Predicting the value of a painting with polynomial regression and neural networks

CS-C3240 Machine Learning D

Introduction to the problem

To set a value for any piece of art is challenging. Hence, artists and buyers rely on the power of good, professional appraisers. However, these services can be expensive. Moreover, one might suspect there is more to the piece's market value than the appearance of the piece itself, which would also be accounted for in the appraiser's view of the value. I seek the value purely based on the piece itself, a true value of a painting without bias for anything outside the frames beside the market prices.

To solve the problem of unbiased estimation, to really predict whether one should drop out and quit their day job for art career[1], we rely on the fundamental unbiasedness itself: logic and the machine that learns. This report outlines a try on models that estimate a painting's value by the properties of the painting itself and the purchase-history of other paintings. The predictors, the hypothesis spaces, studied in this report are the polynomial functions with varying degrees and non-linear multi-layered neural networks, Multilayer Perceptron's, with varying number of layers and neurons.

The founding problem is further formalized as a machine learning problem in the next section, Problem formulation. The following section, the Data-section, discusses the data used for the learning algorithms. The Method-section, consisting of two separate sections for the two hypothesis spaces, discusses the hypothesis spaces and how the data will be applied to them. The Results-section compares the predictions of the resulting hypothesis functions. The Conclusion-section summarizes the report with discussion of the application and the limitations and improvements.

1: quitting jobs is never recommended; this model is purely for fun experiments.

Problem formulation

The problem should be formalized as a machine learning problem. A datapoint for the model represents a painting that is presented to the model by an image. For the quantities of interest, we have just a single label: the value of the painting. To get the value for paintings, we use the most recent price for which the painting was sold. Hence, the more accurate datapoint definition is a painting that has been sold. The label in the data will be an integer representing euros. The predicted value will be a floating-point numeral, though.

The images are 3-dimensional RGB-maps with dimensions (width, height, 3) with integer values ranging from 0 to 255. With the RGB-values we calculate features for the model: the number of colors, 5 most used colors, overall strengths of RGB and the use of same colors. I will explain these.

The number of colors:	Get a list of RGB-colors used in the image and use the order of the distinct set of those colors as a feature. (1 feature of type integer)
5 most used colors:	From all the pixels of the image, use the RGB-values of those N colors that have the highest count in the list of RGB-pixels as features. Also use only those values that have at least a vector distance of 15 to the previously selected value. This prevents similar colors from appearing in the list of most used colors. (5*3 features of type integer within [0, 255])
Overall strengths of RGB:	For all pixels, component-wise sum the RBG-values and divide all with some big float constant so that these features scale nicely with other features. (3 features of type float)
The use of same colors:	Divide the number of pixels with the order of the distinct set of colors used in the image. (1 feature of type float)

With these features, the model uses a total of 20 features. Intuitively, there is much more to a painting than just the color-values. However, considering shapes and algorithms for identifying these shapes, I came across a big obstacle. I found that with the amount of data necessary for this model, it would take too much time to compute these complex features. Thus, I decided to proceed with the aforementioned features which do not take too long to compute and are much easier to process.

Data

To train the model, we need a dataset that contains the RGB-map of the painting and the price that it was sold for. This data is unfortunately very hard to find. Indeed, I could not find any downloadable dataset that would be ready with these requirements.

There are, however, free-to-use websites that allow the user to search for paintings and their prices for which they were sold. One such website would be https://www.artsy.net/price-database. I could not find any link to any downloadable dataset of the database that the servers utilize.

As a solution to this problem, I created scripts (see appendices) that query these databases through the websites and form csv-datafiles from the returned HTML-pages. The script 1 parses the returned HTML-pages and writes rows with the links to the images and the corresponding prices and currencies into a csv-file. The script 2 reads the rows from this csv-file, requests the images from the links and computes the necessary features from the RGB-maps. The computed datapoints are written to another csv-file, resulting in a csv-datafile containing the data with the features and label explained in the previous section. With script 3, the rows are filtered, and the columns are separated into two different csv-files: one with the strengths and prices (“rgb_data_strength_price.csv”), and one with the color counts and most used RGB-values (“rgbdata_freq.csv”). These data frames are joined together in the model code (see appendices) with common column image_id.

From artsy.net with these scripts and after filtering null-valued rows, I managed to collect 2832 valid datapoints.

(The datafiles are not included in this report)

The columns in “rgbdata_strength_price.csv” are named as such:
image_id,R_strength,G_strength,B_strength,price_eur

The columns in “rgbdata_freq.csv” are named as such:
image_id,color_count,avg_use_of_same_color,1_used_R,1_used_G,1_used_B,2_used_R,2_used_G,2_used_B,3_used_R,3_used_G,3_used_B,4_used_R,4_used_G,4_used_B,5_used_R,5_used_G,5_used_B

Methods

For the code of the models, see appendices.

Polynomial functions with varying degrees

The first hypotheses to try are polynomial functions with varying degrees. To minimize loss in learning, the model uses the squared error. This means the model has emphasis on outliers which is wanted. Also, this allows me to use the PolynomialFeatures-class with LinearRegression-class in scikit-library [1] [2] in python, since that model uses the same hypothesis space and loss.

The polynomial functions had degrees from 1 to 5. Testing with larger degrees proved to be too costly for my computer because of the large dimensions of the data. The degree of 1 resulted in the best performance while the larger degrees had significantly greater validation error.

The performance between these hypotheses was compared by the absolute validation error. With this, the unit of error is in euros which makes it easy to interpret.

The validation set comes from the division of the data into a training set and a validation set. The training set is used to train the model while the validation set is used to measure the performance. I used a data-split of 20% validation, 20% testing and 60% training. The validation set is small so that the training is not compromised. The dataset is relatively large so there should be little opportunity for unlucky splits. Moreover, the testing of a degree was done multiple times to minimize the impact of unlucky splits and to better understand the effectiveness of the hypothesis space.

Multilayer Perceptron

The second hypothesis space I consider is the multi-layered neural networks with varying number of layers and neurons. Inspecting individual features with scatterplots, I found that the distribution of the features in relation to the label is complex. Therefore, the deep network of neurons should be suitable for a hypothesis space. The minimized loss-function is again the squared error since the emphasis on outliers is wanted. With this, I can utilize the library functions of scikit-learn for the MLPRegression [3], since the hypothesis space and the loss function used are the same.

The number of layers and neurons was decided with multiple tests of different numbers. I found that the best training and validation errors resulted from 63 layers and 63 neurons. Again, the performance was compared by the absolute error for the same reason as in the polynomial model. The data-split (20-20-60 for validation, test, and training) was also the same since there was no reason to alter the sizes. The motivation behind this design choice still holds (see Polynomial functions with varying degrees for explanation).

Results

The results from the methods are compared by the absolute error. As the units are same, this can be compared to the average value of the label. The average price of the datapoints is about 25 000 euros. With the polynomial model of degree 1, the training error is 27400 euros, and the validation error is 30900 euros. With the Multilayer Perceptron with 63 neurons and layers, the training error is 22500 euros, and the validation error is 26300 euros.

From these results, we can concur that the Multilayer Perceptron performs better as it has smaller errors. We can calculate a final test error for the chosen model. The error is calculated on the test set, which was divided from the original dataset along the validation and training sets. The size of the test set is the same as the validation set (20% of the full dataset) The calculated test error (a mean absolute error) for the Multilayer Perceptron is 25000 euros.

Conclusion

The purpose of this report was to try to predict the value of a painting using the polynomial regression and the multi-layered neural network and compare the results. The best performing model was the Multilayer Perceptron, which predicted the test set with a mean absolute error of 2500 euros. This number is about the same as the average price of the paintings in the full dataset. This shows that even the best performing option of these hypothesis spaces performed rather poorly.

With both models, the training error was significantly larger than the validation error, which suggests underfitting. However, further investigation of the data shows that the more impactful and likely reason for poor performance is the lack of proper correlation between the label and the features.

To improve the performance, better features must be used. More accurately, features, that have stronger correlation to the label, must be used. As mentioned in the Problem formulation -section, there are obviously more features in a painting than just the color values, e.g., the shapes and forms. However, the algorithms to calculate these features of painting are costly.

As a conclusion of the results and investigations, we can speculate that further development with the features used in these models is rather meaningless. To properly predict the values, much more complex features are necessary.

References

[1] Scikit-learn, [Online]. Available: https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.PolynomialFeatures.html. [Accessed 31 3 2022].

[2] Scikit-learn, [Online]. Available: https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LinearRegression.html. [Accessed 31 3 2022].

[3] Scikit-learn, [Online]. Available: https://scikit-learn.org/stable/modules/generated/sklearn.neural_network.MLPRegressor.html. [Accessed 31 3 2022].

Appendices

The model-notebook and scripts 1, 2 and 3 are shown on the following pages. I was unable to upload them here as a file. The scripts are written with tiny font for ease of copying. Please copy and paste them to a preferred editor if you wish to inspect them.

Model-notebook:

```
[ ]: import numpy as np
     import pandas as pd
     from matplotlib import pyplot as plt
     from sklearn.model_selection import train_test_split
     from sklearn.preprocessing import PolynomialFeatures
     from sklearn.linear_model import LinearRegression
     from sklearn.neural_network import MLPRegressor
     from sklearn.metrics import mean_squared_error, mean_absolute_error
```

```
[ ]: data_file1 = "../rgbdata_strength_price.csv"
     data_file2 = "../rgbdata_freq.csv"

     df1 = pd.read_csv(data_file1)
     df2 = pd.read_csv(data_file2)

     df = pd.merge(left=df2, right=df1, left_on="image_id", right_on="image_id")
     df = df.dropna(axis=0)

     data = df.drop(["image_id"], axis=1)

     data.describe()
```

```
[ ]: X = data.drop(["price_eur"], axis=1)
     y = data["price_eur"].to_numpy()
     X_test = data["1_used_B"].to_numpy()
     X_test.reshape(-1,1)


     data.head(5)
     #data["price_eur"].max()
```

```
[ ]: X_train, X_rest, y_train, y_rest  = train_test_split(X, y, test_size=0.4,␣
      ↪random_state=1)
     X_val, X_test, y_val, y_test  = train_test_split(X_rest, y_rest, test_size=0.5,␣
      ↪random_state=1)

     lin_regr = LinearRegression(fit_intercept=False)
```

```
poly = PolynomialFeatures(degree=1)
X_train_poly = poly.fit_transform(X_train)
lin_regr.fit(X_train_poly, y_train)
y_pred_train = lin_regr.predict(X_train_poly)

X_val_poly = poly.fit_transform(X_val)
y_pred_val = lin_regr.predict(X_val_poly)


tr_error = mean_absolute_error(y_train, y_pred_train)
val_error = mean_absolute_error(y_val, y_pred_val)

print("training error: " + str(tr_error) + " euros")
print("validation error: " + str(val_error) + " euros")
```

[ ]:
```
X_train, X_rest, y_train, y_rest = train_test_split(X, y, test_size=0.4,␣
 ↪random_state=1)
X_val, X_rtest, y_val, y_test = train_test_split(X_rest, y_rest, test_size=0.5,␣
 ↪random_state=1)

nof_layers = 63
neurons_on_layer = 63
hidden_layer_sizes = tuple([neurons_on_layer] * nof_layers)

mlp_regr = MLPRegressor(hidden_layer_sizes=hidden_layer_sizes, random_state=1)

mlp_regr.fit(X_train, y_train)

y_pred_train = mlp_regr.predict(X_train)
y_pred_val = mlp_regr.predict(X_val)

tr_error = mean_absolute_error(y_train, y_pred_train)
val_error = mean_absolute_error(y_val, y_pred_val)

print("training error: " + str(tr_error) + " euros")
print("validation error: " + str(val_error) + " euros")
```

[ ]:
```
y_pred_test = mlp_regr.predict(X_test)
test_error = mean_absolute_error(y_test, y_pred_test)

print("test error: " + str(test_error) + " euros")
```

[ ]:
```
for c in ["R", "G", "B"]:
    feature = c + "_strength"

    X_test = data[feature].to_numpy().reshape(-1,1)
    y = data["price_eur"]
    X_train, X_val, y_train, y_val  = train_test_split(X_test, y, test_size=0.2,
↪random_state=1)

    lin_regr = LinearRegression(fit_intercept=False)
    poly = PolynomialFeatures(degree=9)
    X_train_poly = poly.fit_transform(X_train)
    lin_regr.fit(X_train_poly, y_train)

    X_val_poly = poly.fit_transform(X_val)

    X_fit = np.linspace(0, 1, 100)
    #plt.tight_layout()
    plt.plot(X_fit, lin_regr.predict(poly.transform(X_fit.reshape(-1,1))),
↪label="Model")
    plt.scatter(X_train, y_train, color="b",  =10, label="Train Datapoints")
    plt.scatter(X_val, y_val, color="r",  =10, label="Validation Datapoints")
    plt.xlabel(feature)
    plt.legend(loc="best")
    plt.ylabel('price')
    plt.title(feature)
    plt.show()
```

```
[ ]: fig = plt.figure()

     ax = fig.add_subplot(1,1,1)

     feature = "avg_use_of_same_color"
     X_test = data[feature].to_numpy().reshape(-1,1)

     ax.scatter(X_test, y)
     ax.set_ylabel("price in euro")
     ax.set_xlabel(feature)
     ax.set_title("price and " + feature)

     plt.show()
```

Script 1:

```
import requests
from bs4 import BeautifulSoup
from matplotlib import image
from matplotlib import pyplot
from io import BytesIO
from PIL import Image

import re
import time
import csv


def price_filter_thing(c):
    if re.search("\d", c) == None:
        return False
    else:
        return True

cookieHeader = dict(SESSION_ID = "I removed my session_id for security reasons, you can get yours by logging into artsy.net")
datapoints = [] # will have arrays [img-url, price]

print()
print("TIEDONKERÄYS: Maalaukset & niiden hinnat")

print()
print("Aloita tiedonkeräys. 100 sivua.")
print(". . . ")

beginning_time = time.time()
file_path = "data_links.csv"
f = open(file_path, "w", newline="")
writer = csv.writer(f)

writer.writerow(["image_id", "image_link", "price", "currency"])

counter = 1
for page_number in range(100):
    page_number += 1

    collectionR = requests.get("https://www.artsy.net/collect?page=" + str(page_number), cookies=cookieHeader)
    soup = BeautifulSoup(collectionR.text, 'html.parser')

    three_columns = soup.find_all(attrs={"class": "Box-sc-15se88d-0 Flex-cw39ct-0 ArtworkGrid__InnerContainer-sc-1jsqquq-1 LbPWX cqZQYS fresnel-
greaterThanOrEqual-lg"})
    three_columns = str(three_columns[1])

    cols_soup = BeautifulSoup(three_columns, 'html.parser')

    first_col = cols_soup.div.div
    second_col = first_col.next_sibling
    third_col = second_col.next_sibling

    cols = [first_col, second_col, third_col]
    for col in cols:
        soup = BeautifulSoup(str(col), 'html.parser')
        artworkGridItems = soup.find_all(attrs={"data-test": "artworkGridItem"})

        for gridItem in artworkGridItems:
            soup2 = BeautifulSoup(str(gridItem), 'html.parser')

            img_url = soup2.a.div.img['src']

            #<div color="black100" font-weight="bold" font-family="sans" class="Box-sc-15se88d-0 Text-sc-18gcpao-0 eXbAnU jkuGdd">US$19,995 </div>
            info = list(soup2.find("div", attrs={"class": "Box-sc-15se88d-0 Text-sc-18gcpao-0 eXbAnU jkuGdd"}).children)[0]
            price_string = str(info)

            currency = "unknown"
            if "€" in price_string:
                currency = "EUR"
            elif "£" in price_string:
                currency = "POU"
            elif "$" in price_string and "US" in price_string:
                currency = "USD"

            weird_char = "–"
            if weird_char in price_string:
                price_string = price_string.split(weird_char)[1]


            price = filter(price_filter_thing, price_string)
            price = "".join(price)
            if price == "":
                price = 0
            else:
                price = int(price)

            new_row = [counter, img_url, price, currency]
            writer.writerow(new_row)
            datapoints.append([img_url, price])

            counter += 1

#writer.writerows(datapoints) #already doing this just above
f.close()
ending_time = time.time()
running_time = (ending_time - beginning_time) / 60.0

print("\n--- DONE ---\n")
print("datapoints received:  " + str(len(datapoints)))
print("data stored to: " + file_path)
print("time consumed: " + str(running_time) + " minutes")
print()




if False:
    img_url = "https://i.pinimg.com/550x/dd/50/43/dd50435018e7b325324f3f4408774716.jpg"  #test some url
    r = requests.get(img_url)
    img = BytesIO(r.content)
    load_image = Image.open(img)
    load_image.show() #works with default image-opener-programme in my windows 10

if False:
    img_pyplot = image.imread(img)
    pyplot.imshow(img_pyplot)
    pyplot.show()  #doesnt work, i think it needs a PNG
```

Script 2:

```
from ctypes.wintypes import RGB
from pyparsing import line
import requests
import csv
from io import BytesIO
from PIL import Image

import csv
import time

beginning_time = time.time()

read_file_path = "data_links.csv"

write_file_path = "image_RGB_data.csv"
#write_test_path = "data_test.csv"

f = open(read_file_path)
w = open(write_file_path, "w", newline="")
reader = csv.reader(f, delimiter = ",")
writer = csv.writer(w)

print()
print("Aloitetaan tiedon prosessointi...")
print(". . .")
print()


def vector_distance(origin: tuple, destination: tuple):
    ret = 0
    for i in range(len(origin)):
        ret += abs(destination[i] - origin[i])
    return ret


line_count = 0
misshapes = 0

for row in reader:
    if line_count == 0:
        text = "columns in " + read_file_path + ": " + ", ".join(row)
        line_count += 1
        writer.writerow(["image_id", "rgb_strengths", "five_most_frequent_rgb", "color_count", "avg_use_of_same_color", "width", "height", "price", "currency"])
    else:
        image_id = row[0]
        price = row[2]
        currency = row[3]

        img_url = row[1]
        r = requests.get(img_url)

        img = BytesIO(r.content)
        load_image = Image.open(img)

        pil_img = load_image.load()


        rgb_values = dict()
        rgb_strengths = [0.0, 0.0, 0.0]
        divider = 10000.0

        width, height = load_image.size
        max_strength = 255 * width * height / divider
        for i in range(width):
            for j in range(height):
                value = pil_img[i, j]

                if not isinstance(value, int):
                    length = len(value)
                    for x in range(min(3, length)):
                        rgb_strengths[x] = rgb_strengths[x] + value[x]/divider

                    if length == 3:
                        if value in rgb_values:
                            rgb_values[value] = rgb_values[value] + 1
                        else:
                            rgb_values[value] = 1

        line_count += 1
        if rgb_strengths[0] == 0.0 and rgb_strengths[1] == 0.0 and rgb_strengths[2] == 0.0:
            misshapes += 1
            continue
        rgb_strengths = [str(rgb_strengths[0]/max_strength), str(rgb_strengths[1]/max_strength), str(rgb_strengths[2]/max_strength)]
        #print(rgb_strengths)


        rgb_values_sorted = sorted(rgb_values, key=rgb_values.get, reverse=True)
        if len(rgb_values_sorted) == 0:
            misshapes += 1
            continue

        take_first: list[tuple[int]] = []
        test_len = len(rgb_values_sorted)
        bruh = True
        i = 0
        prev = (0,0,0)
        while (bruh):
            if len(take_first) == 0:
                try:
                    a = rgb_values_sorted[i]
                    take_first.append(a)
                    prev = a
                except:
                    j = 0
                    while (len(take_first < 5)):
                        if rgb_values_sorted[j] not in take_first:
                            take_first.append(rgb_values_sorted[j])
            else:
                candidate = rgb_values_sorted[i]
                if vector_distance(prev, candidate) > 15:
                    take_first.append(candidate)
                    prev = candidate
            if len(take_first) > 4:
                bruh = False
            else:
                i += 1
        assert(len(take_first) == 5)

        take_first = [str(tup) for tup in take_first]
        color_count = len(rgb_values.keys())
        avg_use_of_same_color = width * height / float(color_count)

        writer.writerow([image_id, "-".join(rgb_strengths), ">".join(take_first), color_count, avg_use_of_same_color, width, height, price, currency])

f.close()
w.close()

ending_time = time.time()
running_time = (ending_time - beginning_time) / 60.0

print("---DONE---")
print("datapoints processed: " + str(line_count))
print("misshaped datapoints: " + str(misshapes))
print("time consumed: " + str(running_time) + " minutes")
print()
```

Script 3:

```
from asyncio import base_futures
from audioop import avg
from ctypes.wintypes import RGB
from pyparsing import line
import csv
import time

beginning_time = time.time()

read_file_path = "image_RGB_data.csv"

write_file_path1 = "rgbdata_freq.csv"
write_file_path2 = "rgbdata_strength_price.csv"

f = open(read_file_path)

w_rgb_freq = open(write_file_path1, "w", newline="")
w_strength_price = open(write_file_path2, "w", newline="")

reader = csv.reader(f, delimiter = ",")
writer_strength_price = csv.writer(w_strength_price)
writer_rbg_freq = csv.writer(w_rgb_freq)

print()
print("Aloitetaan tiedon filterointi ja separointi...")
print(". . .")
print()

line_count = 0
filtered_datapoints = 0

for row in reader:
    currency_unknown = False

    if line_count == 0:
        text = "columns in " + read_file_path + ": " + ", ".join(row)

        writer_strength_price.writerow(["image_id", "R_strength", "G_strength", "B_strength", "width", "height", "price_eur"])
        r = ["image_id", "color_count", "avg_use_of_same_color"]
        for i in ["1", "2", "3", "4", "5"]:
            for c in ["R", "G", "B"]:
                r.append(i + "_used_" + c)
        writer_rbg_freq.writerow(r)

    else:
        image_id = row[0]
        rgb_strengths = row[1]
        most_freq_rgb = row[2]
        color_count = int(row[3])
        avg_use_of_same_color = float(row[4])
        width = int(row[5])
        height = int(row[6])
        price = int(row[7])
        currency = row[8]

        match currency:
            case "EUR":
                price = price
            case "POU":
                price = int(price * 1.21)
            case "USD":
                price = int(price * 0.92)
            case unknown:
                currency_unknown = True

        rgb_strengths = rgb_strengths.split("-")

        r_strength = float(rgb_strengths[0])
        g_strength = float(rgb_strengths[1])
        b_strength = float(rgb_strengths[2])

        if ( r_strength == 0.0 and g_strength == 0.0 and b_strength == 0.0 ) or currency_unknown:
            filtered_datapoints += 1
            continue

        temp = most_freq_rgb.split(">")
        most_freq_rgb: list[list[int]] = []
        for a in temp:
            a = a[1:-1]
            a = a.split(", ")
            try:
                a = [int(b) for b in a]
            except:
                filtered_datapoints += 1
                continue
            most_freq_rgb.append(a)

        r = [image_id, color_count, avg_use_of_same_color]
        for i in range(len(most_freq_rgb)):
            for c in range(3):
                i_used_c = most_freq_rgb[i][c]
                r.append(i_used_c)

        writer_strength_price.writerow([image_id, r_strength, g_strength, b_strength, width, height, price])
        writer_rbg_freq.writerow(r)


    line_count += 1



f.close()
w_rgb_freq.close()
w_strength_price.close()

ending_time = time.time()
running_time = (ending_time - beginning_time) / 60.0

print("---DONE---")
print("datapoints accepted: " + str(line_count))
print("filtered datapoints: " + str(filtered_datapoints))
print("time consumed: " + str(running_time) + " minutes")
print()
```