_backpacks                17240
dtype: int64

Item count of the five smallest categories:
 full
sports-outdoors_recreation_water-sports_clothing     6
fitness_running_gps                                  6
tools-home_appliances_dishwasher                    14
tools-home_appliances_freezers                      27
boy-fashion_jewelry_tie-clips                       30
dtype: int64
```

Tokenization

TF-IDF is used to convert the text data to numerical representation in matrix format.

Acronym TF-IDF comes from Term Frequency, Inverse Document Frequency. The first part, Term Frequency, tells how many times a certain word is found in the text. The second part, inverse of Document Frequency, divides the first part by how many documents contain the word. The division lowers the score for common words that appear often, like 'the', 'of', etc.

Bag of words and TF-IDF should be good choices for the text in descriptions since they usually are just a list of keywords that do not necessary form meaningful sentences.

Below a few categories (for example 7) are randomly chosen for the models. There are two ways to choose the data. Either by the main category or by detailed category. Based on testing, categorizing main categories is challenging, probably due to wide range of items in each main category. Clustering the items into detailed categories is shown in this work.

Vastly different amount of items in different categories caused extracted keywords to spill into wrong categories. For example, in one case having a modeling set including women's fashion with 20x items than the smallest category caused the word 'leggings' to spill into three extracted categories (tennis gear and industrial equipment). For that reason, the code below picks equal amount of items from each category. NMF and K-means clustering should be able to tolerate differences in category sizes but over 10x difference seemed to be too much.

In [21]:
```
import random
number_of_categories = 7
```

```
main_category = False
mask = []
counts_train = []
if main_category:
    chosen_categories = random.sample(list(pd.unique(amz.df_cat['main'])), number_o
    for index, cat in enumerate(chosen_categories):
        m = amz.df['main'] == cat
        counts_train.append(sum(m))
        mask.append(m)
    amount = min(counts_train)
    idx_all = []
    for index, cat in enumerate(chosen_categories):
        m = amz.df['main'] == cat
        idx_all.append(np.where(m)[0][0:amount])

    idx_all = np.array(idx_all).flatten()
    X_train = amz.df['description'].iloc[idx_all]
    y_train = amz.df['main'].iloc[idx_all]
else:
    chosen_categories = amz.df_cat.sample(number_of_categories, random_state=25)
    for index, cat in chosen_categories.iterrows():
        m = amz.df['full'] == cat['full']
        counts_train.append(sum(m))
        mask.append(m)
    amount = min(counts_train)
    idx_all = []
    for index, cat in chosen_categories.iterrows():
        m = amz.df['full'] == cat['full']
        idx_all.append(np.where(m)[0][0:amount])

    idx_all = np.array(idx_all).flatten()
    X_train = amz.df['description'].iloc[idx_all]
    y_train = amz.df['full'].iloc[idx_all]

print(X_train.info())
print(y_train.info())
y_train = np.array([s.replace('-',' ').replace('_',' - ') for s in y_train])
categories_train = pd.unique(y_train)
# categories_train = [s.replace('-',' ').replace('_',' - ') for s in categories_tra
print('\n' + '\n'.join([cat+': '+str(cnt) for cat, cnt in zip(categories_train, cou
```

```
<class 'pandas.core.series.Series'>
Index: 3143 entries, 2268936 to 681415
Series name: description
Non-Null Count  Dtype
--------------  -----
3143 non-null   object
dtypes: object(1)
memory usage: 49.1+ KB
None
<class 'pandas.core.series.Series'>
Index: 3143 entries, 2268936 to 681415
Series name: full
Non-Null Count  Dtype
--------------  -----
3143 non-null   object
dtypes: object(1)
memory usage: 49.1+ KB
None

tools home - hardware - adhesives sealers: 1165
sports outdoors - recreation - skates: 6902
boy fashion - jewelry - rings: 1071
men fashion - access - wallets cards: 10081
electronics - computers - scanners: 671
tools home - appliances - range hoods: 449
girl fashion - clothing - sweater: 972
```

Based on testing a few TFID parameters (some of them shown in the code below), the following parameters were chosen:

- Sublinear Term Frequency (TF) scaling
- Ignore terms that have a document frequency strictly higher 95%
- Ignore terms that have a document frequency strictly lower 5 words
- L2 norm
- Latin-1 encoding and English stop words
- Only single words considered

In [39]:
```
doe = 2
if doe == 1:
    my_vectorizer = TfidfVectorizer(sublinear_tf=True, max_df=0.95, min_df=5,
                                    norm='l2', encoding='latin-1', ngram_range=(1,
                                    stop_words="english", max_features=10000)
elif doe == 2:
    my_vectorizer = TfidfVectorizer(sublinear_tf=True, max_df=0.95, min_df=5,
                                    norm='l2', encoding='latin-1', ngram_range=(1,
                                    stop_words="english", max_features=10000, strip
elif doe == 3:
    my_vectorizer = TfidfVectorizer(sublinear_tf=True, max_df=0.95, min_df=2,
                                    norm='l2', encoding='latin-1', ngram_range=(1,
                                    stop_words="english", max_features=10000)

mat = my_vectorizer.fit_transform(X_train)
# mat.shape
```

Inspecting the feature names printed below, shows that the vectorizer does well extracting the key words. All the words are proper english words, there is no punctuation, number, partial words, etc.

In [23]:
```
print('Shape of the matrix:', mat.shape)
ftr_names = my_vectorizer.get_feature_names_out()
print('Length of ftr_names:',len(ftr_names))
print('Sample of the feature names:\n', ftr_names[0:100])
```

```
Shape of the matrix: (3143, 1021)
Length of ftr_names: 1021
Sample of the feature names:
 ['abec' 'ac' 'accessories' 'accessory' 'acchen' 'acrylic' 'acsuss' 'ada'
 'adapter' 'adapters' 'adf' 'adhesive' 'adjustable' 'adult' 'adults'
 'advantage' 'age' 'ages' 'aging' 'air' 'akdy' 'alexandrite' 'allinone'
 'alpine' 'aluminium' 'aluminum' 'amazon' 'amber' 'ambir' 'american' 'amy'
 'anchor' 'ancona' 'animal' 'anniversary' 'anr' 'anti' 'antique'
 'applicator' 'aproca' 'aquamarine' 'armor' 'art' 'assembly' 'assn'
 'authentic' 'auto' 'automatic' 'av' 'available' 'baby' 'baffle' 'bag'
 'ballet' 'bamboo' 'band' 'bands' 'bar' 'barefoot' 'basic' 'bath'
 'bathroom' 'battery' 'bay' 'bb' 'beaded' 'bearings' 'beginner'
 'beginners' 'beige' 'bellroy' 'best' 'beveled' 'bicycle' 'bifold' 'big'
 'bike' 'biker' 'biking' 'billfold' 'billy' 'birthstone' 'bisque' 'bit'
 'black' 'blackbutterfly' 'blades' 'blizzard' 'blocking' 'blower' 'blue'
 'bmx' 'board' 'boards' 'boboyoyo' 'body' 'bolero' 'bones' 'bonny' 'book']
```

Models

In this chapter, the learning models are built. First some of the helper functions are defined. The the non-Negative Matrix Factorization model is built and tested. Then k-Means model and lastly the supervised models for benchmarking.

Helper functions

In [24]:
```
from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay, accuracy_scor

class calc_metrics:
    def __init__(self, y_test, y_pred):
        self.confusion_matrix = None
        self.recall = None
        self.f1_score = None
        self.precision = None
        self.recall = None
        self.FNR = None

        self.get_metrics(y_test, y_pred)

    def get_metrics(self, y_test, y_pred):
        conf_mat = confusion_matrix(y_test, y_pred)
        self.confusion_matrix = pd.DataFrame(conf_mat)
        self.accuracy = accuracy_score(y_test, y_pred)
        print('Accuracy:', round(self.accuracy, 3))
```

```
        print('Confusion matrix:')
        disp = ConfusionMatrixDisplay(conf_mat)
        disp.plot()
        plt.show()
```

Non-negative Matrix Factorization

A few sets of parameters for NMF were evaluated and the following parameters gave the best performance. However, the differences were small, in order of a few %-points in accuracy.

- Multiplicative Update solver
- Nndsvda (NNDSVD with zeros filled with the average of X) initializion procedure
- Beta loss: kullback-leibler
- Alpha W/H: 0.00005

In [40]:
```
doe = 2
if doe == 1:
    myNMF = NMF(n_components=number_of_categories, solver='cd', beta_loss='frobeniu
                alpha_W=0.0001, alpha_H='same')
elif doe == 2:
    myNMF = NMF(n_components=number_of_categories, random_state=123, init='nndsvda'
                solver='mu', beta_loss="kullback-leibler", alpha_W=0.00005, alpha_H
elif doe == 3:
    myNMF = NMF(n_components=number_of_categories)
elif doe == 4:
    myNMF = NMF(n_components=number_of_categories, solver='cd', init='nndsvd',
                beta_loss='frobenius',alpha_W=0.0001, alpha_H='same')

W = myNMF.fit_transform(mat)
H = myNMF.components_
```

A few checks on the W and H matrices in the code below.

The 'short' dimension of both matrices matches the number of categories. The H matrix scores the most prevalent words and the W matrix scores the category for each item.

In [26]:
```
print('Shape of the matrix:', mat.shape)
print('Shape of W:', W.shape)
print('The first row of W:', W[0,:])
print('Shape of H:', H.shape)
print('The first column of H:', H[:,0])
```

```
Shape of the matrix: (3143, 1021)
Shape of W: (3143, 7)
The first row of W: [1.81450913e-65 0.00000000e+00 0.00000000e+00 0.00000000e+00
 0.00000000e+00 9.28222247e-02 0.00000000e+00]
Shape of H: (7, 1021)
The first column of H: [0.         0.         0.         0.         0.         0.
 0.07174864]
```

In [27]:
```
# Adapted from:
# https://scikit-learn.org/stable/auto_examples/applications/plot_topics_extraction
```

```
def plot_top_words(H, feature_names, n_top_words):
    print('Top words in the H-matrix:')
    for topic_idx, topic in enumerate(H):
        top_features_ind = topic.argsort()[-n_top_words:]
        top_features = feature_names[top_features_ind]
        print(str(topic_idx) + ':', top_features)

print('Categories in the data set:\n' + '\n'.join(categories_train) + '\n')
plot_top_words(H, ftr_names, 10)
```

```
Categories in the data set:
tools home - hardware - adhesives sealers
sports outdoors - recreation - skates
boy fashion - jewelry - rings
men fashion - access - wallets cards
electronics - computers - scanners
tools home - appliances - range hoods
girl fashion - clothing - sweater

Top words in the H-matrix:
0: ['pocket' 'holder' 'bifold' 'blocking' 'card' 'slim' 'leather' 'rfid'
 'men' 'wallet']
1: ['kitchen' 'ancona' 'cabinet' 'mount' 'wall' 'cfm' 'steel' 'stainless'
 'range' 'hood']
2: ['pullover' 'shrug' 'little' 'knit' 'big' 'long' 'sleeve' 'cardigan'
 'sweater' 'girls']
3: ['stainless' 'steel' 'jewelry' 'wedding' 'band' 'rings' 'sterling' 'size'
 'silver' 'ring']
4: ['slide' 'duplex' 'digital' 'photo' 'film' 'fujitsu' 'color' 'portable'
 'document' 'scanner']
5: ['glue' 'grout' 'rubber' 'white' 'roll' 'masking' 'tesa' 'adhesive'
 'black' 'tape']
6: ['complete' 'inline' 'adjustable' 'kids' 'skates' 'wheels' 'skate'
 'roller' 'skateboard' 'scooter']
```

Looking at the lists of top the top words, it should not be hard to figure out the category for each index. However, this work can be done programatically. The code below permutes all index combinations to find the best label order based on the accuracy metric.

Like mentioned in the beginning, the complexity of this method increases sharply with increasing number of categories. Only up to 10 categories is a reasonble number for this method.

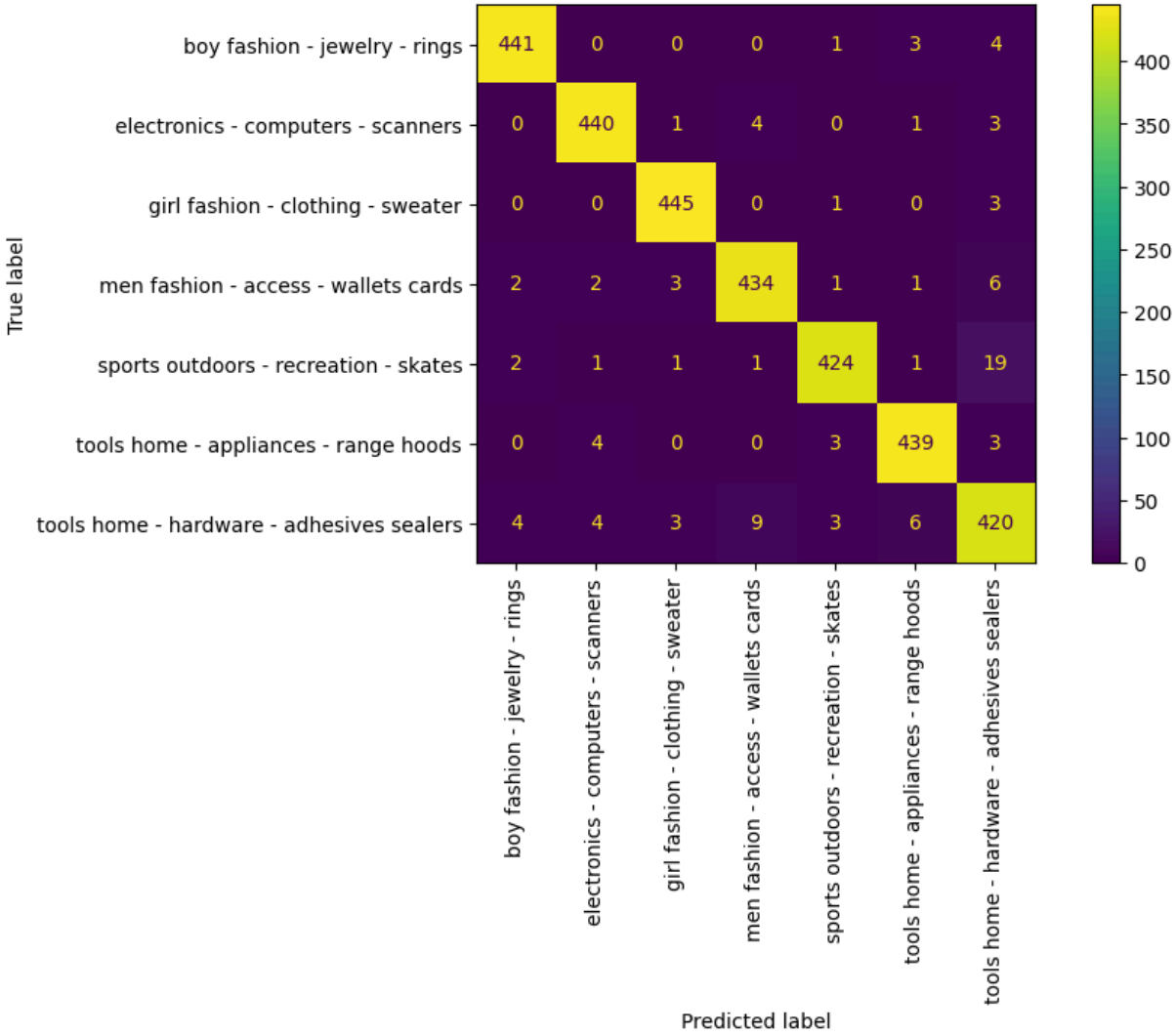
In [28]:
```
# The label_permute_compare -function below was adapted from my Week 2 peer review.
import itertools
from sklearn.metrics import ConfusionMatrixDisplay
import random


def label_permute_compare(ytdf,yp,subset=1):
    # your code here
    if subset > 1:
        indices = random.sample(range(len(yp)), int(subset * len(yp)))
        ypp = yp[indices]
```

```
        ytdfp = ytdf.iloc[indices]
    else:
        ypp = yp
        ytdfp = ytdf
    labels = pd.unique(ytdf)
    best_accuracy = 0
    best_labelorder = []
    # ytdf: from str to int
    labels_lst = list(labels)
    yt_vals = np.array([labels_lst.index(yt) for yt in list(ytdfp)])
    i = 0
    for labelorder in itertools.permutations(np.unique(ypp)):
        yp_labels = [labelorder[y] for y in ypp]
        accuracy = sum(yt_vals == yp_labels)
        i=i+1
        if accuracy > best_accuracy:
            # print(i, ' - ', best_accuracy / len(yp_labels), end='\r')
            best_accuracy = accuracy
            best_labelorder = labelorder
    # print()
    best_accuracy = best_accuracy / len(yp_labels)
    return best_labelorder, best_accuracy

def reorder_prediction(yp, label_order):
    # Remap the predicted labels
    ypp = [label_order[y] for y in yp]
    return ypp

def inspect(item, yp, X_train, y_train):
    print('Index:', item)
    print('Predicted:\t', categories_train[yp[item:item+1]][0])
    print('Actual:\t\t', y_train[item])
    print('Item description:', X_train.iloc[item])
```

As can be seen below, the model has some trouble categorizing adhesive sealer items. Otherwise it does a decent job.

In [29]:
```
yp = np.argmax(W,axis=1)
print('yprediction length:',len(yp))
label_order, best_accu = label_permute_compare(y_train, yp, 1)
print('\nLabel order:', ' --> '.join([str(lbl) for lbl in label_order]))
print('\nAccuracy:', round(best_accu,4))

ypp = reorder_prediction(yp, label_order)
yp_cat = [categories_train[y] for y in ypp]
fig, ax = plt.subplots(figsize=(10, 5))
ConfusionMatrixDisplay.from_predictions(y_train, [categories_train[y] for y in ypp]
plt.show()
```

```
yprediction length: 3143

Label order: 3 --> 5 --> 6 --> 2 --> 4 --> 0 --> 1

Accuracy: 0.9682
```

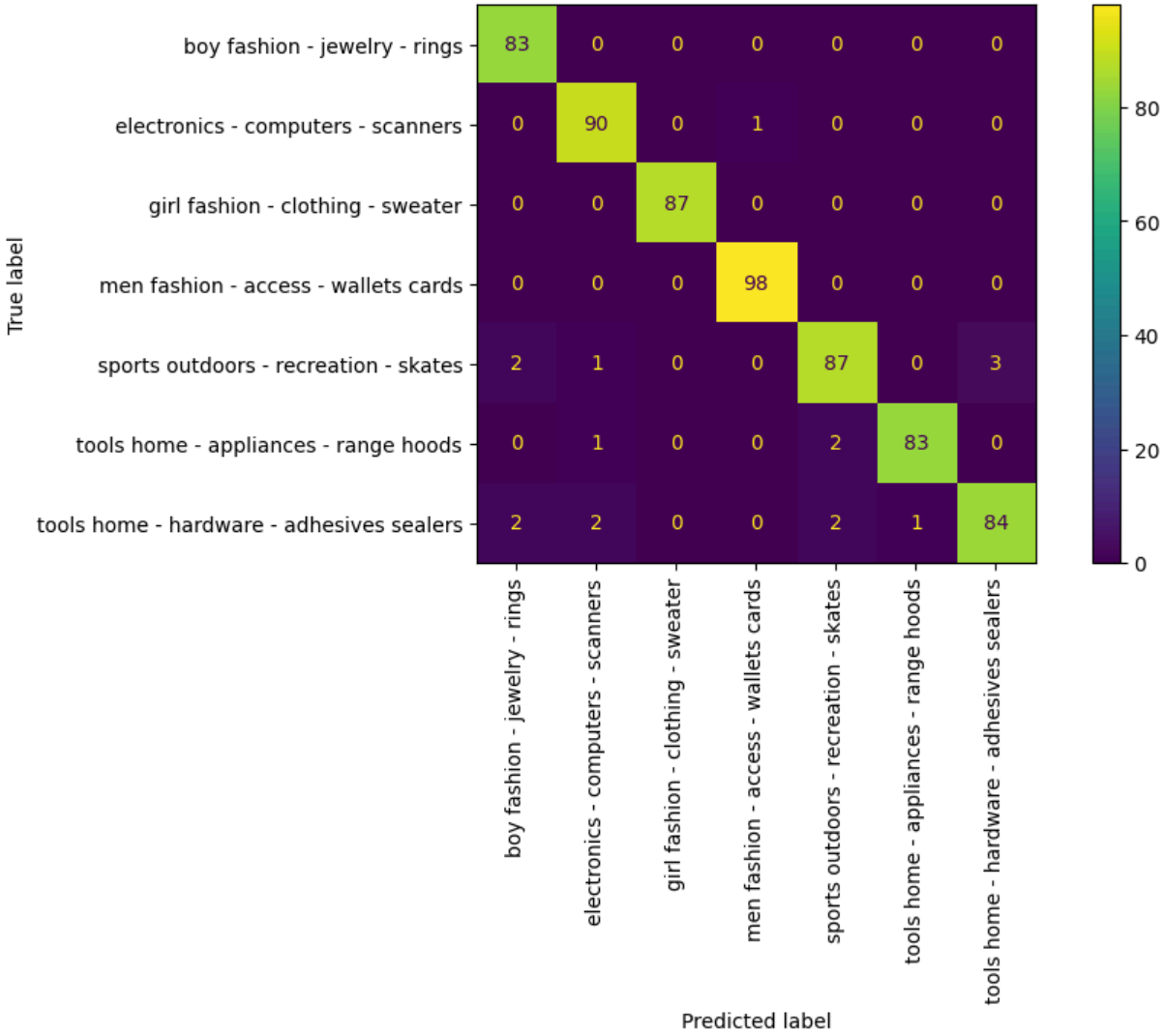


Further improve label mapping by using supervised learning

Supervised learning can be used for label assigment instead of picking the largest value in the W-matrix. Accuracy is slightly better with the Support Vector Machine model used below.

In [30]:
```
from sklearn.model_selection import train_test_split
from sklearn import svm

W_train_svm, W_test_svm, y_train_svm, y_test_svm = train_test_split(W, y_train, tes

clf = svm.SVC(decision_function_shape='ovo')
clf.fit(W_train_svm, y_train_svm)
# print(W_train_svm.shape)
yhat_test_svm = clf.predict(W_test_svm)
print('Accuracy:', round(accuracy_score(y_test_svm, yhat_test_svm),3))

fig, ax = plt.subplots(figsize=(10, 5))
ConfusionMatrixDisplay.from_predictions(y_test_svm, yhat_test_svm, ax=ax, xticks_ro
plt.show()
```

```
Accuracy: 0.973
```



Inspecting a few samples from the incorrect predictions reveals that they are not necessary the most clear descriptions but most of humans should not have trouble categorizing them correctly.

In [31]:
```
# Check
for item in random.sample(list(np.where(yp_cat != y_train)[0]), 4):
    print('Wrong prediction:')
    inspect(item, ypp, X_train, y_train)
    print()
```

```
Wrong prediction:
Index: 446
Predicted:       boy fashion - jewelry - rings
Actual:          tools home - hardware - adhesives sealers
Item description: nonskid traction strips set of  color while

Wrong prediction:
Index: 2167
Predicted:       tools home - appliances - range hoods
Actual:          electronics - computers - scanners
Item description: phonepc  turn your smartphone into  remote control for pcmac

Wrong prediction:
Index: 1684
Predicted:       boy fashion - jewelry - rings
Actual:          men fashion - access - wallets cards
Item description: toughergun wife to husband father mother to son gift best annivers
ary christmas birthday gifts slim wallet

Wrong prediction:
Index: 367
Predicted:       sports outdoors - recreation - skates
Actual:          tools home - hardware - adhesives sealers
Item description: zing  lean at work sign can lot size be reduced hx w recycled plas
tic
```

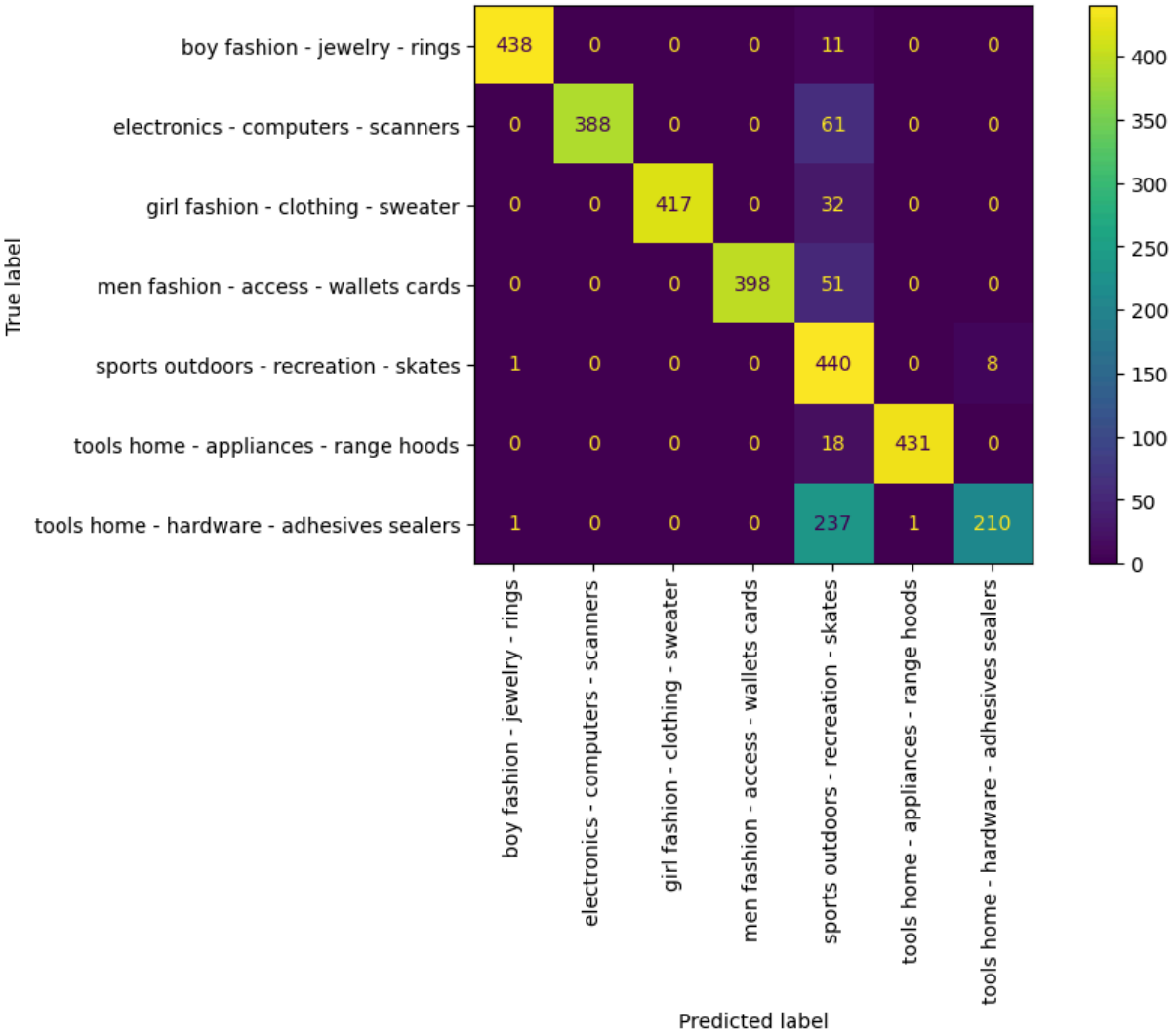
K-Means

The number of clusters is known, 7 product categories selected.

The model has trouble also with adhesive sealers and seems to be biased towards skates category.

In [32]:
```
from sklearn.cluster import KMeans
my_kmeans = KMeans(n_clusters=number_of_categories, random_state=12, init='random',
# my_kmeans = KMeans(n_clusters=number_of_categories, random_state=124).fit(mat)
print('Values:', pd.unique(my_kmeans.labels_))
print('Length of labels:', len(my_kmeans.labels_))
print('Five first labels:', my_kmeans.labels_[0:6])

label_order_k, best_accu_k = label_permute_compare(y_train, my_kmeans.labels_, 1)
print('\nLabel order:', ' --> '.join([str(lbl) for lbl in label_order_k]))
print('\nAccuracy:', round(best_accu_k,4))

ypp_k = reorder_prediction(my_kmeans.labels_, label_order_k)
fig, ax = plt.subplots(figsize=(10, 5))
yp_cat_k = [categories_train[y] for y in ypp_k]
ConfusionMatrixDisplay.from_predictions(y_train, yp_cat_k, ax=ax, xticks_rotation=9
plt.show()
```

```
Values: [3 5 2 1 0 4 6]
Length of labels: 3143
Five first labels: [3 3 3 3 3 3]

Label order: 3 --> 2 --> 5 --> 1 --> 4 --> 0 --> 6

Accuracy: 0.8661
```



Similarly as with NMF, the incorrect predictions reveal that most of humans should not have trouble categorizing them correctly.

In [33]:
```python
# Check
for item in random.sample(list(np.where(yp_cat != y_train)[0]), 4):
    print('Wrong prediction:')
    inspect(item, ypp, X_train, y_train)
    print()
```

```
Wrong prediction:
Index: 777
Predicted:       tools home - hardware - adhesives sealers
Actual:          sports outdoors - recreation - skates
Item description: black diamond longboard skateboard grip tape sheet

Wrong prediction:
Index: 667
Predicted:       girl fashion - clothing - sweater
Actual:          sports outdoors - recreation - skates
Item description: ezyroller ride on toy  new twist on  classic scooter

Wrong prediction:
Index: 1666
Predicted:       electronics - computers - scanners
Actual:          men fashion - access - wallets cards
Item description: set of   replacement insert for for bifold or trifolds wallet  car
d or picture insert

Wrong prediction:
Index: 700
Predicted:       tools home - hardware - adhesives sealers
Actual:          sports outdoors - recreation - skates
Item description: handmind plastic scooter board with safety handles for physical ed
ucation class or home use blue
```

Comparison with supervised methods

Training and testing split is done in the code below.

In [34]:
```
from sklearn.model_selection import train_test_split, cross_val_score, GridSearchCV

print('mat.shape:', mat.shape, '\nY_train.shape:', y_train.shape)
X_train_clf, X_test_clf, y_train_clf, y_test_clf = train_test_split(mat, y_train, t
```

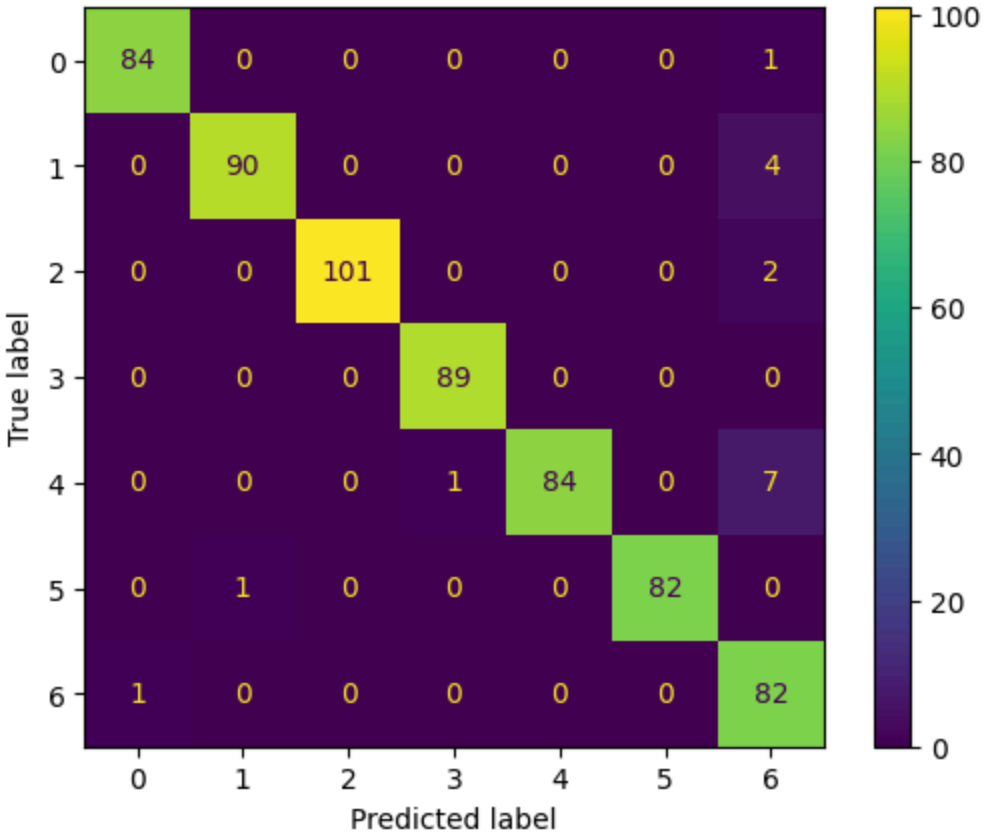
```
mat.shape: (3143, 1021)
Y_train.shape: (3143,)
```

Random Forest

Random forest, even without grid search for optimizing the hyperparameters, performs very well with always well over 90% accuracy.

In [35]:
```
from sklearn.ensemble import RandomForestClassifier

my_RF = RandomForestClassifier().fit(X_train_clf, y_train_clf) # Create the model
y_hat_RF = my_RF.predict(X_test_clf)

calc_metrics(y_test_clf,  y_hat_RF)
plt.show()
```

```
Accuracy: 0.973
Confusion matrix:
```

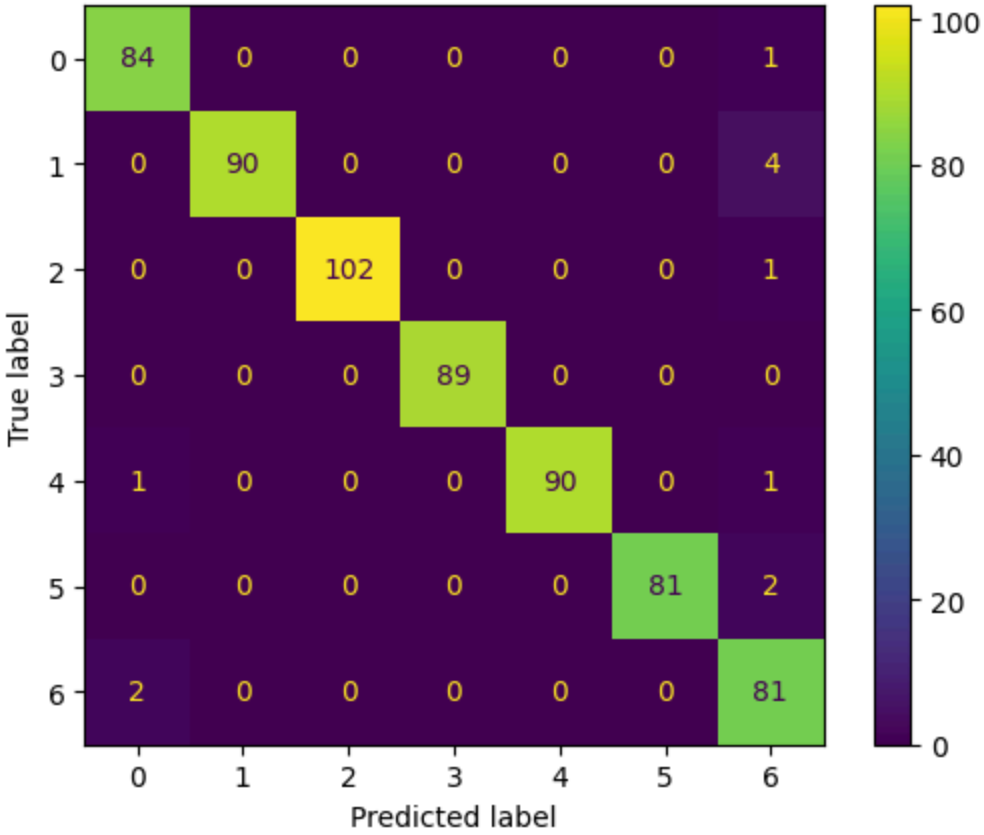


Support Vector Machine

Similarly a simple SVM has no trouble categorizing the data with most of the time over 95% accuracy.

In [36]:
```
from sklearn.svm import SVC

print('With Radial Basis Function kernel:')
my_SVC = SVC().fit(X_train_clf, y_train_clf) # Create the model
y_hat_SVC = my_SVC.predict(X_test_clf)
calc_metrics(y_test_clf, y_hat_SVC) # calculate metrics
plt.show()
```

```
With Radial Basis Function kernel:
Accuracy: 0.981
Confusion matrix:
```



Discussion and conclusion

All the models, NMF, k-means, Random Forest and SVM, worked. The supervised models had consistenlty very good accuracy, over >95% most of the time.

Non-negative Matrix Factorization performed fairly well, with accuracy typically between 60% and 90%.

In some cases the K-means model did not perform well, having accuracy of 50%. In those cases it was biased towards certain categories. Overall, k-means had similar or slightly lower accuracy than the NMF.

Both, NMF and k-means, seemed to have difficulties when there were a lot of items in the data set. With smaller data sets they perfomed at the same level as the supervised models. Supervised models perfomed well regardless of the number of items in the data set.

Training time

This data set is fairly large so just loading and cleaning the data takes a few minutes.

Detailed categories

Clustering less than 10 detailed categories is fairly fast. Most of the steps taking less than a few seconds on a laptop.

If more than 10 categories are chosen, model creation is still reasonable but label assignment for model verification using brute-force permutation becomes too long.

Main categories

Main categories have much more items so overall data handling and model building time is much longer.

Additional improvements

The models can be improvement by further cleaning the description texts. For example, removing digits often leaves the characters of part numbers intact. A better way would be removing the words with digits in them. Also, even though most of the units were removed, such as 'mm', there still seems to be some left in the data.

More filtering of the data could also help. Dropping the categories with only a handful of items can help (for example less than 100 items).