Smiling or not

Vasilescu Andreea[a]

[a] *University of Bucharest, Academiei Street, no. 14, Bucharest, 077131, Romania*

Abstract

This paper is about applying Unsupervised Learning on a Kaggle data set that contains pictures with smiling and not smiling people that aims to cluster them.

This is my first unsupervised learning task I am working on, and clustering it's considered to be a good task to start with.

To conduct this study I have searched for more possible data sets available on Kaggle, and I picked this one as it fits the best the recommendations for my project.

Keywords: Clustering, Unsupervised Learning, AI, Computer Vision

1. Introduction

Unsupervised learning is a machine learning approach that deals with finding patterns and structures in data without using labeled examples. It includes tasks like clustering (grouping similar data points) and dimensionality reduction (simplifying data while retaining important information). It's used for tasks such as customer segmentation, anomaly detection, and reducing data complexity.

1.1. Related Studies

Clustering is a fundamental concept in unsupervised machine learning. It has been widely studied and applied across various fields. Some of the studies conducted are as follows:

- Data mining, pattern recognition, infromational retrieval.
- Bioinformatics gene expression analysis, protein structure prediction or DNA clasification
- Clustering aids in text recognition or topic modelling
- Anomaly detection

2. Preliminaries

The following section will provide an overview of the current advancements and achievements in the Machine Learning field, as well as state of the art models, methodologies, limitations and potential applications. The purpose of this section aims to set a foundation for the conducted study that will be presented.

2.1. Random Forests

Random Forest solves problems of regression and classification using ensembling methods and performing regression these things making it similar to the Random Forest model. 1

The difference between Random Forest and XGBoost lies both in the way the trees are constructed and the manner in which the results will be combined.

Random Forest algorithm need to have set 3 parameters before starting the training:

1. node size
2. number of featured samples
3. number of nodes

Random Forest model uses an algorithm that leverages many decision trees and combines them in order to return a single result. An instance of randomness is applied in the algorithm through a method called "Feature bagging", used to reduce the correlation between trees, but also for diversity.

The prediction varies based on the problem.

For **regression** it's performed the average of individual trees.

In case of a **classification** task it's used something like "majority vote". For example, the predicting class can be based on which categorical variable is the most common.

At the end, the algorithm uses for testing a sample called **Out-of-bag** (OOB), which represents a third of training samples. OOB is used in finalizing the prediction, as it's used for cross validation.

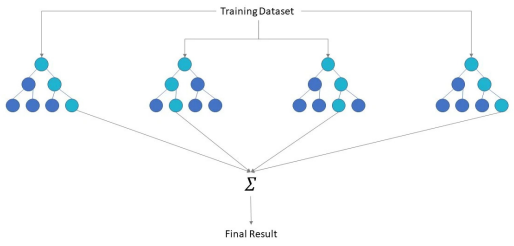


Figure 1: Random Forest Model

One of the **advantages** of Random Forests if its efficiency on data-set with a higher number of variables. Among the benefits,

I can enumerate also the reduced risk of over-fitting and the ability to solve both regression and classification problems.

The biggest **disadvantage** of Random Forest are complexity, as this model create and combines lots of decision trees (by using sklearn library, the number of decision trees created will be 100).

2.2. Isolation Forests

Isolation Forests is an anomaly detection algorithm that focuses on ifentifying the anomalies ina data set. It works by isolating the anomalies into individual trees on the prodcess o building a random forest model.

The isolation forests work as follows:

- **Random Selection of Features and Splitting Points:** At each step of building an isolation tree, a random feature is selected, and a random value within the range of that feature's values is chosen as a splitting point. This randomization helps isolate anomalies faster.
- **Recursive Partitioning:** The dataset is partitioned into smaller subsets by recursively creating branches of the tree. Anomalies, being fewer in number, are more likely to end up in smaller partitions with fewer splits.
- **Depth of the Tree:** The algorithm continues building the tree until a predefined depth is reached or all data points are isolated into their own leaf nodes.
- **Scoring Anomalies:** To identify anomalies, the path length from the root to each data point's leaf node is measured. Anomalies are expected to have shorter paths because they require fewer splits to be isolated.
- **Anomaly Score:** The anomaly score for each data point is computed based on the average path length. Lower average path lengths indicate higher anomaly scores, suggesting that the points are more likely to be anomalies.

Isolation Forests have advantages for detecting anomalies in high-dimensional data or when the distribution of anomalies is not well-defined. They are relatively efficient, especially for large datasets, and can be particularly useful for preprocessing tasks in data quality assessment or identifying outliers before further analysis.

2.3. Balanced Iterative Reducing and Clustering using Hierarchies - BIRCH

BIRCH is a clustering algorithm for large dataset. It works by first generating a small and compact summary of the largest dataset, and retain as much information as possible.

There are four phases followed by BIRCH, but the second and the fourth are optional:

1. Scan data in the memory (similar to loading data in a model, followed by scanning the entire data and fir it into clustering feature trees (CF trees),
2. (optional) Condense the data (resize)
3. Global clustering
4. (optional) Refine clustering

threshold is the maximum number of data points a subcluster in the leaf node of a CF can hold **branching factor** represents the maximum number of CF tree in each cluster **n_cluster** is the number of clusters to be returned after entire BIRCH algorithm

The **advantages** of BIRCH algorithms are as follows:

- easy implementation
- works fast on large datasets as it only require one time scan of the dataset
- it uses an incremental method that doesn't require the entire dataset in advance

2.4. Agglomerative Clustering

Agglomerative clustering is a hierarchical clustering technique used in unsupervised machine learning. It starts by considering each data point as its own cluster and progressively merges similar clusters to form a hierarchical tree-like structure known as a dendrogram.

The process begins with the two closest data points being grouped into a single cluster. Then, at each step, the two nearest clusters (or data points) are merged into a larger cluster until all data points are in a single cluster. This creates a hierarchy of clusters, where the leaves of the dendrogram are individual data points, and the root represents the entire dataset.

Agglomerative clustering offers flexibility in determining the number of clusters based on where the dendrogram is cut. Cutting the dendrogram at different levels yields varying numbers of clusters. It's often used to reveal relationships and structures within data that may not be apparent at first glance.

Key aspects of agglomerative clustering:

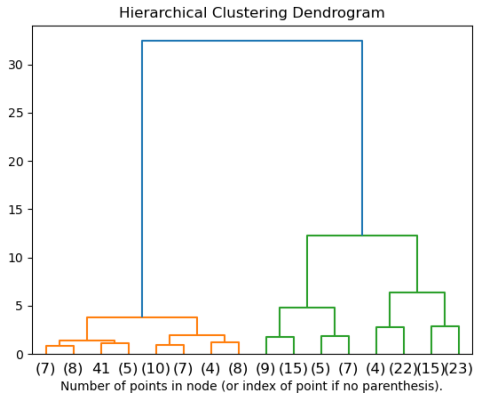
Distance Metric: The choice of distance metric (e.g., Euclidean, Manhattan) influences how similarity is measured between data points or clusters.

Linkage Criteria: Different methods, like single linkage, complete linkage, or average linkage, determine how the distance between clusters is calculated during merging.

Dendrogram: The dendrogram visually represents the merging process and can guide the selection of clusters based on the desired granularity.

Agglomerative clustering is computationally more intensive for larger datasets but offers insights into hierarchical relationships within the data. It's suitable for tasks like exploratory data analysis, understanding data structures, and identifying natural groupings.

Pedregosa et al. (2011)



3. Experiment

In the following are going to explain my work done on clustering images of people that are smiling or not, as well as present how the experiment was conducted. This chapter contains information regarding the implementation of the coding part that stands behind this paper, what I have tried and how I get to the final version. At the end of this chapter, I am going to provide the pros and cons of each model we've used on our data-set, providing a brief explanation.

3.1. Motivation for the theme

For this experiment, I used **Python** programming language. I have used one of the main libraries in python such as:

- numpy for numerical operation, as it is considered to be one efficient on array operations, as well as mathematical functions.
- pandas for reading the data.

Google Collaborator was my choice because I was already familiar with it from previous university projects I worked on. One big advantage for us was the fact that I could easily store the data in Cloud, and then be able to access our work from any device.

3.2. Data

For this project I used **Python** programming language. I have used one of the main libraries in python such as:

- **numpy** - for numerical operation, as it is considered to be one efficent on array operations, as well as mathematical functions.
- **pandas** - for reading the data.
- **matplotlib** - for plotting the graphics
- **PIL** - for ploting the immages
- **Sklearn**, - for agglomerative clustering and pariwise_distance
- **Tensorflow** for preprocessing the input, loading the images and transforming the images into arrays

3.3. Reading the data

The initial data was as follows:

1. train
   (a) smile: 600 images
   (b) non_smile: 603 images
2. test
   (a) smile + non_smile = 350

I have defined the paths for the unlabled train data for smiling and non_smiling images.

Then I have defined a data frame for both categories. The data frame contains two columns. One with the actual image (each image had 128 pixels) , and the second column with the path.

After that, I have created a function for loading an image and preprocess it.

- Loaded the image from the path and used load_img function. As argument I have passed 224 pixels for each image.
- Converted the image into a NumPy array using img_to_array function to transform the image data into a numerical array, so it can be processed by the computer.
- used preprocess_input function to prepare the image for future model requirements.

Checked to see how one image looks like after preprocessing:

Figure 2: Image after preprocessing

3.4. Isolation Forest

For **Isolation Forest** algorithm I have used the Isolation Forest pre-difined mode.

I have created an instance of the isolationForest class that I called classifier. The contamination parameter was originally set to 0.1, meaning that 10 percent of the data was expected to be outlier. This was just to test things our.

Then, I have fitted the Isolation Model to the data set that I have, using the array reshaped 2D format, where each row represented an image, and the pixels were flattened. Using this, the model learned to distinguish the normal and anomaly pattern.

After that, I have calculated the anomaly score for each image in the data set, using the trained model.

The anomaly threshold was set to -0.5, meaning that images below this threshold will be categories as as normal(smile), while the ones above will be categorised as anomalies (non_smile).

Finally, I have created an array with anomaly scores, assigning each image to a category, based on properties described above.

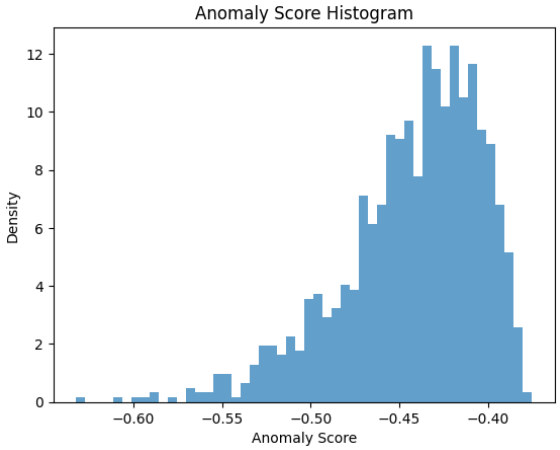


Figure 3: Isolation Forest Anomaly Score

To test more, I have created a parameter grid and opted for different parameters, plotting the images of anomaly score for each of them.

```
param_grid = {
    'contamination': [0.05, 0.1, 0.15, 0.2],
    'max_samples': ['auto', 100, 200, 500],
    'n_estimators': [50, 100, 200, 500]
}
```

3.5. BIRCH

For the BIRCH model I have created a different function for loading and preprocessing the images that takes the path to the folder of images and transform the images into a size of 224, 224 before transforming it into an array. Next, it flattens the image using np.ravel and append it to a list of images. I am preprocessing both categories of images and then use vstack to concat the images.

1. For the first try I have generated **600 samples** using make_blobs, choosing **2 centers** as I have two classes and a **standard deviation of 0.75**. Next, from sklearn.cluster I have imported the Brich model. To control the maximum number of subclusters for each node in the structure I have set the **branching_factor** parameter to **20**, **n_clusters = None**, as the algorithm will automatically determine the number of clusters based on the data and the other specific parameters I chose, and fit the model on dataset.
   I have train and predicted on the same dataset, so I can see how the model was capturing the underlying patterns:
2. The second apporach was to test diferent parameter values:

Figure 4: n_samples=600, centers=2, clusters_std=0.75, branching_factor=20, n_clusters=None, threshold=1.5

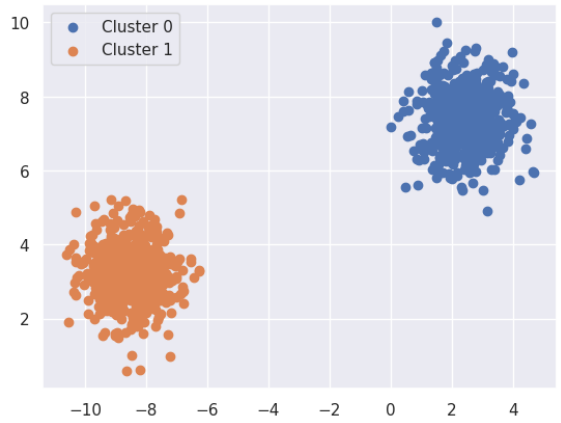


Figure 5: n_samples=600, centers=2, clusters_std=0.75, branching_factor=20, n_clusters=None, threshold=1.5

3.6. Agglomerative Clustering

To apply the agglomerative clustering I have combined the two categories of images and then calculate the features using np.vstack, a NumPy function that vertically stacks arrays, combining them into a single array along a new axis. After, I have calculated the pairwise distance (distance between all pairs of data points in my data set with the scope of producing a distance matrix).

Tested out the agglomerativeCluster provided by scikit-lean. As agglomerative is a hierarchical clustering technique, it starts with each data point as a single cluster and then it gradually merge the clusters together based on the distance or similarity metric.

In my experiment I set the number of clusters to two, as I have two categories, **smile** and **non_smile**.

I have created an instance of the AgglomerativeCluster class, using the parameters affinity as precomputed and linkage as avarage.

To fit the agglomerative clustering model to the data and then predict the cluster label for each point, I have used cluster_labels = agg_clustering.fit_predict(distances)

I have tried to use Agglomerative cluster, but got a warning

that indicated the input matrix provided to the linkage function for hierarchical clustering wasn't value. The issue was that pairwise distance function returned a symmetric matrix while the function expected a condensed one.

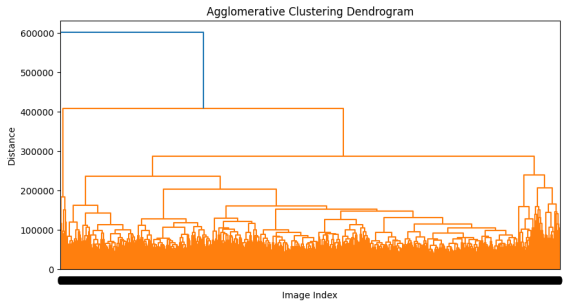


Figure 6: Agglomerative Clustering Dendrogram for symmetric matrix instead of condensed

Also tried a second approach by adding precomputed affinity and complete linkage. I have fitted the cluster labels on the distances. Still got the same warning but I could've seen a slight improvement on the dendogram:

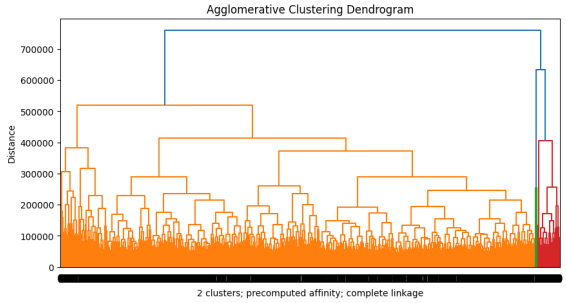


Figure 7: Agglomerative Clustering Dendrogram for symmetric matrix instead of condensed

I also had a third attempt:

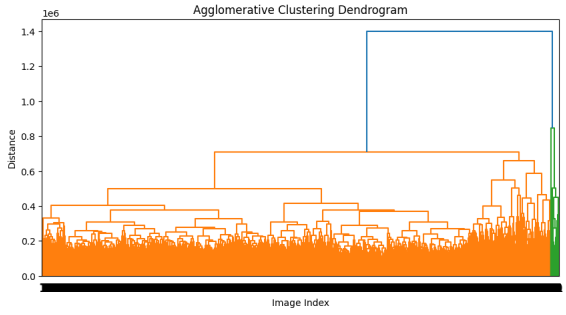


Figure 8: Agglomerative Clustering Dendrogram

4. Issued Encountered

– I have tried to use the same load and preprocessing function for Birch as for Isolation Forests, but below error was raised: ValueError: Found array with dim 4. Birch expected ¡= 2. still not working]. To fix it I created a new preprocessing function to flatten the images.

5. Future Work

As future work I would like to test other methods of unsupervised learning on the same data set, so I can become familiar with them, and be able to know which one to choose when facing a different task.

References

Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., Vanderplas, J., Passos, A., Cournapeau, D., Brucher, M., Perrot, M., Duchesnay, E., 2011. Scikit-learn: Machine learning in Python. Journal of Machine Learning Research 12, 2825–2830.