Building an AI-Powered Outfit Recommendation System With Dataiku

November 9, 2021

Use Cases & Projects, Dataiku Product
Emma Huang

Ever have trouble deciding what to wear in the morning? Well, it may not be a problem for much longer! Thanks to this computer vision-based outfit recommendation system, don't waste those precious minutes deciding what to wear, let AI do it for you!

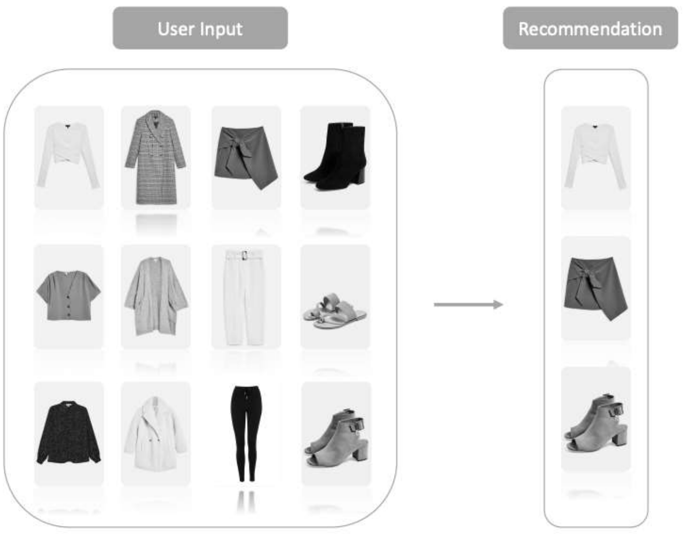
This article will cover how I approached building out a fashion recommendation system as well as some basics on how you might approach building a computer vision project with deep learning. This particular system takes separate articles of clothing as image inputs and recommends what it considers to be appealing outfits. It is built from a combination of the following models: a convolutional neural network (CNN) autoencoder, a multi-input CNN, and K-Means clustering. The final user interface is created with a Dataiku application (/dataiku-applications-a-boost-to-collaboration-and-reuse).

→ Learn more about Recommendation Engines **(https://blog.dataiku.com/cs/c/?cta_guid=56d77261-7c71-4c44-b159-026d170ef81e&signature=AAH58kGeuHAlYJsQn13oCedb0SJTNBnLBQ&pageId=59281588109&placement_guid=47063f78-a835-44b1-83c4-d54659240c45&click=455e4cb6-ca54-40e1-aa5c-c9e9eae8c36d&hsutk=86ec773a2710a2f711b1166a572467ac&canon=https%3A%2F%2Fblog.dataiku.com%2Foutfit-recommendation-system&portal_id=2123903&redirect_url=APefjpFTZ7VifxC3v3DcsLKxuPm2dTkNZ9iOceejE5Er2g_ebWMMzWO1EV6Oc13M2n0Q2zU2QhCS18LWUfJH post)**

Project Overview

In order to begin any machine learning project, we need to start with some data! In this case, I used the Polyvore dataset (https://github.com/xthan/polyvore-dataset). Polyvore is a fashion website where users upload images of outfits and each outfit receives likes from other users. The data consists of folders of outfit images as well as JSONs containing metadata for each outfit (i.e., the number of likes that each outfit received and the category of clothing for each component of the outfit).

Since each outfit has a score (the number of likes), we can generate all combinations of outfits in a particular user's closet, predict the number of likes each outfit would receive, and return the outfit with the highest number of likes as predicted outfit recommendations.



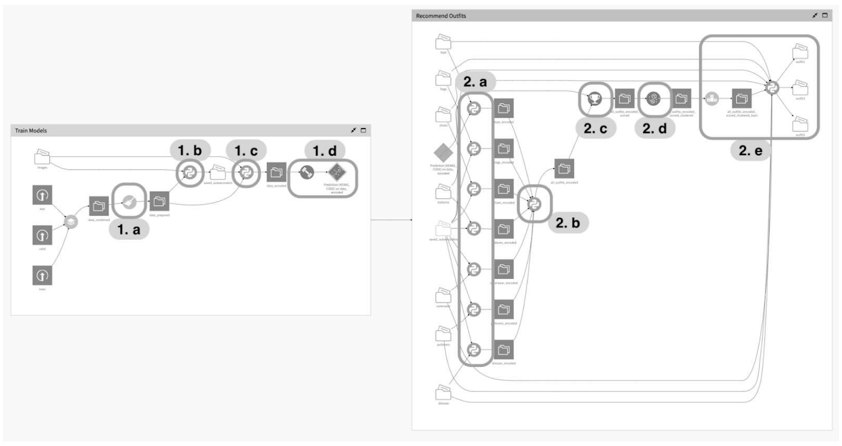
The final project flow is a little more complicated. We can break it down into two general sections:

1. Model Training

- Use the outfit descriptions to extract only the images that match articles of clothing in our definition of an "outfit."
- Train an autoencoder model for each type of clothing in our defined "outfit."
- Predict embeddings for each input image.
- Train a multi-input CNN to predict the likes for each outfit.

2. Outfit Recommendation

- Input end user wardrobe and predict the embedding for each image.
- Generate all possible combinations of outfits using the images.
- Score each possible outfit using the CNN.
- Create three clusters of outfits based on the embeddings.
- Return the outfit with the top score for each cluster as predictions.



Initial Data Cleaning

The Polyvore dataset includes three JSONs (test, train, and validation) with metadata for each outfit. We can import these JSONs as datasets in Dataiku to obtain the paths to relevant images from each outfit. Because Dataiku automatically handles test/train splits, I combined all the data before cleaning it in a prepare recipe.

In the prepare recipe, I removed all outfits that weren't relevant to the problem by using the "filter on value" processor. For example, I removed all outfits that contained "A menswear look" in the description to make sure we were only looking at women's fashion for this project.

Next, I needed to define what I would consider an “outfit.” In this project I considered two basic types of outfits:

Type 1:

- Dress
- Outerwear (optional)
- Shoes
- Bag

Type 2:

- Top
- Bottom
- Pullover (optional)
- Outerwear (optional)
- Shoes
- Bag

I used a series of Python steps in my prepare recipe to remove any outfits that didn’t match the two outfit criteria above. For example, if an outfit contained a dress but no shoes, I would drop it from the dataset. Finally, I created a column containing the path to the actual image for each article of clothing for each outfit, which we can reference to build the models.

Image Processing for Computer Vision

Most machine learning models can only take numerical inputs, so we need a way to convert our .jpg files to a format that a model can read. In this case, I used the Python package OpenCV-Python, which contains pre-built OpenCV with dependencies and Python bindings. OpenCV is an open source library for computer vision, machine learning, and image processing and can be accessed through a variety of different programming languages. OpenCV allows you to read images as bytes in Python through the methods cv2.imread(`image_path`) and cv2.imdecode(`image_path`).

When an image is read into Python, it can be represented as either a 2D or 3D matrix, depending on the number of color channels present in the image. For example, if you have a black and white image that is 200x300 pixels, it will be represented by a 2D array of shape (200,300). Each point on the matrix will be a number from 1 to 255 representing the brightness of that pixel. If you have a 200x300 pixel color image, it will be represented by a 3D array of shape (200,300,3). Each of the channels in this matrix represents red, green, or blue, and each point of the channel will again be a number from 1 to 255 representing the brightness of that pixel.

When working with image data, just like in any other machine learning problem, the data should be normalized. The models that these images will be fed to will need inputs of universal sizes, so I could either resize the images to all be the same dimensions or pad the images to match the size of the largest image. In this case, I started by using another cv2 method to resize all images to the same dimensions. After that, I simply divided the image arrays by 255 so each point would be constrained to a scale from 0 to 1. The images are now ready to be fed into a machine learning algorithm!

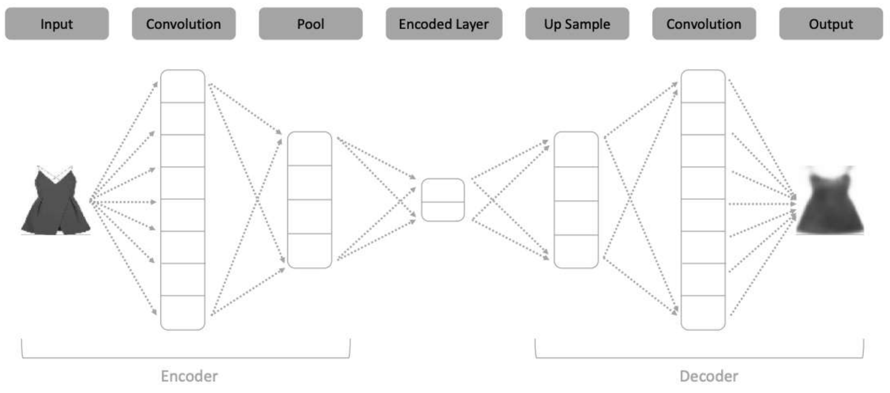
```
def prepare_image(fpath):
    with images.get_download_stream(fpath) as f:
        data = f.read()
    nparr = np.fromstring(data, np.uint8)
    img_np = cv2.imdecode(nparr, cv2.IMREAD_COLOR)
    im = cv2.resize(img_np, (64, 64))
    im = im/255
    return im
```

Dimensionality Reduction With an Autoencoder Model

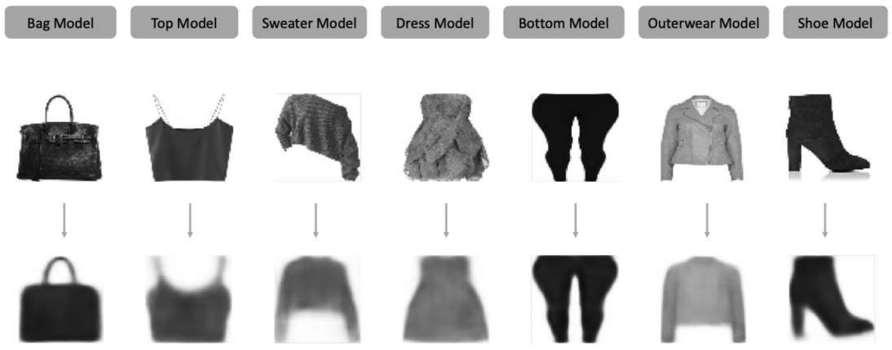
Before training the predictive model, I decided to train a model to reduce the dimensionality (/dimensionality-reduction-how-it-works-in-plain-english) of the images to feed into concurrent models. This has two benefits: the first is that we can work with smaller representations of images, which increases the efficiency. The second benefit is that the concurrent models can learn additional signals from the autoencoder that the original image doesn’t contain.

So what actually is an autoencoder? It’s a structure used in artificial neural networks type algorithms used to learn efficient codings of data. You can learn more about deep learning here (/ai-vs.-machine-learning-vs.-deep-learning) and we have a guidebook on the topic here (https://pages.dataiku.com/deep-learning). While working with neural networks, the modeler defines a variety of layers. In the case of autoencoders, each layer will decrease the dimensionality of the original input up until the smallest layer, which is taken as the encoding of the input.

Then the same process is reversed, increasing the dimensionality up until the output is the same size as the input. The model is actually trying to predict the input (in other words, the target is the same object as the features). Because of this, it's actually quite easy to visualize how well the model is working. The better the model, the closer the output images will match the inputs.



For this project, I trained a separate model for each article of clothing, so that each model has a general understanding of its specific article. You can see that the model is actually able to generalize the clothing. For example, the sweater model was able to understand what the slouchy gray sweater would look like straight on. This shows how feeding the embeddings into the concurrent models can reduce some of the noise from the variety of input images.

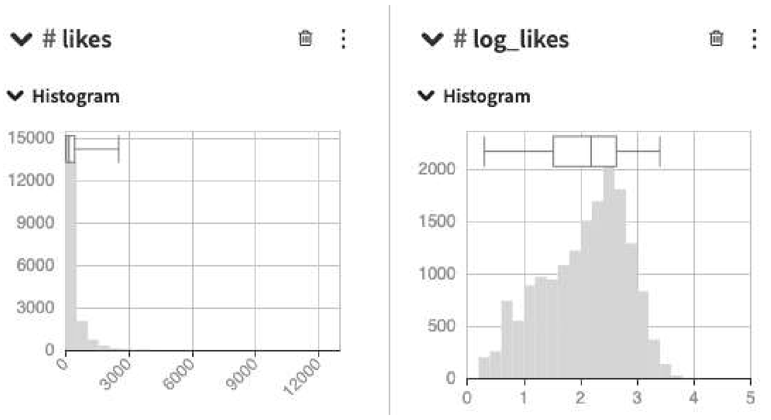


Typically computer vision problems involve neural networks, but not necessarily. You can build decent models using almost any algorithm, even something like logistic regression. In this case, the project is more complex, so I used a series of neural networks. I built the autoencoder with a CNN, since we typically use this type of neural network while working with images. I trained the autoencoder models in a Python recipe and saved the model weights in a managed folder to call them at other points in the flow.

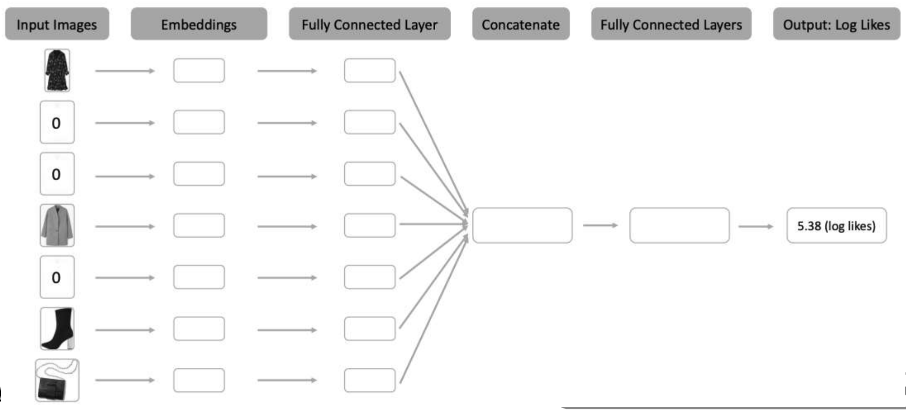
```
clothing_list = ["dress","top","pullover","outerwear","bottom","shoe","bag"]
for clothing in clothing_list:
    features = get_data(list(df[clothing+"_path"]),df)

    input_layer = Input(shape=(64,64,3))
    # encoder
    conv1 = Conv2D(496, (3, 3), activation='relu', padding='same')(input_layer)
    pool1 = MaxPooling2D(pool_size=(2, 2))(conv1)
    conv2 = Conv2D(248, (3, 3), activation='relu', padding='same')(pool1)
    pool2 = MaxPooling2D(pool_size=(2, 2))(conv2)
    conv3 = Conv2D(124, (3, 3), activation='relu', padding='same')(pool2)
    pool3 = MaxPooling2D(pool_size=(2, 2))(conv3)
    conv4 = Conv2D(64, (3, 3), activation='relu', padding='same')(pool3)
    pool4 = MaxPooling2D(pool_size=(2, 2))(conv4)
    conv5 = Conv2D(32, (3, 3), activation='relu', padding='same')(pool4)
    pool5 = MaxPooling2D((2, 2), padding='same')(conv5)
    conv6 = Conv2D(16, (3, 3), activation='relu', padding='same')(pool5)
    pool6 = MaxPooling2D((2, 2), padding='same')(conv6)
    conv7 = Conv2D(16, (3, 3), activation='relu', padding='same')(pool6)
    pool7 = MaxPooling2D((2, 2), padding='same')(conv7)

    encoded = Flatten(name = 'encoded')(pool7) # Embedded layer

    #decoder
    conv8 = Conv2D(16, (3, 3), activation='relu', padding='same')(pool7)
    up1 = UpSampling2D((2,2))(conv8)
    conv9 = Conv2D(16, (3, 3), activation='relu', padding='same')(up1)
    up2 = UpSampling2D((2,2))(conv9)
    conv10 = Conv2D(32, (3, 3), activation='relu', padding='same')(up2)
    up3 = UpSampling2D((2,2))(conv10)
    conv11 = Conv2D(64, (3, 3), activation='relu', padding='same')(up3)
    up4 = UpSampling2D((2,2))(conv11)
    conv12 = Conv2D(124, (3, 3), activation='relu', padding='same')(up4)
    up5 = UpSampling2D((2,2))(conv12)
    conv13 = Conv2D(248, (3, 3), activation='relu', padding='same')(up5)
    up6 = UpSampling2D((1, 1))(conv13)
    conv14 = Conv2D(496, (3, 3), activation='relu', padding='same')(up6)
    up7 = UpSampling2D((2, 2))(conv14)
    decoded = Convolution2D(3, 1, 1, activation='sigmoid', padding='same')(up7)

    autoencoder = Model(input_layer, decoded)
    autoencoder.compile(optimizer='adadelta', loss='binary_crossentropy')
    encoder = Model(input_layer, encoded)

    early_stopping = EarlyStopping(monitor='val_loss', min_delta=0, patience=10, verbose=1, mode='auto')
    print("Fitting "+clothing)
    autoencoder.fit(features, features, epochs=20, batch_size=100, validation_split=.15, callbacks=[early_stopping])

    weights= encoder.get_weights()
    with saved_autoencoders.get_writer(clothing+"_model_weights.pkl") as w:
        pickle.dump(weights,w)

    del(autoencoder)
    del(encoder)
    keras.backend.clear_session()
```

Predict Outfit Scores With a Multi-Input CNN Model

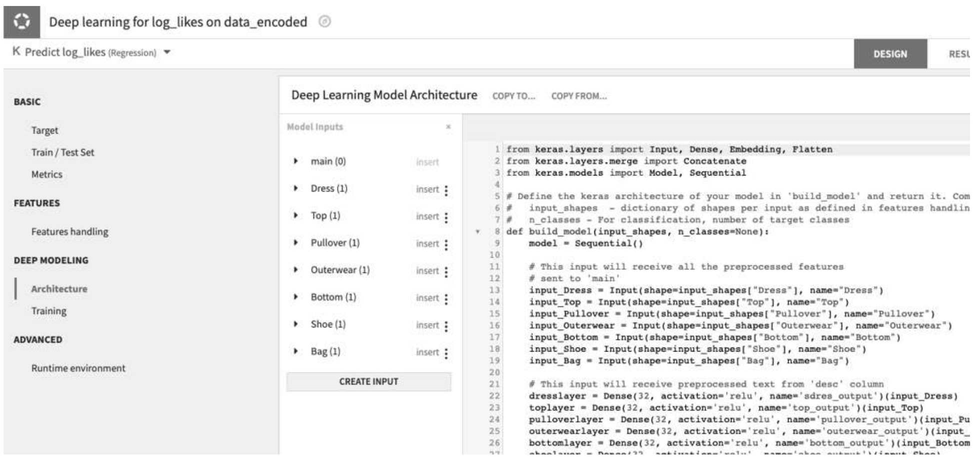
Now that we have the autoencoder model, we can use the encodings to train a predictive model. This model takes separate articles of clothing as inputs and predicts the log likes that outfit would receive. I chose to predict the log likes in this case rather than the actual likes because the distribution of likes was incredibly skewed. Taking the log provides a more normal distribution.



Each image's encoding was passed individually to fully connected layers before they were all concatenated together and passed through an additional fully connected layer, before predicting the output. In this way, each image was evaluated separately and together as a whole in the outfit.



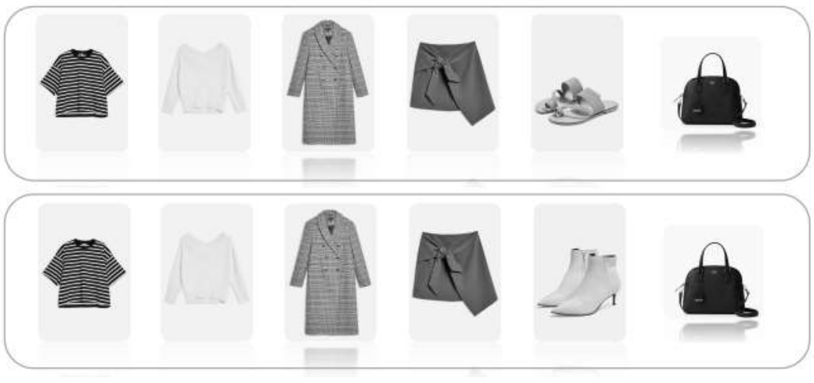
To train the model, I used the deep learning prediction visual machine learning lab in Dataiku. I was able to use each of my features (each article of clothing) as a separate input and inserted them directly into my model. Once the model was trained, I published it to the flow so I could predict scores on new users' outfits.



```
from keras.layers import Input, Dense, Embedding, Flatten
from keras.layers.merge import Concatenate
from keras.models import Model, Sequential

# Define the keras architecture of your model in 'build_model' and return it. Com
#   input_shapes  - dictionary of shapes per input as defined in features handlin
#   n_classes - For classification, number of target classes
def build_model(input_shapes, n_classes=None):
    model = Sequential()

    # This input will receive all the preprocessed features
    # sent to 'main'
    input_Dress = Input(shape=input_shapes["Dress"], name="Dress")
    input_Top = Input(shape=input_shapes["Top"], name="Top")
    input_Pullover = Input(shape=input_shapes["Pullover"], name="Pullover")
    input_Outerwear = Input(shape=input_shapes["Outerwear"], name="Outerwear")
    input_Bottom = Input(shape=input_shapes["Bottom"], name="Bottom")
    input_Shoe = Input(shape=input_shapes["Shoe"], name="Shoe")
    input_Bag = Input(shape=input_shapes["Bag"], name="Bag")

    # This input will receive preprocessed text from 'desc' column
    dresslayer = Dense(32, activation='relu', name='sdres_output')(input_Dress)
    toplayer = Dense(32, activation='relu', name='top_output')(input_Top)
    pulloverlayer = Dense(32, activation='relu', name='pullover_output')(input_Pu
    outerwearlayer = Dense(32, activation='relu', name='outerwear_output')(input_
    bottomlayer = Dense(32, activation='relu', name='bottom_output')(input_Bottom
```

Putting It Together With Clustering Algorithms

Using the trained models, we are finally able to create some new outfit recommendations! Given the new clothing input by the user, the first step was to read the images using OpenCV and then encode them using the autoencoder. After that, I created all possible combinations of outfits using the definition of "outfit" that I defined above. Next, I used the predictive model to predict the score that each outfit would receive. We could stop there and just return the outfits with the top scores. However, when I tried that method, I found that the top outfits were almost all the same with just some variation on the shoes or bag.

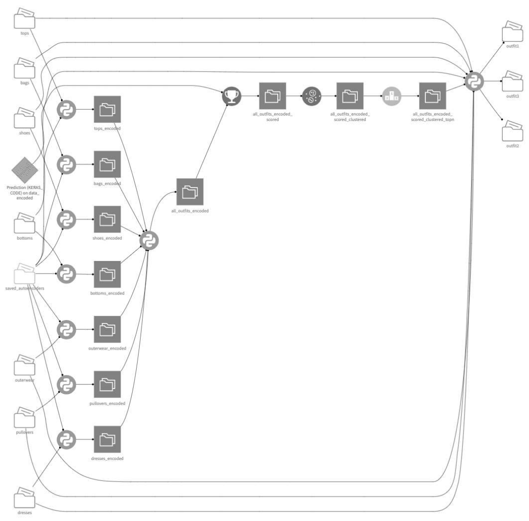


In order to ensure that the outfits were relatively different from each other, I decided to cluster the outfits using the embeddings and return the top outfit from each cluster. This is another advantage of using the embeddings, as the dimensionality is extremely reduced for the clustering step. You can learn more about clustering and how the algorithms work here (https://knowledge.dataiku.com/latest/courses/intro-to-ml/clustering/clustering-summary.html). In this case I used K-Means clustering to find three clusters (for three outfit recommendations).

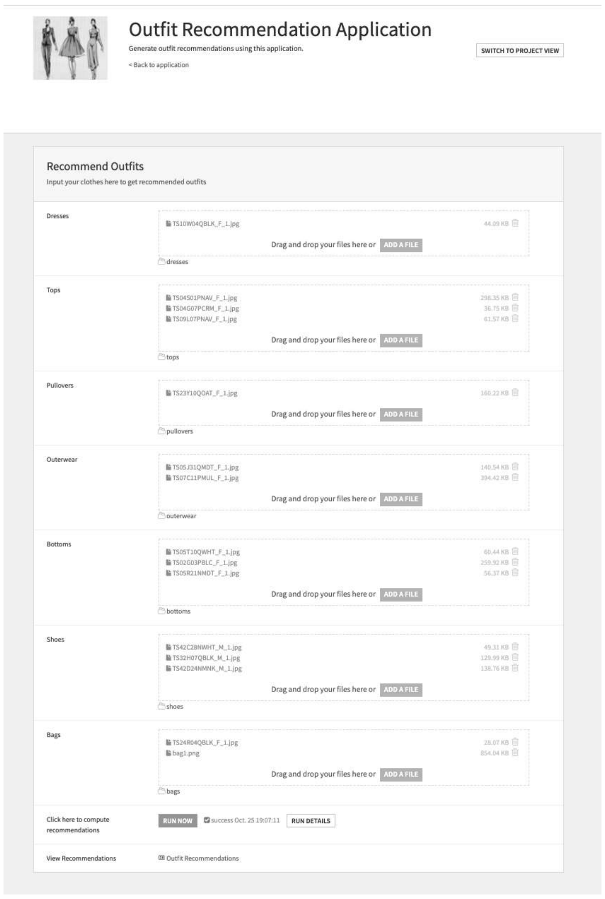
I created a clustering lab to initialize the settings and deployed the algorithm as a recipe to perform clustering so that each time the flow is run with new images, new clusters will be formed accordingly. After a cluster is identified for each potential outfit, the Top N recipe can be used to find the top scoring outfit for each cluster. Finally, I added the images from each top scoring outfit to a folder using a python recipe. The recommendations are complete.

End User Interface

Now that the entire pipeline was complete, I wanted to build an interface that an end user would actually be able to interact with. Thankfully, using Dataiku applications, this was actually really easy! I started by creating a scenario that could automate the recommendation flow zone (the autoencoder and predictive models don't need to be retrained to make recommendations). The scenario triggers the recommended outfits to rebuild from the user's input images onward.



Next, I built a Dataiku application for an end user interface. The end user will be able to drag their photos into the application, click run, and then see their recommendations in a dashboard!


Outfit Recommendation Application

Generate outfit recommendations using this application.

< Back to application

SWITCH TO PROJECT VIEW

Recommend Outfits

Input your clothes here to get recommended outfits

Dresses
TS10W04QBLK_F_1.jpg 44.09 KB
Drag and drop your files here or ADD A FILE
dresses

Tops
TS04S01PNAV_F_1.jpg 298.35 KB
TS04G07PCRM_F_1.jpg 36.75 KB
TS09L07PNAV_F_1.jpg 61.57 KB
Drag and drop your files here or ADD A FILE
tops

Pullovers
TS23Y10QOAT_F_1.jpg 160.22 KB
Drag and drop your files here or ADD A FILE
pullovers

Outerwear
TS05J31QMDT_F_1.jpg 140.54 KB
TS07C11PMUL_F_1.jpg 394.42 KB
Drag and drop your files here or ADD A FILE
outerwear

Bottoms
TS05T10QWHT_F_1.jpg 60.44 KB
TS02G03PBLC_F_1.jpg 259.92 KB
TS05R21NMDT_F_1.jpg 56.37 KB
Drag and drop your files here or ADD A FILE
bottoms

Shoes
TS42C28NWHT_M_1.jpg 49.31 KB
TS32H07QBLK_M_1.jpg 129.99 KB
TS42D24NMNK_M_1.jpg 138.76 KB
Drag and drop your files here or ADD A FILE
shoes

Bags
TS24R04QBLK_F_1.jpg 28.07 KB
bag1.png 854.04 KB
Drag and drop your files here or ADD A FILE
bags

Click here to compute recommendations
RUN NOW ✓ success Oct. 25 19:07:11 RUN DETAILS

View Recommendations
Outfit Recommendations

Results and Further Applications

With the application done, we can finally get to recommending outfits and reviewing the model on real examples. I used an assortment of images I found online to test my system, and here were a couple outfits it returned:

CONTACT US (HTTPS://WWW.DATAIKU.COM/HOME/CONTACT-US/) **GET STARTED (HTTPS://WWW.DATAIKU.COM/PRODUCT/GET-STARTED/)**

This is an unsupervised problem and fashion is pretty subjective, so it's tough to really evaluate what a "good" or "bad" model looks like. I would say that my measure is whether or not I would personally wear the outfits that the system recommends, which in this case is a resounding yes.

There are many extensions of this project, such as recommending articles of clothing to users to enhance their existing wardrobe or dressing store mannequins. If nothing else, it can at least help you get dressed in the morning when it feels like you have nothing to wear.

Go Further on Recommendation Engines

Want to build your own recommendation system? In this ebook, uncover how they're built and a step-by-step walkthrough (with code samples for advanced users) to build your own.

You May Also Like

(https://blog.dataiku.com/data-visualization-tools-in-plain-english)

April 17, 2023

Dataiku Product, Featured, In Plain English | Catie Grasso

Data Visualization Tools (In Plain English!) (https://blog.dataiku.com/data-visualization-tools-in-plain-english)

READ MORE (HTTPS://BLOG.DATAIKU.COM/DATA-VISUALIZATION-TOOLS-IN-PLAIN-ENGLISH)

(https://blog.dataiku.com/llms-in-the-enterprise)

April 13, 2023

Use Cases & Projects, Featured | Kurt Muehmel

How to Use Large Language Models in the Enterprise (https://blog.dataiku.com/llms-in-the-enterprise)

READ MORE (HTTPS://BLOG.DATAIKU.COM/LLMS-IN-THE-ENTERPRISE)

(https://blog.dataiku.com/governance-how-to)

April 12, 2023

Use Cases & Projects, Scaling AI, Featured | Patrick Peinoit

Governance: What It Is and How to Do It Right (https://blog.dataiku.com/governance-how-to)

READ MORE (HTTPS://BLOG.DATAIKU.COM/GOVERNANCE-HOW-TO)

CONTACT US (HTTPS://WWW.DATAIKU.COM/HOME/CONTACT-US/) **GET STARTED (HTTPS://WWW.DATAIKU.COM/PRODUCT/GET-STARTED/)**

(https://blog.dataiku.com/ready-set-bake)

April 12, 2023

Use Cases & Projects, Dataiku Product, Featured | Christina Hsiao

Ready, Set, BAKE! (https://blog.dataiku.com/ready-set-bake)

READ MORE (HTTPS://BLOG.DATAIKU.COM/READY-SET-BAKE)

CONTACT US (HTTPS://WWW.DATAIKU.COM/HOME/CONTACT-US/) **GET STARTED (HTTPS://WWW.DATAIKU.COM/PRODUCT/GET-STARTED/)**