Designing Image Recommendation System for Search Purpose

Author 1	Author 2	Author 3	Author 4
Jiachen Zhao	Shiwei Zhou	Xianya Xiong	Yilin Jin
j1zhao@ucsd.edu	s1zhou@eng.ucsd.edu	x1xiong@ucsd.edu	yij008@ucsd.edu

Abstract

Search problem has been studied for many years and has been practically solved only in recent decades. Google was founded 20 years ago and spent numerous amounts of resources in advancing its algorithm and hardware, so that we can have a powerful search engine that completes our search query in milliseconds. Google Image Reverse Search was introduced only 10 years ago, where search query allows not only words but also images. However, not everyone owns the same advanced infrastructure and hardware as Google have after years of development. What if someone asks "who is the creator of this artwork" when Google fails at the moment. Hence, in this paper, we attempted to reshape a common search problem to make it less costly and explored solutions that minimal infrastructure can afford.

As for the most basic search problem, the most basic solution would cost O(n) for linear search through the search domain if data is unsorted or O(log(n)) for sorted data using algorithm like binary search. For multiple words search, data domain is normally not sorted and it can be extremely costly if data domain is very large. Hence, instead of seeing this as a search problem, we reshape search problem into recommendation problem. Instead of asking "who is the creator of this artwork", we reshape the question into "recommend me the artists who are most likely to create this kind of artwork". When the accuracy of recommendation model is high enough, it essentially outputs the same results as a search model.

In this paper, we broke down our recommendation model into 2 tasks. The first task will be predicting label based on categorical information. The second task will be predicting label based on the feature extracted from feature images. Those 2 tasks attempt to reassemble the functionality of Google Search and also Google Image Reverse Search.

Keywords: Safebooru, Image Features, MLP, Jaccard Similarity, Feature Encoding, Classification, Image Tags

1 Dataset

The dataset used for this project is "Safebooru's image metadata of 2.4 million records" on Kaggle, which consists of 2,443,000 records (except header row) from the data uploaded on Safebooru website until May 26, 2018. This dataset clusters information of artworks and their corresponding creators from Pixiv, which is a platform allows everyone to share their art works. This dataset contains columns:

id, creator_id, created_at, width, height, tags, source, file_url

This dataset is chosen for several reasons. First of all, this dataset contains enough volume of data entries which gives us enough space to filter unqualified data. Secondly, this dataset contains many-to-one schema from *file_url* to *creator_id*, making this dataset viable for training for our task. In other words, multiple artworks are recorded for every single creator, providing solid training data for our training model. Otherwise, lack of many-to-one schema will create very sparse and noisy dataset for our training model. Lastly, this dataset contains no numerical feature data which allows us to test our feature extraction methods.
In this project, we mainly explored the tags column for task 1 and the actual images accessed from *file_url* column for task 2. For task 1, our major challenge will be focusing on extracting features from tags column, where shows all related tags for the corresponding art work. For task 2, our major challenge will be focusing on extracting features from image of *file_url* column, which is essentially downloading link for actual artwork image.

2 Identify Tasks

The final goal of our project is to predict the creator of a certain image, and recommend the closest images. There are two ways of achievement: to predict the creator with image tags, or to predict with image colors. Therefore, our task was divided into two parts:

- Task 1: Given the tags of a certain image, transform tags into feature vectors and predict its creator.
- Task 2: Given an image, produce feature vectors related to its colors, and predict its creator.

Details of these two tasks will be illustrated separately below.

3 Predictive Task 1

For task 1, we look into the tags of the image. Our design is based on an assumption that every creator has his or her own preference in content. For example, some creators are more willing to produce images with sunshine, uniform, and long black hair. As a result, these tags will frequently show up in image records created by this creator.

Figure 1: Tag Frequency Visualization

Therefore, the goal of our model should be to use special tags (e.g., "black_nails") to predict its creator.
In this task, we ran through the total dataset and chose 209,806 records created by creators who own more than 5 images. Among all the records, we chose 90% for training and 10% for validation. Accuracy of both training and validation set will be the key to evaluate our model, and validity of model's predictions can be assessed through accuracy of validation set.

3.1 Literature

The dataset we used is an image metadata extracted using Safebooru API. It is inspired by Alexander Lamson's dataset on Kaggle, which contains 2.7 million rows of tag-based anime image metadata.
It has been used to analyze the correlation between several tags (e.g. Is it really true that blonde hair is related to twin tail), and to classify them into more detailed categories according to their tags.
In our task, the most important two columns we are using for predictive model are creator_id and tags. Former research related with image prediction and tags is mainly focused on automatically produce tags for images. However, there are some previous findings and methods related with feature encoding, which is necessary when processing image tags. Here are some encoding methods mentioned (Hale, 2018; He, 2016):

- Dummy encoding: A categorical variable with k levels will be transformed into k-1 dichotomous variables each with two levels: 0 and 1. For a categorical variable with levels ['A', 'B', 'C'], variable a='A', variable b='B'. These two variables can be transformed into [1,0,0], [0,1,0].

- One-hot encoding: A categorical variable with k levels, each category will be transformed into a vector that contains 1 and 0, denoting whether the feature is present. This method is quite close to dummy encoding, except column numbers it produces will still be k.

- Ordinal encoding: Ordinal encoding assumes that the categories follow certain order and then will assign a natural number for each category, such as 0, 1, 2. However, it may suggest unfavoured correlation between features.

- Hashing: Similar to one-hot-encoding, but with less dimension. Depending on the algorithm and feature dimension, it also contains risk of hash collision.

- Binary encoding: Similar to ordinal encoding, but changing natural number or binary number and assign each digit an independent column. This has the same problem of suggesting wrong correlation between features and also cause more info loss.

3.2 Baseline Model

Here are two types of tags: the tags with underscores '_' and tags without '_'. We call tags without '_' as primary tags (e.g. "hair") and tags with '_' as secondary tags (e.g. "black_hair"). Primary tags are only composed with one keyword, while secondary tags composed with multiple keywords. A baseline model was constructed using primary tags only. We only one-hot-encoded primary tags since they are frequently used.
Here is an example of feature processing:
The original tag set:

> {*1girl blush sunny skirt bow collared_shirt from_behind grey_skirt hair_bow higuchi_kaede leaning_forward long_hair long_sleeves looking_at_viewer looking_back misumi_(macaroni) mole mole_under_eye necktie nijisa ...* }

Corresponding categories (primary tags):

> *sunny, day, skirt, white, blush, hair*

One-hot-encoded feature vector:

> *1, 0, 1, 0, 1, 0*

To build an effective baseline model, several algorithms were tried, including K-nearest Neighbors Classifier, Random Forest and Decision Tree and Logistic Regression. Among them all, Logistic Regression showed the optimal performance. However, this baseline model ended up with a poor accuracy lower than 55%.

3.3 Final Model

The baseline model using primary tags only showed a poor accuracy in both training and validation set, which indicates primary tags are not enough for us to predict the creator.
Besides, a problem was spotted that some data entry had very sparse feature vector due to lack of primary tags but richness in secondary tag. This suggests that we lost too much information in building feature vector in this way. Therefore, those tags should be transformed more efficiently and comprehensively.

Feature Processing

The most reliable way of transforming features is to one-hot-encode every tag. However, the total number of unique tags is huge, and feature dimension can be too high and too sparse. An improved method based on baseline model was raised up: Only one hot encoding primary tags since they are frequently used (Step1); then extract more information from secondary tags without increasing feature dimensions (Step2).

Step1: One-Hot-Encode Dummy Primary Tags

To strengthen the problem exposed in the baseline model, we will create dummy primary tags for those data entries.
Except for the original primary tags, secondary tags can also be broken down into multiple primary tags (e.g: "grey_skirt" =¿ "grey", "skirt"). This process will preserve the same information while losing some correlation among words. We call the new primary tags produced from secondary tags as dummy primary tags.
Now we collect all dummy primary tags and take first a% ranking as primary tags for the art. Once new primary tags are generated for all arts, we can one-hot-encode the feature again and obtain the feature vector.

Step2: Extract Information from Secondary Tags

For step2, we will try to extract more information from secondary tags without increasing feature dimensions. We started with a simple strategy. For all secondary tags for an art, we increment the primary tag code by one if they are related (here we define as primary tag is the sub-string for secondary tag).
Here is an example of feature processing:

Step 1: one-hot encode primary tags

Corresponding categories (primary tags):

sunny, day, skirt, white, blush, hair

Feature vector produced after step1:

1, 0, 1, 0, 1, 0

Step 2: Extract information from secondary tags

Secondary tags:

black hair, black skirt, black shirt

Feature vector produced after step2:

1, 0, 2, 0, 4, 2

Model selection

A few models were used to compare different results, including Decision Tree, Random Forest, K-nearest Neighbors Classifier and Logistic Regression. Pros and cons are listed below:(Jansen, ; Varghese, 2018)

- Dummy encoding: A categorical variable with k levels will be transformed into k-1 dichotomous variables each with two levels: 0 and 1. For a categorical variable with levels ['A', 'B', 'C'], variable a='A', variable b='B'. These two variables can be transformed into [1,0,0], [0,1,0].
- Decision Tree: One most important advantage for decision tree is that there is no requirement on data distribution, and can handle collinearity efficiently. Disadvantage is, tree may grow too complex and cause overfitting.
- Random Forest: The predictive performance of random forest can compete with the best supervised learning algorithms, but also has high computational costs, which will slow down the speed. In this task, we set n_estimators=100.
- K-nearest Neighbors: KNN is easy and simple, with few hyperparameters to tune. However, feasibly moderate sample size is required, and collinearity outliers should be treated before model. (Cunningham and Delany, 2007) In this task, we set n_neighbors=3.
- Logistic Regression: Logistic Regression is easy, simple and fast to do classification, and can be used for multi classification also. Given it's a linear classification, it is valid to use logistic regression model. However, careful feature selection is crucial for LR. In this task, we set penalty='l2',C=1,random_state=423.

Among all the models, Logistic Regression showed the highest accuracy for both training and validation set (as seen below).

Model	Training Accuracy	Validation Accuracy
Decision Tree	0.92	0.85
Random Forest	0.92	0.87
K-nearest Neighbors	0.65	0.54
Logistic Regression	0.93	0.90

Table 1: Accuracy of final models

According to pros and cons, as well as model accuracy, we chose Logistic Regression as our final model. The accuracy for training set is 0.9274, and that of validation set is 0.9023.

3.4 Performance and Results

Application of our predictive model is shown below:

```
[26]: #define your search tag here
search_tag = ['blush','1girl','ponytail']

#recommend one artist
recommendations(search_tag,mod.predict)
```

```
(array([420]),)
encoded index:
recommended artist ID:  ['8970']
related art to the tag
```

Figure 2: Application of the predictive model

Given a search_tag "blush", the artist with ID '8970' is predictive to be the most likely creator. The creator's arts related to this tag are also listed.
The new feature building method we applied in our final model will show more information on certain feature element, which helps improve accuracy to higher than 0.9. This improvement of accuracy can be explained by extra information from secondary tags. For example, for a set of tags, if only primary tags are encoded, the feature vector will be:

1, 0, 1, 0, 1, 1

Corresponding categories (primary tags):

sunny, day, skirt, white, black, hair

However, when the new method is applied, corresponding categories (primary tags):

sunny, day, skirt, white, black, hair

Secondary Tags:

black hair, white skirt, black shirt

Final feature processing results after step 2 would be:

1,0,2,1,3,2

The new feature vector after step 2 implies that there are 3 black-related components in this image. Since 'hair' feature element also has higher value, it is reasonable to assume that this artist has certain specified tags related to "hair", which is "black" color in this case. Given more specific information extracted from tags, it is reasonable that accuracy should improve.

4 Predictive Task 2

We looked into the images themselves and explored what we can use to predict their creator. Before we dig into the images we listed some questions to answer: How can we translate a creator's painting style into features? What kinds of image features are there? Is there a painting pattern that we are able to identify?
We ran through the CSV file and used the safebooru's file url to download the images and built a smaller database of 1500 images first, then a whole dataset of 2,443,000 images. As downloading we noticed that many of the paintings are from the same creators so we shuffled the dataset before downloading and set a threshold for each creator to have 20 pieces in our training dataset. Then we split it into 90% for training and 10% for testing.

4.1 Literature

The idea of how we define image styles and their relationship with creators came from two papers "Recognizing Image Style"(Sergey Karayev,)and "Image Feature Engineering"(Hurford, 2017). Image styles exhibit strong dependence on color and color statistics could be a powerful feature. There are two color extraction methods that we looked into:

- Color statistics in RGB channels:
  RGB color space represents color with three integer values from 0 to 255 to denote the intensity in the three colored lights of red, green and blue.

- Color statistics in L*a*b* channels:
  L*a*b* is a conversion of the same information to a lightness component L*, and two color components - a* and b*. L* defines lightness, a* represents red/green value, and b* the yellow/blue value(RawPedia, 2014). It is more accurate in defining the numerical change in colors since it is designed to correspond to roughly the same amount of visually perceived change.

Although the L*, a*, and b* channels are more accurate in representing colors in a mathematical way, RGB color system is implemented in all color computer monitors which is the best applicable in our case, so we decided to apply RGB in our model. Some of the methods in extracting content-wise features such as GIST, Meta-class binary features and content classifiers were also listed

in the paper. Since all the images in our dataset are animation portraits, content-wise features are not very applicable so we decided to not use them. The state-of-the-art method for image recognition is deep convolutional neural networks, inspired by which, we included Multilayer Perceptron in our models. Our accuracy result is consistent with the existing work that MLP does perform better with images.

4.2 Baseline Model

Feature Processing

First, we developed a baseline model that extracts 10 color features for each image as following:

- image size (width length)
  E.g. (img_size_x = 1100, img_size = 600)
- Overall mean color and standard deviation in RGB
  E.g. (mean_color = 207.98, std_color = 50.38)
- Mean Red / Green / Blue colors and their corresponding standard deviations in RGB
  E.g. (img_blue_mean = 213.65, img_green_mean = 199.37, img_red_mean = 210.93, img_blue_std = 45.16, img_green_std = 53.14, img_red_std = 51.36)

Our assumption is developed as: if two images share similar means and standard deviations in the primary colors, they have higher probability being created by the same author. In other words, we assume that creators often use similar colors in his or her paintings, so by comparing the colors of two painting we can test if they are created by the same person. This assumption is justified by the fact that the drawings are drawn using digital palettes so that each creator has a high probability to have some colors that they use most often, for example these colors could be tagged "favorite" on their palette.
For every image, we extracted 10 features including image size and color statistics in RGB using the 'CV2' package. We obtained the size of the image and color information of each pixel of the image stored in a three-dimensional matrix. As the images are read with three channels by default, red, green and blue, the matrix only has color statistics of these three colors. We then calculated the mean and standard deviation in RGB overall and also for specific colors which made up the 8 color features.

Model Selection

Since our color feature and the predicted creator is a linear relationship, we chose four models from sklearn library to build our classifier: Random Forest, K-nearest Neighbours Classifier, Multilayer Perceptron, and Linear Logistics Regression. Then we calculated their accuracies to test their performance and picked the best one – MLP as our final classifier.

- Linear Logistics Regression
  Since the project we are doing is a classification problem, the first model we came up with was linear logistics regression which predicts the probability of the image if the work of this specific creator. We set the penalty as l2 and C as 1. However, the features of overall mean color in RGB correlates with the features of mean color in red, green and blue. This model has limitations dealing with multicollinearity between features.

- Random Forest
  Random Forest is a bagging algorithm that draws a random bootstrap samples and features to train individual decision trees. Each individual tree classifies first and the class with the most votes becomes our prediction. Random forest operates fast as every tree in the model is processing independently and simultaneously. However, overfitting can easily occur in this model. As it shows in the accuracy table below, random forest has an accuracy of 0.97 in training set but only 0.45 in validation set.

- K-nearest Neighbours Classifier
  KNN classifier defines similarity as the distance between data points; the closer they are, the more similar they are. The distance is calculated using the Euclidean distance and K is the number of neighbors we choose. Here we initiated K as 5 and tested around to get the best performance. KNN is easy to understand and implement. However, the reason why KNN does not perform well on our dataset is that it requires homogeneous features.

- Multilayer Perceptron
  A multilayer perceptron (MLP) is a deep, artificial neural network. It is composed of more than one perceptron. They are composed of an input layer to receive the signal, an output layer that makes a decision or prediction about the input, and in between those two, an arbitrary number of hidden layers that are the true computational engine of the MLP(Nicholson,). MLP works better with data that has a spatial relationship especially images which is the reason why it performs the best.

The accuracy for these four models are shown below:

Model	Training Accuracy	Validation Accuracy
Logistic Regression	0.56	0.53
MLP Classifier	0.56	0.54
Random Forest	0.97	0.45
K-nearest Neighbors Classifier	0.65	0.53

Table 2: Accuracy of baseline models

4.3 Final Model

Feature Processing

We reflected on baseline model and concluded that our features were not sufficient enough to represent an image's diverse color combination. Primary color data cannot represent the color style of an entire image. So we

improved our color feature extraction to a 7*10 dimensional color histogram, which we used package Colorgram to implement.
On top of RGB, our final model added HSL (Hue, Saturation, Lightness), which are alternative representations of the RGB color model. Hue is a degree on the color wheel from 0 to 360, where 0 is red, 120 is green, 240 is blue. Saturation is a percentage value; 0% means a shade of gray and 100% is the full color. Lightness is also a percentage; 0% is black, 100% is white.
For each color used in a given image, we extract 7 features of them:

- RGB (The color represented as a namedtuple of RGB from 0 to 255)
  E.g. (R=255, G=151, B=210).
- HSL (The color represented as a namedtuple of HSL from 0 to 255)
  E.g. (H=230, S=255, L=203).
- Proportion (The proportion of the image that is in the extracted color from 0 to 1)
  E.g. 0.34.

And then we rank the colors based on proportion and select the top 10 colors as our features for one image. This improvement is also based on our previous assumption that each creator has a frequently-used color palettes. The top 10 colors we select here are equivalent to the 10 frequent colors that the creator uses.

Model Selection

To explore the best relationship between colors and image style, other than traditional models in sklearn library, there is another option: Jaccard similarity. It can be a better option for some situations. For one case, when building color feature vector for logistic regression, if we order the vector in color proportion order, the model can miss the similarity among colors. For example, if we have 2 same arts drawn in same color set but one is black background and the other is white background. Even if they have the same color set, due to the different color in the background, same color will have different order in feature vector. Let's say in picture 1, color [233,122,135] is in position 3 in feature vector while it is in position 5 in picture 2. Linear regression model will not spot this similarity since every single coefficient is calculated based on the values on the same vector position. To eliminate this drawback, we explore the Jaccard Similarity where the similarity is calculated regardless of the position of the feature value.
We first implemented Jaccard Similarity to our validation dataset, we surprisingly found out that many Jaccard comparison ended up with result 0, which means no same color is spotted even among the arts created by the same creator. This disproved our assumption that creators use preset color plate to create artworks. After looking into RGB values of similar artworks, it was found out that many art creators pick color manually, resulting in similar color use but not the exact same color in previous artworks.
To overcome this problem in Jaccard Model, we can improve Jaccard Similarity by taking advantage of "similar color". Jaccard similarity is calculated as:

$$J(A,B) = \frac{|\ A \cap B\ |}{|\ A \cup B\ |}$$

Where intersection is counted only when the elements from 2 sets are identical. Here we create a new Jaccard Similarity called Jaccard Close Similarity. Jaccard Close Similarity differs from Jaccard Similarity by defining that intersection is counted when elements from 2 sets have difference below certain threshold instead of being identical. In our model, we define the difference between colors as:

$$|\ r1 - r2\ | + |\ g1 - g2\ | + |\ b1 - b2\ |$$

And threshold is set as 65 as optimal value in validation dataset.
We used Multilayer Perceptron Classifier with setting of 2 layers and 5 perceptrons per layer. This setting is used due to limitation of our machine and we believe that more complex perceptron layer set-up has the potential to improve our model performance.

4.4 Performance and Results

When we test this model on artworks created by creator '79', we managed to predict correctly on his/her works that were painted with a similar color palette, shown as following:

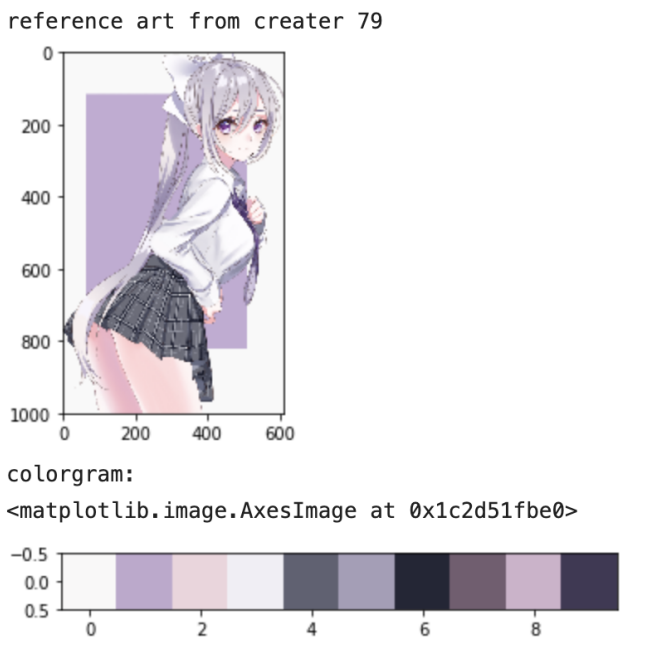

Figure 3: reference image of creator 79 from training set

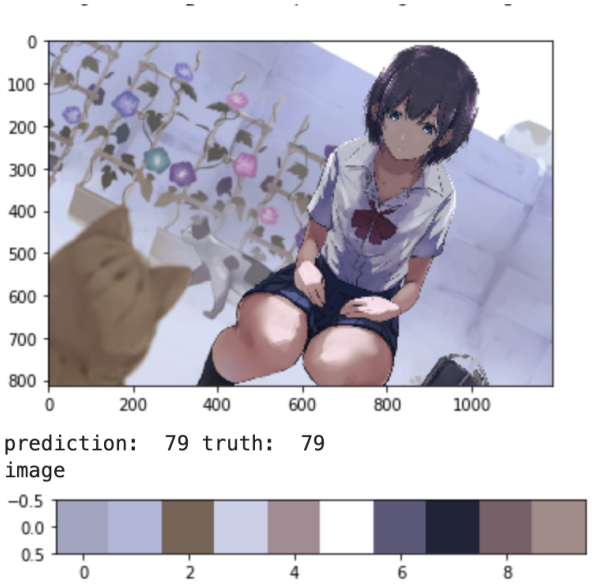

Figure 4: Correct predictions from validation set

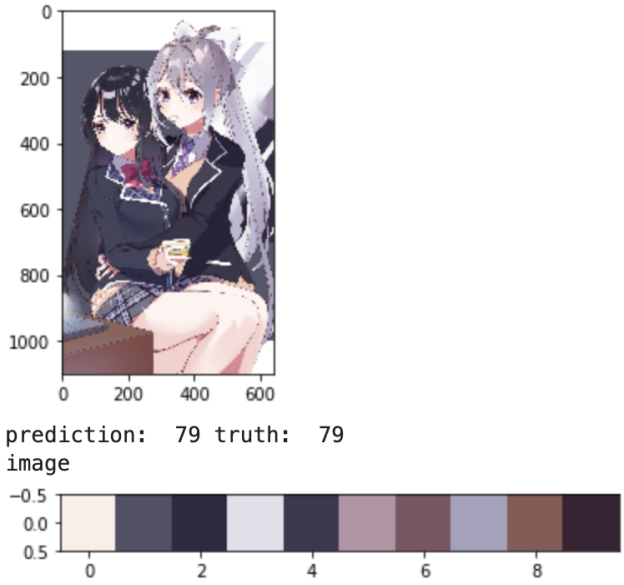

Figure 5: Correct predictions from validation set

However, our validation accuracy fluctuate heavily due to lack of training set for each creator. This limitation is resulted from restricted computer memory. After several tests, the average validation accuracy is about 68%.

5 Conclusions

Based on all the work we've done, our model is proved to be efficient in creator prediction. In task 1, tags were encoded into numerical feature vectors through a blended encoding method. The logistic regression model with image tags as input was built and optimized to 90% accuracy. In task 2, we extracted color features in RGB and HSL, optimized the model by combining Jaccard Similarity and MLP classifier which increased the accuracy to 68 percent.
However, limitations exist for both two tasks and models. Here is a short illustration about how we can improve our models in future research.

Task 1

(a) Since correlation information between tags is not preserved, different secondary tags may produce the same feature vector. Here is an example:
For two samples (a, b) with the same primary tags, the feature vector produced after step 1:

1, 0, 1, 0, 1, 1

Corresponding categories (primary tags):

sunny, day, skirt, white, black, hair

Secondary tags for sample (a):

black hair, white skirt, black shirt

Feature vector produced by step2:

1, 0, 2, 1, 3, 2

Secondary tags for sample (b):

black hair, black skirt, white shirt

Feature vector produced by step2:

1, 0, 2, 1, 3, 2

Although secondary tags are different, the final feature vectors are the same, which will cause inaccuracy. However, comparing to one-hot-encoding all tags, the general trade-off between information loss and space efficiency is worthy here.
(b) Another problem of our model is that if there are few input tags, and those tags are common for all images (e.g. "black", "white"), the model tends to provide an average result. One possible solution is to assign a weight to the feature vector, which is not applied in our current model, but can be a possible optimization method.

Task 2

Due to the limitations of computer memory size and CPU, we are only able to process 1500 images, which is a very small fraction of our entire dataset. What's more, the dataset is unbalanced in which creator '79' and creator '168' uploads extremely high amounts of images. We had to pre-process the dataset by filtering the creators who have more than 100 images. Then we have 100 images from each of the 15 creators which is 1500 images in total. However, in order for our recommendation system to perform well, it is better to have a larger dataset.

References

Padraig Cunningham and Sarah Delany. 2007. k-nearest neighbour classifiers. *Mult Classif Syst*, 04.

Jeff Hale. 2018. Smarter ways to encode categorical data for machine learning.

Parida L. He, D. 2016. Does encoding matter? a novel view on the quantitative genetic trait prediction problem. *BMC Bioinformatics*, 17:272.

Peter Hurford. 2017. Image feature engineering. *Kaggle*.

Stefan Jansen. Hands-on machine learning for algorithmic trading.

Chris Nicholson. A beginner's guide to multilayer perceptrons (mlp).

RawPedia. 2014. Rgb and lab.

Helen Han Aseem Agarwala Trevor Darrell Aaron Hertzmann Holger Winnemoeller Sergey Karayev, Matthew Trentacoste. Recognizing image style. *KARAYEV ET AL.: RECOGNIZING IMAGE STYLE*.

Danny Varghese. 2018. Comparative study on classic machine learning algorithms.