Artificial Intelligence Capstone Project1

- Art Style Classification for Otakus

109550135 范恩宇

1 Introduction

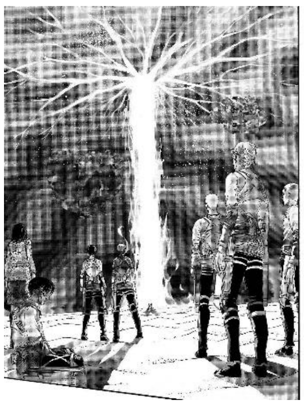
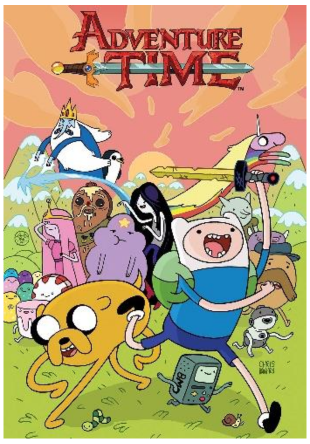
For most otakus, things that relate to ACGN are their favorite, whether they're animations, comic books, or games, otakus' life and social circle are often centered by them. However, some trendies and moralists enjoy making fun of otakus and the ACGN culture they love. Fortunately, in recent years, culture of ACGN has not only been widely spread, but also become more popular to the public. We can see children and news media talk about the plot of Attack on Titan, Jujutsu Kaisen, and so on all the time. Despite the fact that ACGN culture becomes a lot trendier these days, some people still have a hard time realizing the difference between comic(western ones like DC & Marvel), manga(usually Japanese, like DRAGON BALL), cartoon(western animations that often contains dramatic body movements), anime(Japanese animations, often adapted from manga).

Since I consider myself an otaku, I hope that more and more people know how to differentiate these things. Recalling those I've heard, the method of differentiating ACGN things that most otakus teach others, are often ambiguous and too difficult for those knowing not that much about ACGN culture. In my opinion, the art style of comic, manga, cartoon, and anime are quite different, so it should be possible for us to create a practical AI classifier to recognize the difference between those medias and find out what type of artwork that an image we input is. Thus, the task becomes classifying several images into 4 categories, which are comic, manga, cartoon, and anime, and trying to get better accuracy.

To complete the task, I use 2 supervised algorithms and 1 unsupervised method. For supervised learning, I choose SVM algorithm and Decision Tree for classification and regression modeling. As for the unsupervised method, I choose K-Mean algorithm as most people suggest, for clustering training images. Since I believe that the quality of dataset affects the result the most, experiments I did basically are related to it, like smaller/bigger dataset and data augmentation, and the result yields an interesting relation between data manipulation and training accuracy, which will be further discussed in following sections.

2 Dataset(The file size is big , so they are put in the cloud link: https://drive.google.com/drive/folders/1yI-dPgU7J6wcCxao6Znyt2tEqQo0RVKX?usp=sharing)

There are 3163 images in my dataset, with 4 categories : comic, manga, cartoon, and anime, and the corresponding amount are 1318,236,385,and 1224 respectively. For supervised learning ,images of different categories are stored in folders with the name of the category. While for unsupervised learning, they are put into a same folder.

comic	manga
	
cartoon	anime
	

a. Category Confirmation

For all images I used, most of them came from a famous website with many kinds of images, Pinterest. As I said before, I am an otaku that have enjoyed many kinds of comics, mangas, cartoon, and animes, so I know what category an artwork belongs to and thus be able to search for those I want directly by keywords. Just in case, I still check Wikipedia pages that talks about these things to make sure my knowledge is correct.

b. Image Capture

As the previous part said, those images are mostly from Pinterest, and the others are my personal collections. However, there are still some problems existing in the search result, for example, I search for Attack on Titan(anime), but I get images of titans from Dark Souls(video game). These kinds of problems spend me quite a while to filter some ridiculous or ambiguous data.

c. Image Resize

For image classification, I don't think it's a good idea to make the images' size unchanged, this'll result in too many kinds of sizes. Therefore, I resize all images into 256*256 in the end, since I've heard that it's a better size for model training. In addition, set all images to square may cause less distortion between rectangle images and lied-down rectangle images.

3 Methods

For this project I use actually 4 different approaches to for the ACG artwork classification task. Three are supervised algorithm, which are support vector machine (SVM), decision tree, and random forest. Another one is an unsupervised method, which is the K-means algorithm. The following are the implementation details.

3.1 Supervised Learning

A. Support Vector Machine(SVM)

The fundamental principle behind Support Vector Machine (SVM) regression is to identify a better kernel function for mapping data onto a hyperplane, enabling effective separation of data points.

The data is split into training and testing sets using the train_test_split function here, with 20% of the data used for testing and 80% for training. An SVM classifier with a linear kernel is instantiated and trained using the training data.

Also do SVM classification with different numbers of cross-validation sets specified in cv_sets list. For each number of folds, it creates an SVM classifier with a linear kernel, performs cross-validation using cross_val_score function, and prints the mean cross-validation score.

B. Decision Tree

The core concept of Decision Tree classification is to create a tree-like model that makes predictions by recursively partitioning the input space into disjoint regions.

The data is split in the same way with SVM for getting training and testing sets. Of course, I also do cross-validation with different numbers of folds specified in the cv_sets list, just like I what I do for SVM."

C. Random Forest

Of course I've tried random forest, which combines multiple decision tree models for prediction. It builds multiple decision trees by randomly sampling training data and selecting features. Each decision tree independently predicts the outcome, and the final result is made by aggregating predictions of all trees through voting or averaging. But the precision of one categories I give seems to be overfitting, so I didn't choose this eventually.

3.2 Unsupervised learning

A. K-Means

It's used for clustering data points into groups or clusters, it basically partitions the input into K distinct clusters, where each cluster is represented by its centroid. It iteratively assigns each data point to the nearest centroid and updates the centroids based on the mean of the data points assigned to each cluster. This process continues until convergence, where centroids no longer change their significantly or maximum number of iterations is reached. It's efficiency and scalability make me think that it can be a good choice for classifying ACGN artworks.

4 Experiments

A. Performance Comparison

First, I compare the performance of 3 supervised algorithms and 1 unsupervised algorithm. The below tables are the results.

Despite the fact that Random Forest gives me the best number, it's classification accuracy of cartoon being 100% seems really strange to me. Not only because of the 100% score, also because cartoon isn't predicted well in all other methods, comparing with manga & anime.

In most results, the predictions of manga & anime are obviously better, and I believe that it's because images of manga & anime I give are more than two other categories.

Supervised ones

Method	SVM	Decision Tree	Random Forest
Overall Accuracy	62.8%	63.9%	80.1%
Accuracy(comic)	47%	29%	72%
Accuracy(manga)	85%	84%	88%
Accuracy(cartoon)	42%	26%	100%
Accuracy(anime)	56%	66%	73%

Unsupervised ones(K-Means)

	comic	manga	cartoon	Anime
Accuracy	18.37%	81.94%	8.24%	63.77%

B. Decrease Training Data

Through the process of performance comparison, I infer that amount of dataset might affects the result. So I tried using a dataset that manga & anime images only have 1/3 left, then the results are in the below tables.

Supervised ones

Method	SVM	Decision Tree	Random Forest
Overall Accuracy	53.9%	58.4%	-
Accuracy(comic)	45%	41%	-
Accuracy(manga)	71%	80%	-
Accuracy(cartoon)	22%	24%	-
Accuracy(anime)	59%	63%	-

Unsupervised ones(K-Means)

	comic	manga	cartoon	Anime
Accuracy	18.37%	62.71%	8.24%	54.39%

C. Effect of Data Augmentation

Since the dataset I get for comic & cartoon are much less than the two other categories, so I basically do data augmentation on comic & cartoon images, generating 6 times more images for them, and the new generated images are different duplicates of the original ones. The below tables are the results.

Supervised ones

Method	SVM	Decision Tree	Random Forest
Overall Accuracy	42.6%	63.9%	-
Accuracy(comic)	49%	29%	-
Accuracy(manga)	72%	84%	-
Accuracy(cartoon)	31%	26%	-
Accuracy(anime)	32%	66%	-

Unsupervised ones(K-Means)

	comic	manga	cartoon	Anime
Accuracy	30.47%	57.41%	24.48%	48.03%

5 Discussion

a. Performance between different categories

For the original experiment, manga & anime are predicted better in all approaches, except for Random Forest. Through the experiment of decreasing data amount, we can see that the accuracy of both manga & anime are decreased significantly. So here I inferred that amount of training data does affect the final result. It's sad that I didn't find enough qualified images of comic & cartoon.

b. Data Augmentation

For those two categories having less images, I do data augmentation on them. But only comic images with SVM perform better, which is out of my expectation. After some study, I think it's because that,

1. Decision trees are robust to small variations in the training data. They partition the feature space based on thresholds and are less sensitive to minor changes in data points. This makes data augmentation with small variations may not significantly impact the decision boundaries learned by decision trees.
2. K-means is a centroid-based clustering algorithm that assigns data points to the nearest centroid, and data augmentation techniques may distort the underlying structure of the data, leading to misalignment between the augmented data points and their true clusters. This situation make K-mean algorithm's struggle to accurately cluster the augmented data, especially if the augmentation introduces significant noise or biases.

c. Data Overfitting

When I tried using Random Forest, the result of cartoon seems to be overfitting. I think it may result from my insufficient regularization or improper tuning of hyperparameters. The imbalanced data may be also the cause, since the overfitting one is cartoon, which has the least amount of data.

d. Cluster Matching in K-Means

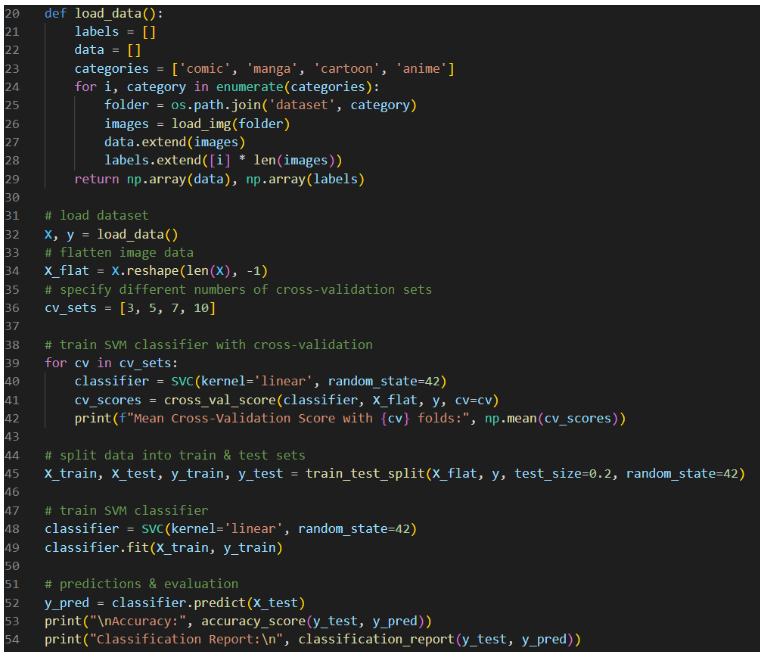
When using K-Means, I found out that the mode & second most result of the prediction of comic and anime are the same. I eventually check how they match by the resulting accuracy score and result from supervised methods. Since comics and animes are both colored, characters in them are also both drawn in ordinary ways (unlike cartoons, which are often exaggerated) .

e. **How to improve the result**

Since data augmentation works in some methods, I believe that it'll be possible to reach better results through this in other training methods, if the image duplication is done more exquisitely. In addition, the dataset should be bigger and more balanced, and this leads to a must for finding better methods of crawling data.

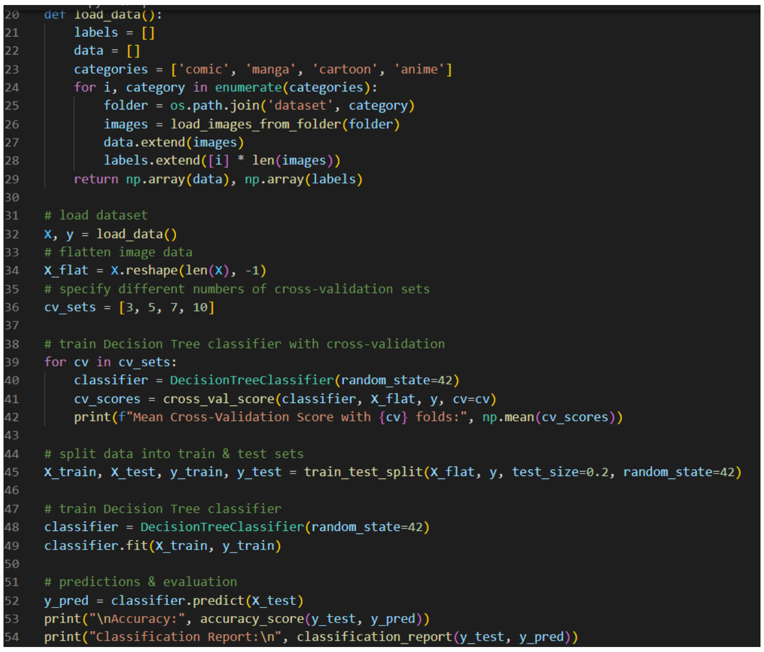
6 Appendix

A. svm.py



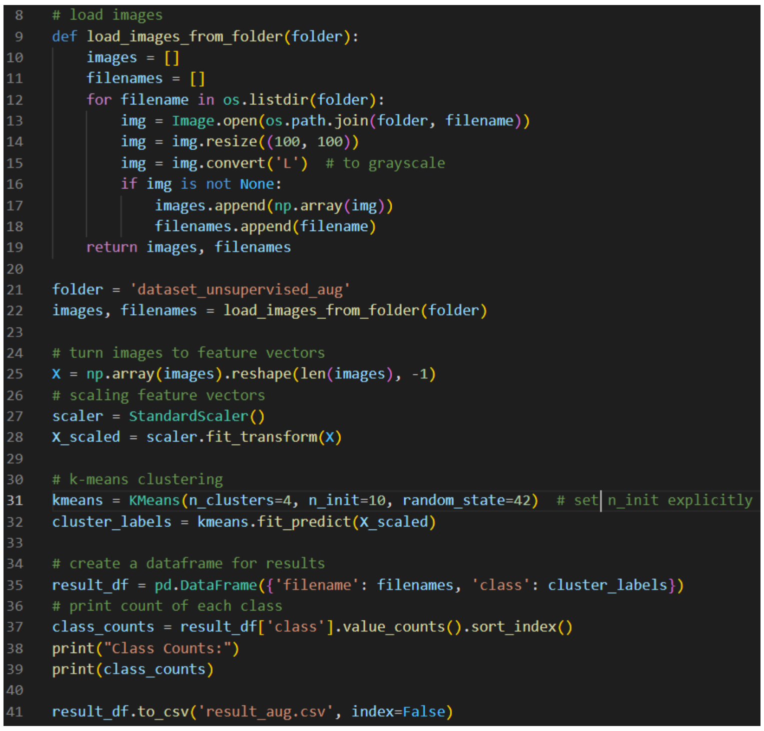
```python
def load_data():
    labels = []
    data = []
    categories = ['comic', 'manga', 'cartoon', 'anime']
    for i, category in enumerate(categories):
        folder = os.path.join('dataset', category)
        images = load_img(folder)
        data.extend(images)
        labels.extend([i] * len(images))
    return np.array(data), np.array(labels)

# load dataset
X, y = load_data()
# flatten image data
X_flat = X.reshape(len(X), -1)
# specify different numbers of cross-validation sets
cv_sets = [3, 5, 7, 10]

# train SVM classifier with cross-validation
for cv in cv_sets:
    classifier = SVC(kernel='linear', random_state=42)
    cv_scores = cross_val_score(classifier, X_flat, y, cv=cv)
    print(f"Mean Cross-Validation Score with {cv} folds:", np.mean(cv_scores))

# split data into train & test sets
X_train, X_test, y_train, y_test = train_test_split(X_flat, y, test_size=0.2, random_state=42)

# train SVM classifier
classifier = SVC(kernel='linear', random_state=42)
classifier.fit(X_train, y_train)

# predictions & evaluation
y_pred = classifier.predict(X_test)
print("\nAccuracy:", accuracy_score(y_test, y_pred))
print("Classification Report:\n", classification_report(y_test, y_pred))
```

B. decision tree.py



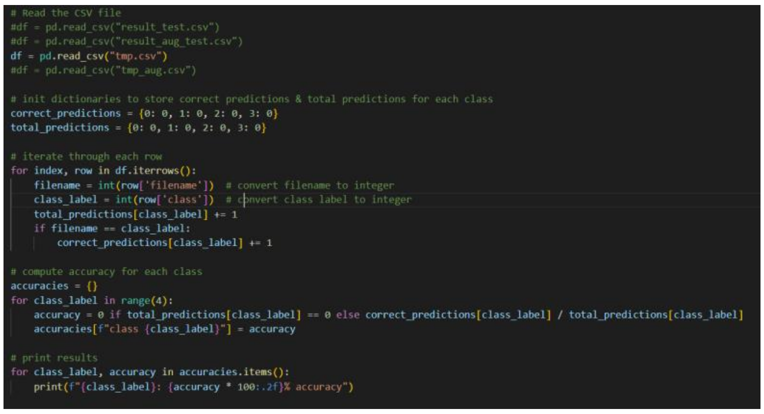
```python
def load_data():
    labels = []
    data = []
    categories = ['comic', 'manga', 'cartoon', 'anime']
    for i, category in enumerate(categories):
        folder = os.path.join('dataset', category)
        images = load_images_from_folder(folder)
        data.extend(images)
        labels.extend([i] * len(images))
    return np.array(data), np.array(labels)

# load dataset
X, y = load_data()
# flatten image data
X_flat = X.reshape(len(X), -1)
# specify different numbers of cross-validation sets
cv_sets = [3, 5, 7, 10]

# train Decision Tree classifier with cross-validation
for cv in cv_sets:
    classifier = DecisionTreeClassifier(random_state=42)
    cv_scores = cross_val_score(classifier, X_flat, y, cv=cv)
    print(f"Mean Cross-Validation Score with {cv} folds:", np.mean(cv_scores))

# split data into train & test sets
X_train, X_test, y_train, y_test = train_test_split(X_flat, y, test_size=0.2, random_state=42)

# train Decision Tree classifier
classifier = DecisionTreeClassifier(random_state=42)
classifier.fit(X_train, y_train)

# predictions & evaluation
y_pred = classifier.predict(X_test)
print("\nAccuracy:", accuracy_score(y_test, y_pred))
print("Classification Report:\n", classification_report(y_test, y_pred))
```

C. k-means.py



```python
# load images
def load_images_from_folder(folder):
    images = []
    filenames = []
    for filename in os.listdir(folder):
        img = Image.open(os.path.join(folder, filename))
        img = img.resize((100, 100))
        img = img.convert('L')  # to grayscale
        if img is not None:
            images.append(np.array(img))
            filenames.append(filename)
    return images, filenames

folder = 'dataset_unsupervised_aug'
images, filenames = load_images_from_folder(folder)

# turn images to feature vectors
X = np.array(images).reshape(len(images), -1)
# scaling feature vectors
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

# k-means clustering
kmeans = KMeans(n_clusters=4, n_init=10, random_state=42)  # set n_init explicitly
cluster_labels = kmeans.fit_predict(X_scaled)

# create a dataframe for results
result_df = pd.DataFrame({'filename': filenames, 'class': cluster_labels})
# print count of each class
class_counts = result_df['class'].value_counts().sort_index()
print("Class Counts:")
print(class_counts)

result_df.to_csv('result_aug.csv', index=False)
```

D. k-means_accuracy_simple.py



```python
# Read the CSV file
#df = pd.read_csv("result_test.csv")
#df = pd.read_csv("result_aug_test.csv")
df = pd.read_csv("tmp.csv")
#df = pd.read_csv("tmp_aug.csv")

# init dictionaries to store correct predictions & total predictions for each class
correct_predictions = {0: 0, 1: 0, 2: 0, 3: 0}
total_predictions = {0: 0, 1: 0, 2: 0, 3: 0}

# iterate through each row
for index, row in df.iterrows():
    filename = int(row['filename'])  # convert filename to integer
    class_label = int(row['class'])  # convert class label to integer
    total_predictions[class_label] += 1
    if filename == class_label:
        correct_predictions[class_label] += 1

# compute accuracy for each class
accuracies = {}
for class_label in range(4):
    accuracy = 0 if total_predictions[class_label] == 0 else correct_predictions[class_label] / total_predictions[class_label]
    accuracies[f"class {class_label}"] = accuracy

# print results
for class_label, accuracy in accuracies.items():
    print(f"{class_label}: {accuracy * 100:.2f}% accuracy")
```