ML_Project_IshaTawde

May 15, 2020

1 Machine Learning Project - Isha Tawde | it746

In [1]:
```
# Import statements
import matplotlib.pyplot as plt
from matplotlib import style
import numpy as np
from sklearn import preprocessing
import pandas as pd
from numpy import *
```

In [2]:
```
# Regression , K-means(Isha) , K-median(Soni), NB, perceptron, KNN , PCA/TSNE & Bokeh
```

In [3]:
```
#load the dataset
df = pd.read_csv("cosmetics.csv")
#check the rows
display(df.sample(5))
```

	Label	brand	name	price	rank	ingredients	Combination	Dry
1147	Eye cream	FRESH	Crème Ancienneő Eye Cream	125	4.3	Limnanthes Alba (Meadowfoam) Seed Oil, Simmond...	0	0
520	Cleanser	MURAD	Pore Reform Skin Smoothing Polish	34	4.5	Water, Silica, Stearic Acid, Jojoba Esters, Gl...	0	0
518	Cleanser	KIEHL'S SINCE 1851	Clearly Corrective Brightening & Exfoliating ...	29	4.3	Visit the Kiehl's Since 1851 boutique	1	1
939	Face Mask	DR. JART+	Dermask Water Jet Soothing Hydra Solution	6	4.1	Water, Glycerin, Butylene Glycol, Alcohol, Pan...	1	1
592	Treatment	OLEHENRIKSEN	Truth Serum - Starlight Holiday Edition	128	4.0	Water, Sodium Ascorbyl Phosphate, Calcium Asco...	1	1

```
      Normal  Oily  Sensitive
1147       0     0          0
520        0     0          0
518        1     1          1
939        1     1          1
592        1     1          0
```

```
In [4]: #df[df['ingredients']=='No Info'].count()
        df = pd.DataFrame(df[df.ingredients != 'No Info'])
```

2 Regression

```
In [5]: #Setup
        X = np.array(df['price']).reshape(-1,1)
        Y = np.array(df['rank']).reshape(-1,1)

        ones = np.ones(len(X)).reshape(-1,1)
        A=np.hstack((ones,X))

        w=np.array([0,0]).reshape(-1,1)
```

```
In [6]: #Direct OLS
        w_directOLS =np.linalg.inv((A.T).dot(A)).dot(A.T).dot(Y)
        w_directOLS
```

```
Out[6]: array([[ 4.16502602e+00],
               [-2.85973912e-04]])
```

```
In [7]: #Direct Ridge
        w_directRidge =np.linalg.inv((A.T).dot(A) + 0.1*np.identity(2)).dot(A.T).dot(Y)
        w_directRidge
```

```
Out[7]: array([[ 4.16429373e+00],
               [-2.78000888e-04]])
```

```
In [8]: plt.rcParams['figure.figsize'] = (20.0, 10.0)
```

```
In [9]: #Simple Linear Regression Model using Ordinary Least Square Method.

        X = df['price'].values
        Y = df['rank'].values
```

```
In [10]: # Mean X and Y
         mean_x = np.mean(X)
         mean_y = np.mean(Y)

         # Total number of values
```

```
        m = len(X)

        # Using the formula to calculate b1 and b2
        numer = 0
        denom = 0
        for i in range(m):
            numer += (X[i] - mean_x) * (Y[i] - mean_y)
            denom += (X[i] - mean_x) ** 2
        b1 = numer / denom
        b0 = mean_y - (b1 * mean_x)

        # Print coefficients
        print(b1, b0)
```

```
-0.00028597391182698633 4.165026022508408
```

In [11]:
```
#Y=0 + 1X ---> Rank = 4.17 + (-0.000355)*Price
```

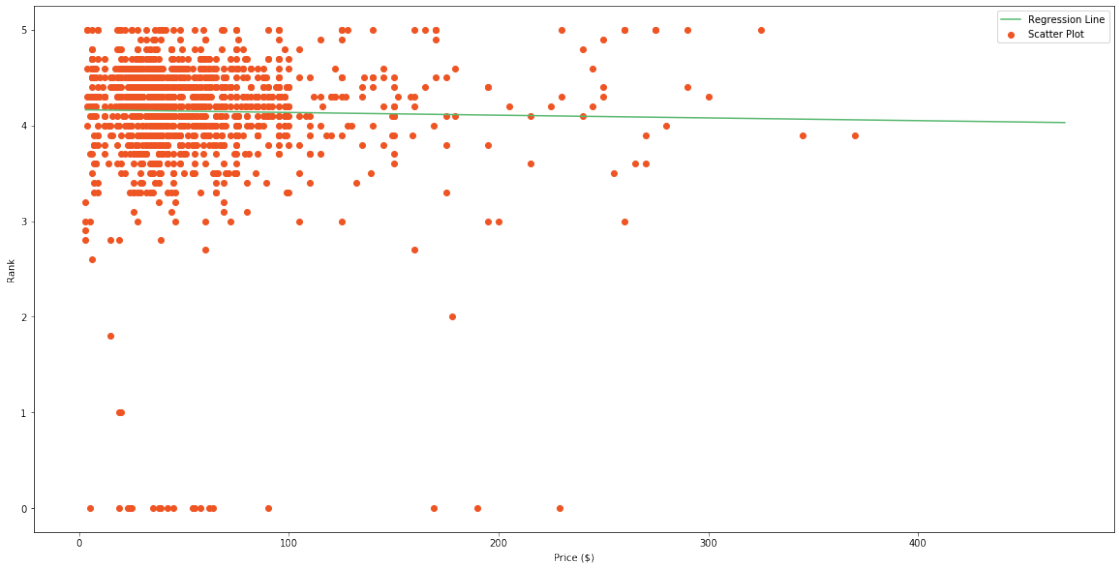
In [12]:
```
        # Plotting Values and Regression Line

        max_x = np.max(X) + 100
        min_x = np.min(X) - 0

        # Calculating line values x and y
        x = np.linspace(min_x, max_x, 1000)
        y = b0 + b1 * x

        # Ploting Line
        plt.plot(x, y, color='#58b970', label='Regression Line')
        # Ploting Scatter Points
        plt.scatter(X, Y, c='#ef5423', label='Scatter Plot')

        plt.xlabel('Price ($)')
        plt.ylabel('Rank')
        plt.legend()
        plt.show()
```




```
In [13]: # Calculating Root Mean Squares Error
         rmse = 0
         for i in range(m):
             y_pred = b0 + b1 * X[i]
             rmse += (Y[i] - y_pred) ** 2
         rmse = np.sqrt(rmse/m)
         print(rmse)
```

```
0.6359393328657976
```

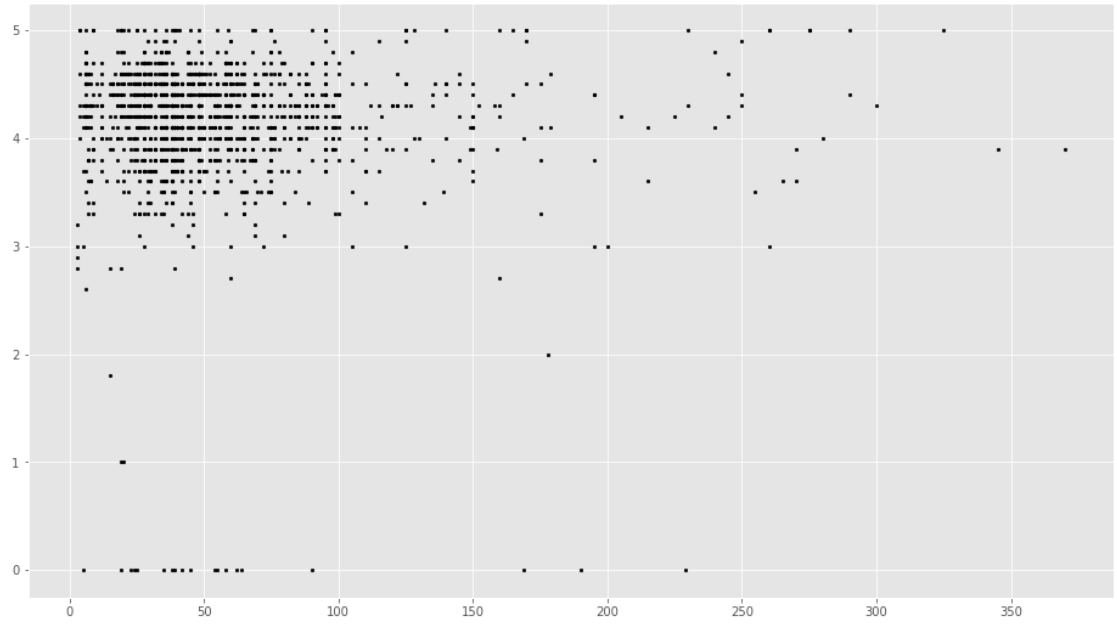
The result for RMSE is satisfactory for the given hypothesis.

3 K-Means

```
In [14]: #K-Means Clustering
         %matplotlib inline
         from copy import deepcopy
         plt.rcParams['figure.figsize'] = (16, 9)
         plt.style.use('ggplot')
```

```
In [15]: # Getting the values and plotting it
         f1 = df['price'].values
         f2 = df['rank'].values
         X = np.array(list(zip(f1, f2)))
         plt.scatter(f1, f2, c='black', s=7)
```

```
Out[15]: <matplotlib.collections.PathCollection at 0x1071e5c18>
```



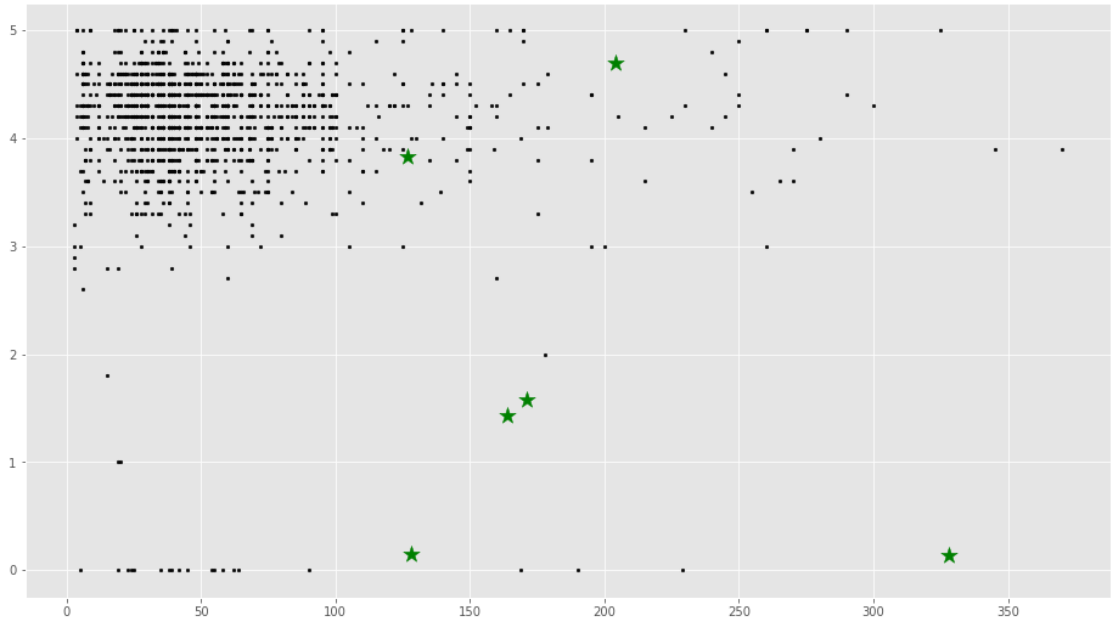
In [16]:
```
# Euclidean Distance Caculator
def dist(a, b, ax=1):
    return np.linalg.norm(a - b, axis=ax)
```

In [17]:
```
# Number of clusters
k = 6
# X coordinates of random centroids
C_x = np.random.randint(0, np.max(f1)-20, size=k)
# Y coordinates of random centroids
C_y = np.random.uniform(0, np.max(f2), size=k)
C = np.array(list(zip(C_x, C_y)), dtype=np.float32)
print(C)
```

```
[[1.6400000e+02 1.4371647e+00]
 [1.2700000e+02 3.8293843e+00]
 [1.7100000e+02 1.5847275e+00]
 [1.2800000e+02 1.5365778e-01]
 [2.0400000e+02 4.6981449e+00]
 [3.2800000e+02 1.3689765e-01]]
```

In [18]:
```
# Plotting along with the Centroids
plt.scatter(f1, f2, c='#050505', s=7)
plt.scatter(C_x, C_y, marker='*', s=200, c='g')
```

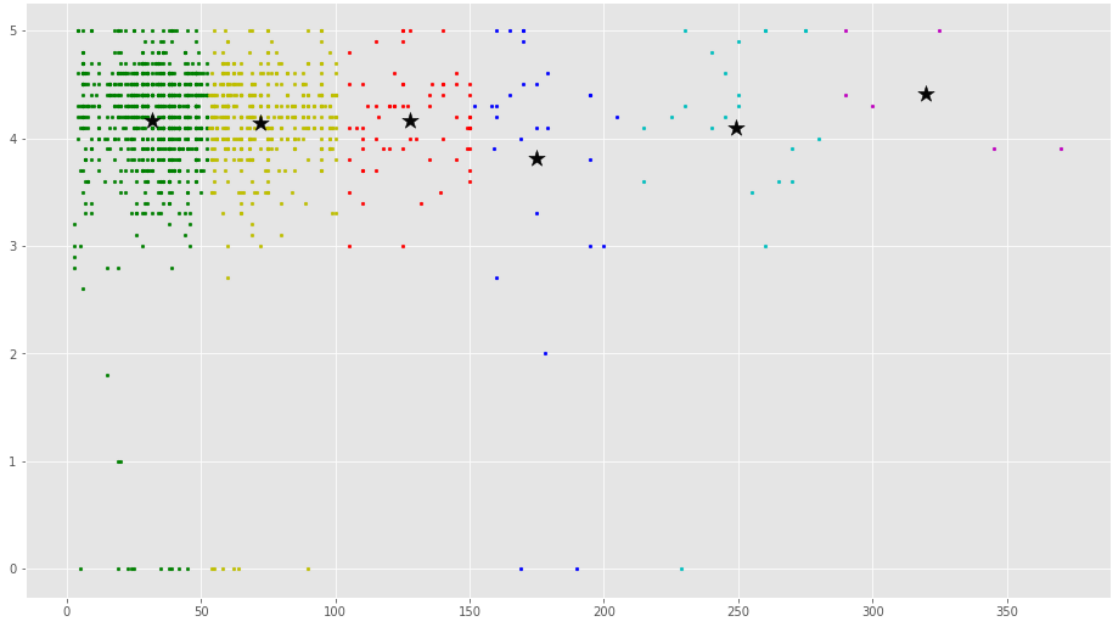
Out[18]: <matplotlib.collections.PathCollection at 0x10739cc18>



In [19]:
```
# To store the value of centroids when it updates
C_old = np.zeros(C.shape)
# Cluster Lables(0, 1, 2)
clusters = np.zeros(len(X))
# Error func. - Distance between new centroids and old centroids
error = dist(C, C_old, None)
# Loop will run till the error becomes zero
while error != 0:
    # Assigning each value to its closest cluster
    for i in range(len(X)):
        distances = dist(X[i], C)
        cluster = np.argmin(distances)
        clusters[i] = cluster
    # Storing the old centroid values
    C_old = deepcopy(C)
    # Finding the new centroids by taking the average value
    for i in range(k):
        points = [X[j] for j in range(len(X)) if clusters[j] == i]
        C[i] = np.mean(points, axis=0)
    error = dist(C, C_old, None)
```

In [20]:
```
colors = ['r', 'g', 'b', 'y', 'c', 'm']
fig, ax = plt.subplots()
for i in range(k):
        points = np.array([X[j] for j in range(len(X)) if clusters[j] == i])
        ax.scatter(points[:, 0], points[:, 1], s=7, c=colors[i])
ax.scatter(C[:, 0], C[:, 1], marker='*', s=200, c='#050505')
```

Out[20]: <matplotlib.collections.PathCollection at 0x1a1c08d748>



4 K-Median

In [21]:
```
#Kmeans & K-Medians
colors = ['r','g','b','c','k','o','y']

class K_Means:
    def __init__(self, k=6, tol=0.001, max_iter=300):
        self.k = k
        self.tol = tol
        self.max_iter = max_iter

    def fit(self,data):

        self.centroids = {}

        for i in range(self.k):
            self.centroids[i] = data[i]

        for i in range(self.max_iter):
            self.classifications = {}

            for i in range(self.k):
                self.classifications[i] = []
```

```
                    for featureset in X:
                        distances = [np.linalg.norm(featureset-self.centroids[centroid]) for c
                        classification = distances.index(min(distances))
                        self.classifications[classification].append(featureset)

                    prev_centroids = dict(self.centroids)

                    for classification in self.classifications:
                        self.centroids[classification] = np.mean(self.classifications[classifi

                    optimized = True

                    for c in self.centroids:
                        original_centroid = prev_centroids[c]
                        current_centroid = self.centroids[c]
                        if np.sum((current_centroid-original_centroid)/original_centroid*100.0
                            #print(np.sum((current_centroid-original_centroid)/original_centro
                            optimized = False

                    if optimized:
                        break

            def predict(self,data):
                distances = [np.linalg.norm(data-self.centroids[centroid]) for centroid in sel
                classification = distances.index(min(distances))
                return classification

In [22]: df_temp = pd.read_csv("cosmetics.csv")
         df_temp_dropped = df_temp.drop(['ingredients','name','Combination'  ,'Dry'  ,'Normal'

         def handle_non_numerical_data(df):

             # handling non-numerical data: must convert.
             columns = df.columns.values

             for column in columns:
                 text_digit_vals = {}
                 def convert_to_int(val):
                     return text_digit_vals[val]

                 if df[column].dtype != np.int64 and df[column].dtype != np.float64:

                     column_contents = df[column].values.tolist()
                     #finding just the uniques
                     unique_elements = set(column_contents)
                     # great, found them.
                     x = 0
                     for unique in unique_elements:
```

```
                        if unique not in text_digit_vals:
                            # creating dict that contains new
                            # id per unique string
                            text_digit_vals[unique] = x
                            x+=1
                    # now we map the new "id" vlaue
                    # to replace the string.
                    df[column] = list(map(convert_to_int,df[column]))

            return df

        df_temp_dropped = handle_non_numerical_data(df_temp_dropped)
        print(df_temp_dropped.head())


        X = np.array(df_temp_dropped.drop(['Label'], 1).astype(float))
        y = np.array(df_temp_dropped['Label'])


        clf = K_Means()
        clf.fit(X)
        prediction_list=[]
        correct = 0
        for i in range(len(X)):

            predict_me = np.array(X[i].astype(float))
            predict_me = predict_me.reshape(-1, len(predict_me))
            prediction = clf.predict(predict_me)
            if prediction == y[i]:
                correct += 1
            prediction_list.append(clf.predict(predict_me))

        print(correct/len(X))
```

```
   Label  brand  price  rank
0      3     53    175   4.1
1      3     84    179   4.1
2      3     50     68   4.4
3      3     53    175   3.8
4      3     88     38   4.1
0.20380434782608695
```

```
In [23]: print((correct/len(X))*100)
```

```
20.380434782608695
```

The algorithm has classified 20% correct Labels.

5 Naive-Bayes

```
In [24]: def my_naive_bayes(data, attributes, target):

             x =[0,1,0,1,0]

             def proba(arr, z):

                 num_z = 0
                 for outcome in arr:
                     if (outcome == z):
                         num_z += 1
                 return num_z/len(arr)

             def conditional_proba(data, attributes, target, class_value, xi):
                 arr = data[data[target] == class_value] [attributes]
                 num_xi = 0
                 for outcome in arr:
                     if (outcome == xi):
                         num_xi += 1
                 return num_xi/len(arr)

             class_values = unique(data[target])

             w = []

             for j in range(len(class_values)):
                 product_cond_proba_given_class = 1
                 print(class_values[j])
                 proba_for_class = proba(data[target],class_values[j])

                 print('P(Label='+class_values[j]+')=', proba_for_class)
                 for i in range(len(attributes)):

                     cond_proba = conditional_proba(data,attributes[i], target, class_values[j]
                     temp='P('+attributes[i] +'= '+ str(x[i]) +' |Label='+ class_values[j] +')
                     print(temp)
                     product_cond_proba_given_class *= cond_proba
                     Proba_for_class_given_x = proba_for_class*product_cond_proba_given_class
                 w.append(Proba_for_class_given_x)
                 temp1='P(Label='+class_values[j]+'|x)='#+str(Proba_for_class_given_x)
                 print(temp1)
             print('\n\nThe predcited class is')

             return class_values[argmax(w)]

In [25]: my_naive_bayes(df, ['Combination', 'Dry', 'Normal', 'Oily', 'Sensitive'], 'Label')

Cleanser
```

```
P(Label=Cleanser)= 0.18426501035196688
('P(Combination= 0 |Label=Cleanser)',)
('P(Dry= 1 |Label=Cleanser)',)
('P(Normal= 0 |Label=Cleanser)',)
('P(Oily= 1 |Label=Cleanser)',)
('P(Sensitive= 0 |Label=Cleanser)',)
P(Label=Cleanser|x)=
Eye cream
P(Label=Eye cream)= 0.14216701173222912
('P(Combination= 0 |Label=Eye cream)',)
('P(Dry= 1 |Label=Eye cream)',)
('P(Normal= 0 |Label=Eye cream)',)
('P(Oily= 1 |Label=Eye cream)',)
('P(Sensitive= 0 |Label=Eye cream)',)
P(Label=Eye cream|x)=
Face Mask
P(Label=Face Mask)= 0.18357487922705315
('P(Combination= 0 |Label=Face Mask)',)
('P(Dry= 1 |Label=Face Mask)',)
('P(Normal= 0 |Label=Face Mask)',)
('P(Oily= 1 |Label=Face Mask)',)
('P(Sensitive= 0 |Label=Face Mask)',)
P(Label=Face Mask|x)=
Moisturizer
P(Label=Moisturizer)= 0.2028985507246377
('P(Combination= 0 |Label=Moisturizer)',)
('P(Dry= 1 |Label=Moisturizer)',)
('P(Normal= 0 |Label=Moisturizer)',)
('P(Oily= 1 |Label=Moisturizer)',)
('P(Sensitive= 0 |Label=Moisturizer)',)
P(Label=Moisturizer|x)=
Sun protect
P(Label=Sun protect)= 0.11594202898550725
('P(Combination= 0 |Label=Sun protect)',)
('P(Dry= 1 |Label=Sun protect)',)
('P(Normal= 0 |Label=Sun protect)',)
('P(Oily= 1 |Label=Sun protect)',)
('P(Sensitive= 0 |Label=Sun protect)',)
P(Label=Sun protect|x)=
Treatment
P(Label=Treatment)= 0.17115251897860592
('P(Combination= 0 |Label=Treatment)',)
('P(Dry= 1 |Label=Treatment)',)
('P(Normal= 0 |Label=Treatment)',)
('P(Oily= 1 |Label=Treatment)',)
('P(Sensitive= 0 |Label=Treatment)',)
P(Label=Treatment|x)=
```

```
The predcited class is
```

```
Out[25]: 'Cleanser'
```

6 Perceptron

```
In [26]: d1 = np.unique(df["Label"])
         d2 = np.unique(df["brand"])
         d3 = np.unique(df["ingredients"])

         def dummy_label(label):
             for i in range(len(d1)):
                 if(label == d1[i]):
                     return i
         dummy_labels = np.vectorize(dummy_label)

         def dummy_brand(brand):
             for i in range(len(d2)):
                 if(brand == d2[i]):
                     return i
         dummy_brands = np.vectorize(dummy_brand)

         def dummy_name(name):
             for i in range(len(d3)):
                 if(name == d3[i]):
                     return i
         dummy_names = np.vectorize(dummy_name)

         def y_value(x,y):
             if(x==y):
                 return 1
             else:
                 return -1
         y_values = np.vectorize(y_value)

         def sigmoid(v):
             def sg(x):
                 return 1/(1+np.exp(-x))
             sg_vect = np.vectorize(sg)
             return sg_vect(v)
```

```
In [27]: x  = ["LA MER","Algae (Seaweed) Extract, Mineral Oil, Petrolatum, Glycerin, Isohexade
         #x  = ["ORIGINS","Water, Glycerin, Butylene Glycol, Rosa Centifolia Flower, Rosa Dama
         def perceptron(df,x):
             x=np.array([dummy_brand(x[0]),dummy_name(x[1]),x[2],x[3]])
             x=x.astype(object)
```

```
            X0=np.asmatrix(np.ones((len(df))))
            brands=np.asmatrix(dummy_brands(df["brand"]))
            names=np.asmatrix(dummy_names(df["ingredients"]))
            price=np.asmatrix((df["price"]))
            rank=np.asmatrix((df["rank"]))
            In=X0.T
            X=np.stack((brands,names,price,rank))

            y_m_vs_all = np.asmatrix(y_values("Moisturizer", df["Label"])).T
            y_c_vs_all = np.asmatrix(y_values("Cleanser", df["Label"])).T
            y_fm_vs_all = np.asmatrix(y_values("Face Mask", df["Label"])).T
            y_t_vs_all = np.asmatrix(y_values("Treatment", df["Label"])).T
            y_ec_vs_all = np.asmatrix(y_values("Eye cream", df["Label"])).T
            y_sp_vs_all = np.asmatrix(y_values("Sun protect", df["Label"])).T

            def gradDescent(X, y):
                        w = np.asmatrix(np.zeros(len(X))).T
                        alpha = 0.0001
                        iter_max = 2000
                        iter = 0
                        while (iter<iter_max):
                            w   = w + alpha*((np.multiply(sigmoid(-np.multiply((X.T).dot(w),y)
                            iter += 1
                        return w

            w_m = gradDescent(X, y_m_vs_all)
            w_c = gradDescent(X, y_c_vs_all)
            w_fm = gradDescent(X, y_fm_vs_all)
            w_t = gradDescent(X, y_t_vs_all)
            w_ec = gradDescent(X, y_ec_vs_all)
            w_sp = gradDescent(X, y_sp_vs_all)
            W = np.column_stack((w_m,w_c,w_fm,w_t,w_ec,w_sp))


            winner = np.argmax(x.dot(W))
            winner_name = d1[winner]
            print("The prediction for" ,x[0], x[1],x[2],x[3], "is", winner_name)

        perceptron(df,x)
```

```
The prediction for 64.0 94.0 175.0 4.1 is Moisturizer
```

7 KNN

```
In [28]: #Correct KNN
         d1 = np.unique(df["Label"])
```

```
d2 = np.unique(df["brand"])
d3 = np.unique(df["ingredients"])

def dummy_label(label):
    for i in range(len(d1)):
        if(label == d1[i]):
            return i
dummy_labels = np.vectorize(dummy_label)

def dummy_brand(brand):
    for i in range(len(d2)):
        if(brand == d2[i]):
            return i
dummy_brands = np.vectorize(dummy_brand)

def dummy_name(name):
    for i in range(len(d3)):
        if(name == d3[i]):
            return i
dummy_names = np.vectorize(dummy_name)
```

In [29]:
```
x  = ["SK-II","Algae (Seaweed) Extract, Mineral Oil, Petrolatum, Glycerin, Isohexadeca
def knn_new(df,x,k):
    diff_brand = np.power(dummy_brands(df['brand'])-dummy_brand(x[0]),2)
    diff_ing = np.power(dummy_names(df['ingredients'])-dummy_name(x[1]),2)
    diff_price = np.power(df['price']-x[2],2)
    diff_rank = np.power(df['rank']-x[3],2)

    distances = np.sqrt(diff_brand+diff_ing+diff_price+diff_rank)
    arg_k_lowest = np.argsort(distances)[0:k]

    candidates_label=[]
    for i in range(k):
        candidates_label.append(df['Label'][arg_k_lowest[i]])

    winner = np.bincount(np.array(dummy_labels(candidates_label))).argmax()
    winner_name = d1[winner]
    print("The predicted label is " , winner_name)

knn_new(df,x,12)
```

```
The predicted label is  Moisturizer
```

8 PCA & TSNE

In [30]:
```
def pairwise_squared_d(X):
    sum_X = np.sum(np.square(X), 1, keepdims=True)
```

```
        suqared_d = sum_X + sum_X.T - 2*np.dot(X, X.T)
        sd = suqared_d.clip(min=0)
        return sd

    def cal_p(suqared_d, sigma, i):
        a = 2*sigma**2
        prob = np.exp(-suqared_d/a)
        prob[i] = 0
        #print(prob)
        prob = prob/np.sum(prob)
        H = -np.sum([p*np.log2(p) for p in prob if p!=0])
        perp = 2**H
        #print(H)
        return prob,perp

    def search_sigma(x, i, PERPLEXITY, tol):
        # binary search
        sigma_min, sigma_max = 0, np.inf
        prob,perp = cal_p(x, sigma[i], i)
        perp_diff = PERPLEXITY - perp
        times = 0
        hit_upper_limit = False
        while (abs(perp_diff) > tol) and (times<50):
            #print(perp_diff, sigma_min, sigma_max)
            if perp_diff > 0:
                if hit_upper_limit:
                    sigma_min = sigma[i]
                    sigma[i] = (sigma_min + sigma_max)/2
                else:
                    sigma_min, sigma_max = sigma[i], sigma[i]*2
                    sigma[i] = sigma_max
            else:
                sigma_max = sigma[i]
                sigma[i] = (sigma_min + sigma_max) / 2
                hit_upper_limit = True
            prob,perp = cal_p(x, sigma[i], i)
            perp_diff = PERPLEXITY - perp
            times = times + 1
        #print(times) typically around 20 when tol=1e-4
        return prob

In [31]: def get_prob(X, PERPLEXITY=30, tol=1e-4):
        n = X.shape[0]
        squared_d = pairwise_squared_d(X)
        squared_d = squared_d/np.std(squared_d, axis=-1)*10
        # init
        pairwise_prob = np.zeros((n,n))
        global sigma
```

```
        sigma = np.ones(n)

        for i in range(n):
            x = squared_d[i]
            prob = search_sigma(x, i, PERPLEXITY, tol)
            pairwise_prob[i] = prob
            if i%100 == 0:
                print("processed %s of total %s points"%(i,n))
            #print(pairwise_prob)
        return pairwise_prob

    def pca(x, n_components=None):
        print("Preprocessing the data using PCA...")
        vec, val = np.linalg.eig(np.dot(x.T, x))
        assert np.alltrue(np.imag(val)) == False
        if n_components:
            return np.real(np.dot(x, val[:,0:n_components]))
        else:
            v_p = vec/sum(vec)
            v_s, i = 0, 0
            while v_s < 0.8:
                v_s += v_p[i]
                i += 1
            return np.real(np.dot(x, val[:,0:i]))
```

In [32]:
```
from collections import defaultdict

tsne_Di = defaultdict(list)

def runTSNE(A,learning_rate, momentum, no_dims, max_iter,n):
    key_ = str(learning_rate)+'__'+str(momentum)

    # randomly assign initial values to y
    y_ = np.random.normal(loc=0,scale=0.01,size=(n,no_dims))
    li = tsne_Di[key_]

    v = 0
    #print("Cross entropy:")
    for iter in range(max_iter):
        y_s_dist = pairwise_squared_d(y_)
        q = 1/(1+y_s_dist)
        np.fill_diagonal(q,0)
        Q = q/np.sum(q, axis=1, keepdims=True)
        y_f = y_.flatten()
        d = y_f.reshape(no_dims, n, 1, order='F') - y_f.reshape(no_dims, 1, n, order=

        CE = -A* np.log2(Q)
        np.fill_diagonal(CE, 0)
```

```
            if iter%2==0:
                li.append(y_.copy())

            gd = 4*(A-Q)*q*d
            gradient = np.sum(gd, axis=2).T

            v = learning_rate*gradient + momentum*v
            y_ = y_ - v
        return li
```

In [33]:
```
cosm_2 = pd.read_csv("cosmetics.csv")
# All possible combinations for the option choices
option_1 = cosm_2.Label.unique().tolist()
option_2 = cosm_2.columns[6:].tolist()


## defining a function embedding ingredients and decomposition at once
def my_recommender(op_1, op_2):
    df = cosm_2[cosm_2['Label'] == op_1][cosm_2[op_2] == 1]
    df = df.reset_index()

    # embedding each ingredients
    ingredient_idx = {}
    corpus = []
    idx = 0

    for i in range(len(df)):
        ingreds = df['ingredients'][i]
        ingreds = ingreds.lower()
        tokens = ingreds.split(', ')
        corpus.append(tokens)
        for ingredient in tokens:
            if ingredient not in ingredient_idx:
                ingredient_idx[ingredient] = idx
                idx += 1

    # Get the number of items and tokens
    M = len(df)                 # The number of the items
    N = len(ingredient_idx)     # The number of the ingredients

    # Initialize a matrix of zeros
    X = np.zeros(shape = (M, N))

    # Define the oh_encoder function
    def oh_encoder(tokens):
        x = np.ones(N)
        for t in tokens:
```

```
                # Get the index for each ingredient
                idx = ingredient_idx[t]
                # Put 1 at the corresponding indices
                x[idx] = 2
            return x

        # Make a document-term matrix
        i = 0
        for tokens in corpus:
            X[i, :] = oh_encoder(tokens)
            i += 1


        #pca
        scale = lambda x: np.nan_to_num((x-np.mean(x,axis=0))/np.std(x,axis=0), 0)

        P =get_prob(pca(scale(X)),30, 1e-2)


        P = P + np.transpose(P)


        P = P / (2)

        #TSNE
        no_dims = 2
        max_iter = 200
        learning_rate = 0.6
        momentum = 0.8

        l1 = runTSNE(P,learning_rate, momentum, no_dims, max_iter,len(X))

        # Make X, Y columns
        df['X'] = l1[-1][:, 0]
        df['Y'] = l1[-1][:, 1]

        return df

    # Create the dataframe for all combinations
    df_all = pd.DataFrame()
    for op_1 in option_1:
        for op_2 in option_2:
                temp = my_recommender(op_1, op_2)
                temp['Label'] = op_1 + '_' + op_2
                df_all = pd.concat([df_all, temp])
```

```
/anaconda3/lib/python3.7/site-packages/ipykernel_launcher.py:9: UserWarning: Boolean Series ke
  if __name__ == '__main__':
```

```
Preprocessing the data using PCA...
processed 0 of total 199 points


/anaconda3/lib/python3.7/site-packages/ipykernel_launcher.py:12: RuntimeWarning: invalid value
  if sys.path[0] == '':


processed 100 of total 199 points


/anaconda3/lib/python3.7/site-packages/ipykernel_launcher.py:22: RuntimeWarning: divide by zer
/anaconda3/lib/python3.7/site-packages/ipykernel_launcher.py:22: RuntimeWarning: invalid value


Preprocessing the data using PCA...
processed 0 of total 190 points
processed 100 of total 190 points
Preprocessing the data using PCA...
processed 0 of total 204 points
processed 100 of total 204 points
processed 200 of total 204 points
Preprocessing the data using PCA...
processed 0 of total 179 points
processed 100 of total 179 points
Preprocessing the data using PCA...
processed 0 of total 162 points
processed 100 of total 162 points
Preprocessing the data using PCA...
processed 0 of total 161 points
processed 100 of total 161 points
Preprocessing the data using PCA...
processed 0 of total 140 points
processed 100 of total 140 points
Preprocessing the data using PCA...
processed 0 of total 156 points
processed 100 of total 156 points
Preprocessing the data using PCA...
processed 0 of total 155 points
processed 100 of total 155 points
Preprocessing the data using PCA...
processed 0 of total 122 points
processed 100 of total 122 points
Preprocessing the data using PCA...
processed 0 of total 177 points
processed 100 of total 177 points
Preprocessing the data using PCA...
```

```
processed 0 of total 162 points
processed 100 of total 162 points
Preprocessing the data using PCA...
processed 0 of total 172 points
processed 100 of total 172 points
Preprocessing the data using PCA...
processed 0 of total 166 points
processed 100 of total 166 points
Preprocessing the data using PCA...
processed 0 of total 122 points
processed 100 of total 122 points
Preprocessing the data using PCA...
processed 0 of total 202 points
processed 100 of total 202 points
processed 200 of total 202 points
Preprocessing the data using PCA...
processed 0 of total 189 points
processed 100 of total 189 points
Preprocessing the data using PCA...
processed 0 of total 200 points
processed 100 of total 200 points
Preprocessing the data using PCA...
processed 0 of total 184 points
processed 100 of total 184 points
Preprocessing the data using PCA...
processed 0 of total 159 points
processed 100 of total 159 points
Preprocessing the data using PCA...
processed 0 of total 131 points
processed 100 of total 131 points
Preprocessing the data using PCA...
processed 0 of total 134 points
processed 100 of total 134 points
Preprocessing the data using PCA...
processed 0 of total 133 points
processed 100 of total 133 points
Preprocessing the data using PCA...
processed 0 of total 125 points
processed 100 of total 125 points
Preprocessing the data using PCA...
processed 0 of total 113 points
processed 100 of total 113 points
Preprocessing the data using PCA...
processed 0 of total 96 points
Preprocessing the data using PCA...
processed 0 of total 89 points
Preprocessing the data using PCA...
processed 0 of total 95 points
```

```
Preprocessing the data using PCA...
processed 0 of total 85 points
Preprocessing the data using PCA...
processed 0 of total 78 points
```

```
In [34]: pd.set_option('display.max_rows', None)
         pd.set_option('display.max_columns', None)
         pd.set_option('display.width', None)
         pd.set_option('display.max_colwidth', -1)
```

```
In [43]: df_all = df_all.dropna(how='any',axis=0)
```

9 Implementing Bokeh for Visualization

```
In [36]: from bokeh.io import show, curdoc, output_notebook, push_notebook
         from bokeh.plotting import figure
         from bokeh.models import ColumnDataSource, HoverTool, Select, Paragraph, TextInput
         from bokeh.layouts import widgetbox, column, row
         from ipywidgets import interact
```

```
In [44]: output_notebook()
```

```
In [38]: # make a source and scatter bokeh plot
         source = ColumnDataSource(df_all)
         plot = figure(x_axis_label = 'T-SNE 1', y_axis_label = 'T-SNE 2',
                       width = 500, height = 400)
         plot.circle(x = 'X', y = 'Y', source = source,
                     size = 10, color = '#FF7373', alpha = .8)

         plot.background_fill_color = "beige"
         plot.background_fill_alpha = 0.2

         # add hover tool
         hover = HoverTool(tooltips = [
                 ('Item', '@name'),
                 ('brand', '@brand'),
                 ('Price', '$ @price'),
                 ('Rank', '@rank')])
         plot.add_tools(hover)
```

```
In [39]: # define the callback
         def update(op1 = option_1[0], op2 = option_2[0]):
             a_b = op1 + '_' + op2
             new_data = {
                 'X' : df_all[df_all['Label'] == a_b]['X'],
                 'Y' : df_all[df_all['Label'] == a_b]['Y'],
                 'name' : df_all[df_all['Label'] == a_b]['name'],
```

```
                'brand' : df_all[df_all['Label'] == a_b]['brand'],
                'price' : df_all[df_all['Label'] == a_b]['price'],
                'rank' : df_all[df_all['Label'] == a_b]['rank'],
            }
            source.data = new_data
            push_notebook()
```

In [40]:
```
# interact the plot with callback
output_notebook()

interact(update, op1 = option_1, op2 = option_2)
show(plot, notebook_handle = True)
```

```
interactive(children=(Dropdown(description='op1', options=('Moisturizer', 'Cleanser', 'Treatmer
```

Out[40]:
```
<bokeh.io.notebook.CommsHandle at 0x1a1c10e780>
```

In [ ]: