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[1]. 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 [2], 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 [3].

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, ¿1 disorder] giving 15 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 13 labels, since the maximum sum is 13. Finally, one could use a

[1]https://www.ncbi.nlm.nih.gov/pmc/articles/PMC6783390/

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

[3]https://github.com/jacksonwalters/ml-examples

Figure 1. PCA plot of MNIST handwritten digits data

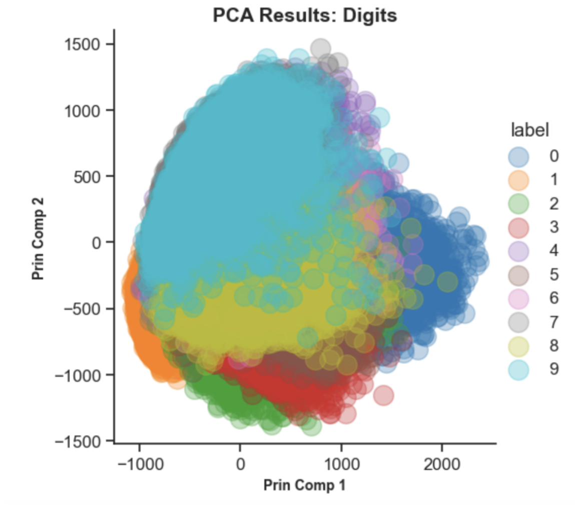
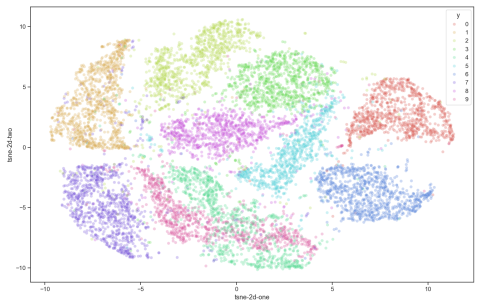


Figure 2. t-SNE plot of MNIST handwritten digits data



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

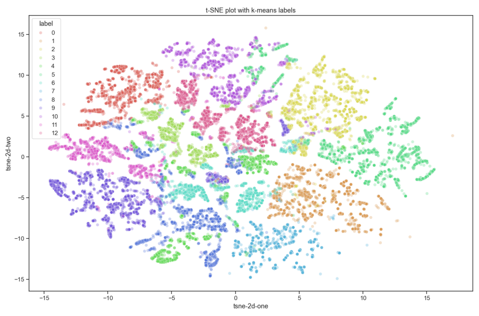
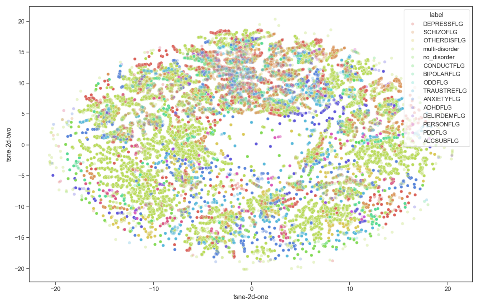


Figure 4. t-SNE plot of mental health data with $unique_disorder$ labeling



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-nearbors 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.

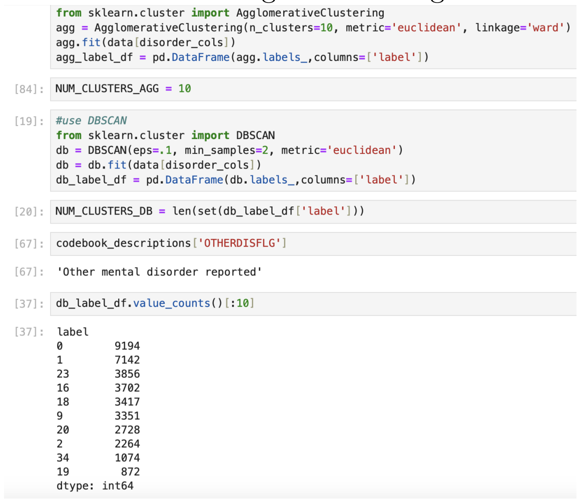
6. Deep Learning and Reinforcement Learning

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.

7. Conclusion

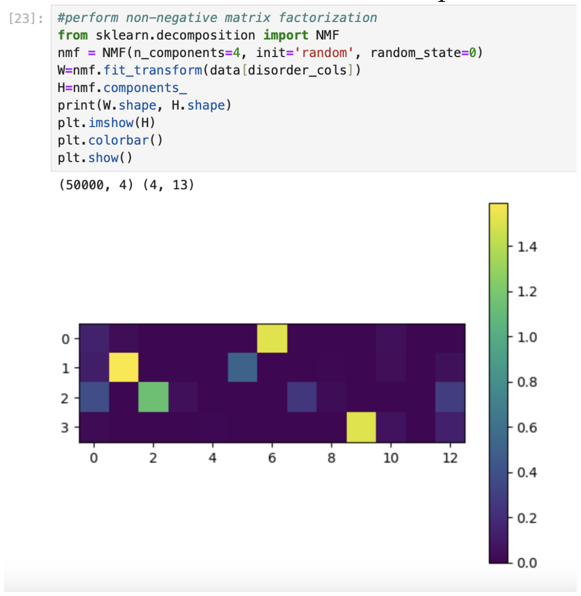
We explored and analyzed the data using a variety of techniques. To detect clustering, we used the unsupervised k-means method and found that it was best, revealing clear clusters when viewed through the lens of a t-SNE plot.

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()[: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
```

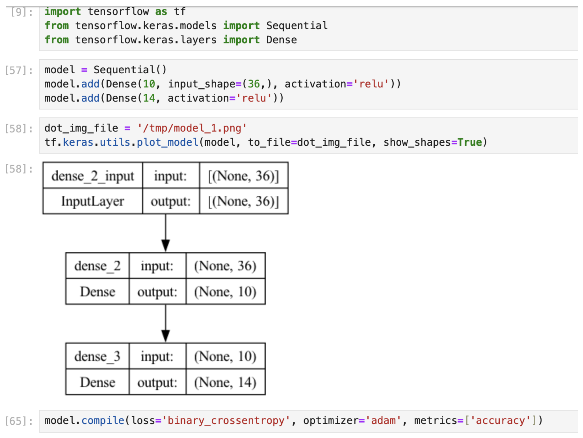
FIGURE 6. matrix factorization plot of H



```
[23]:  #perform non-negative matrix factorization
       from sklearn.decomposition import NMF
       nmf = NMF(n_components=4, init='random', random_state=0)
       W=nmf.fit_transform(data[disorder_cols])
       H=nmf.components_
       print(W.shape, H.shape)
       plt.imshow(H)
       plt.colorbar()
       plt.show()

       (50000, 4) (4, 13)
```

DBSCAN revealed essentially only the relative percentages of each multi-disorder. We learned that this is in fact a multilabel problem, as opposed to a mutliclass problem. However, one can phrase it as a multiclass problem by looking at the $2^14 = 16,384$ distinct multi-disorder classed. Those are reduced to about 400 actual observed multi-disorders, most of which are single, double, or at most triple diagnoses.

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

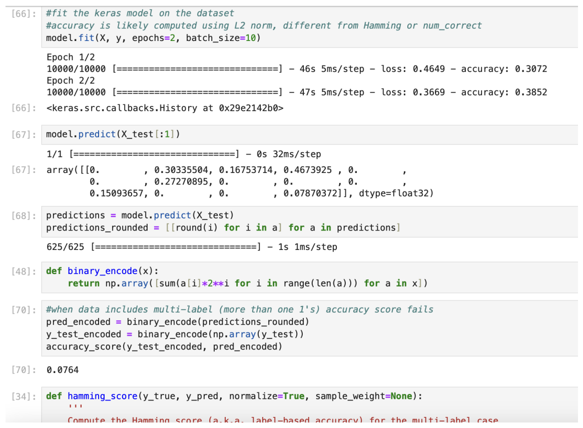


```
[9]:  import tensorflow as tf
      from tensorflow.keras.models import Sequential
      from tensorflow.keras.layers import Dense

[57]: model = Sequential()
      model.add(Dense(10, input_shape=(36,), activation='relu'))
      model.add(Dense(14, activation='relu'))

[58]: dot_img_file = '/tmp/model_1.png'
      tf.keras.utils.plot_model(model, to_file=dot_img_file, show_shapes=True)

[58]:
```

dense_2_input | input: | [(None, 36)]
InputLayer | output: | [(None, 36)]
dense_2 | input: | (None, 36)
Dense | output: | (None, 10)
dense_3 | input: | (None, 10)
Dense | output: | (None, 14)

```
[65]: model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])
```

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.        ,
              0.15093657, 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
```

The goal of this project was to do classfication, and we were able to use a variety of techniques including RandomForest, k-means, and logistic regression. It appears that RandomForest and LogisticRegression are both around 28% accuracy and 19% precision, which beats guessing the majority class which would be 16% accuracy. We don't expect great classification with no symptom-level data, which would be a future direction for this project.