

# UNSUPERVISED ML, FINAL PROJECT

JACKSON WALTERS

## 1. INTRODUCTION

This project is for the Exploratory Data Analysis for Machine Learning certification from IBM via Coursera. The long-term goal is to create a t-SNE plot to observe clustering of mental health disorders based on symptom data, as in the linked paper<sup>1</sup>. I would like to include additional data based on life factors such as income, housing status, and insurance status. The idea is that if someone has no income, is unhoused, and uninsured then they would be much more likely to be classified as having a mental illness. I want to go beyond correlation, and see clustering.

## 2. PCA AND T-SNE FOR MNIST SAMPLE DATA

For now, I implemented code from here <sup>2</sup>, and removed the SAS connection. I rewrote it in pure Python with sklearn. I did both PCA and t-SNE on the MNIST data, and plots are shown below.

All code is publicly available on GitHub <sup>3</sup>.

## 3. MENTAL HEALTH CLIENT-LEVEL DATA

The data are 6.5mil rows and around 50 columns. There are 13 disorder codes listed, each a binary variable. The rest are life factors including employment, demographics like age, gender, ethnicity, geographic information, and more. I have dropped the irrelevant columns, one-hot encoded the categorical columns, and scaled all columns to be in  $[0, 1]$ .

There are a few different ways to think of the labels, but they are all derived from the 13 binary diagnosis columns. This is an example of a multi-label problem, distinct from a multi-output or multi-class problem, though we may transform it into one of these. For example, in one labeling I bin the disorders as [0 disorders, 1 disorder,  $\geq 1$  disorder] giving 3 labels. Another approach is to use a binary encoding so that each 13-tuple of binaries maps to a unique integer between 0-8191 since  $2^{13} = 8192$ . One could also count the number of disorders, yielding 14 labels, since the maximum sum is 13. Finally, one could use a

---

<sup>1</sup><https://www.ncbi.nlm.nih.gov/pmc/articles/PMC6783390/>

<sup>2</sup><https://medium.com/@violante.andre/an-introduction-to-t-sne-with-python-example-47e6ae7dc58f>

<sup>3</sup><https://github.com/jacksonwalters/ml-examples>

FIGURE 1. PCA plot of MNIST handwritten digits data

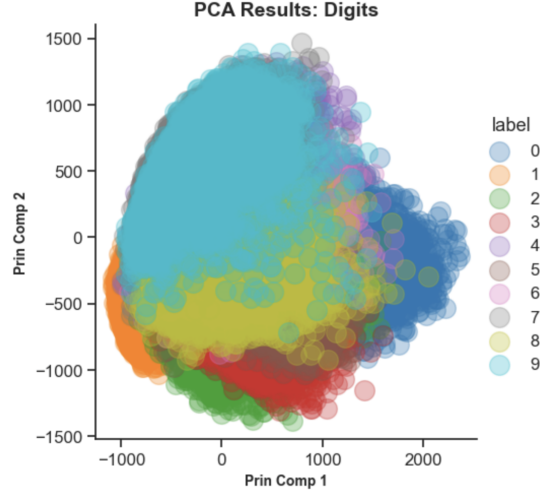
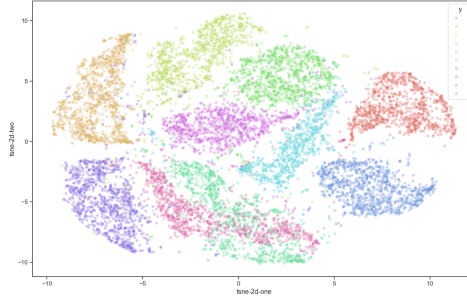


FIGURE 2. t-SNE plot of MNIST handwritten digits data



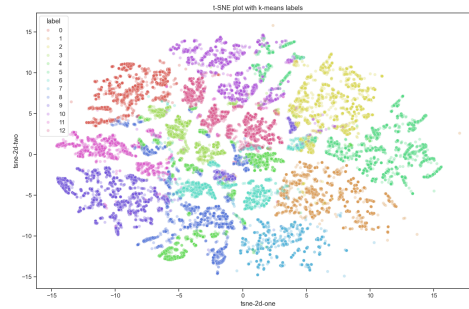
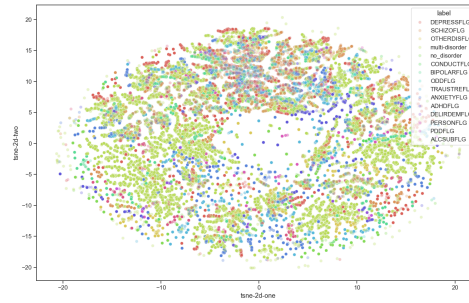
more sophisticated approach like k-means, which finds a centroid of clusters and expands outwards. This is what we will typically use to color the t-SNE and PCA plots:

Here we choose  $num\_clusters = 15$ , and we can clearly see some clusters forming, likely from the demographic information but the diagnostic information is included. If we try to label with *unique\_disorders*, that is 13 categories for those diagnosed with a unique disorder and two for no disorders and multi-disorder, we obtain the following plot:

#### 4. CLASSIFICATION

We can also attempt the classification problem. That is, given the demographic and life factor data, predict which disorder class the patient falls into. Since we have a number of labelings, we have a number of different problems to consider.

FIGURE 3. t-SNE plot of mental health data with k-means labeling

FIGURE 4. t-SNE plot of mental health data with *unique<sub>disorder</sub>* labeling

First we try the unique\_disorders labeling and train a RandomForest classifier. This is a multi-class problem.

accuracy: 0.34965  
precision: 0.30545804698445045  
recall: 0.34965

Next up we try multi-class LogisticRegression. On the unique\_disorders labeling, performance is similar to RandomForest. However, on the k-means labeling, we achieve much better performance:

accuracy: 0.88315  
precision: 0.8819518553140321  
recall: 0.88315

This reflects the much better clustering observed with the k-means labeling. Now the demographic and life factors are able to predict which k-means cluster a point will fall into, without diagnostic information.

We can try to drop 12/13 disorder columns and just do a logistic regression on the remaining column. This gives 13 separate logistic regressions, one for each disorder. We can then use it as a mutli-label classification, and compute the Hamming loss (a metric for distance between 0/1 datasets). In this case, we obtain `hamming_loss=0.12`.

Using `MultiOutputClassifier` or `OneVsRestClassifier`, we obtain `hamming_loss=.1` in both cases. Using k-nearest-neighbors MLkNN from `skmultilearn.adapt`, we obtain `hamming_loss=.11`.

Finally, using `MultiOutputClassifier` with `XGBoost`, we obtain `hamming_loss=.1`. The accuracy score is modified a bit to "subset", and reads:

```
accuracy: 17.2%
precision: 46.8%
recall: 19.8%
```

It's worth noting that just computing the number of multi-labels that are correctly predicted with `XGBoost` is 3%, whereas a random guess would be  $1/8192 = .01$  per data point, so 12.2%.

## 5. UNSUPERVISED LEARNING

I decided to look at the `disorder_cols` of the data, disregarding all other columns. This is a Boolean matrix with a lot of structure to uncover. I used a number of clustering algorithms to try to understand which combinations of disorders were most common. I used Hierarchical Agglomerative Clustering, DBSCAN, and k-means. I used non-negative matrix factorization, PCA, and t-SNE for dimensionality reduction.

## 6. DEEP LEARNING AND REINFORCEMENT LEARNING

I implemented a three layer deep neural network to perform classification on the 14 diagnostic boolean labels. The output of the classifier is 14 numbers which are then rounded to 0 or 1.

FIGURE 5. clustering results using DBSCAN

```

from sklearn.cluster import AgglomerativeClustering
agg = AgglomerativeClustering(n_clusters=10, metric='euclidean', linkage='ward')
agg.fit(data[disorder_cols])
agg_label_df = pd.DataFrame(agg.labels_, columns=['label'])

[84]: NUM_CLUSTERS_AGG = 10

[19]: #use DBSCAN
from sklearn.cluster import DBSCAN
db = DBSCAN(eps=.1, min_samples=2, metric='euclidean')
db = db.fit(data[disorder_cols])
db_label_df = pd.DataFrame(db.labels_, columns=['label'])

[20]: NUM_CLUSTERS_DB = len(set(db_label_df['label']))

[67]: codebook_descriptions['OTHERDISFLG']

[67]: 'Other mental disorder reported'

[37]: db_label_df.value_counts()[0:10]

[37]: label
0      9194
1      7142
23     3856
16     3702
18     3417
9      3351
20     2728
2      2264
34     1074
19      872
dtype: int64

```

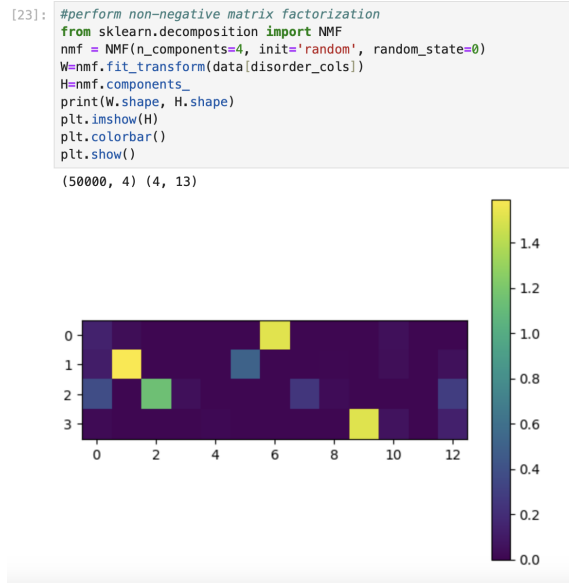
FIGURE 6. matrix factorization plot of  $H$ 

FIGURE 7. Architecture of the deep learning model with three dense layers

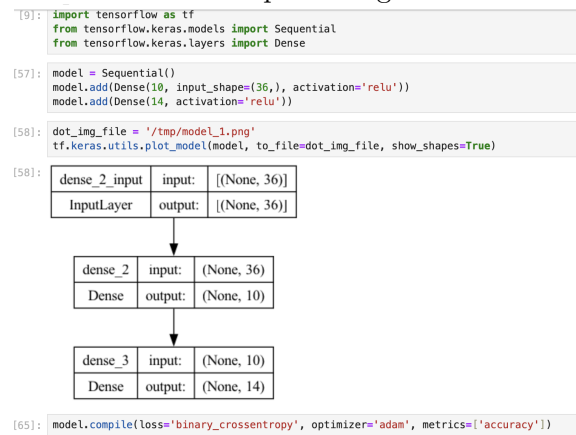


FIGURE 8. Predictions of the model had 7% accuracy

```
[66]: #fit the keras model on the dataset
#accuracy is likely computed using L2 norm, different from Hamming or num_correct
model.fit(X, y, epochs=2, batch_size=10)

Epoch 1/2
10000/10000 [=====] - 46s 5ms/step - loss: 0.4649 - accuracy: 0.3072
Epoch 2/2
10000/10000 [=====] - 47s 5ms/step - loss: 0.3669 - accuracy: 0.3852

[66]: <keras.src.callbacks.History at 0x29e2142b0>

[67]: model.predict(X_test[:1])
1/1 [=====] - 0s 32ms/step
[67]: array([[0. , 0.30335504, 0.16753714, 0.4673925 , 0. ,
0. , 0.27270895, 0. , 0. , 0. ,
0.15993657, 0. , 0. , 0.07870372]], dtype=float32)

[68]: predictions = model.predict(X_test)
predictions_rounded = [[round(i) for i in a] for a in predictions]
625/625 [=====] - 1s 1ms/step

[48]: def binary_encode(x):
return np.array([sum(a[i]*2**i for i in range(len(a))) for a in x])

[70]: #when data includes multi-label (more than one 1's) accuracy score fails
pred_encoded = binary_encode(predictions_rounded)
y_test_encoded = binary_encode(np.array(y_test))
accuracy_score(y_test_encoded, pred_encoded)

[70]: 0.0764

[34]: def hamming_score(y_true, y_pred, normalize=True, sample_weight=None):
...
Compute the Hamming score (a.k.a. label-based accuracy) for the multi-label case
```