Sensor Embeddings

University of North Texas

Jagdishkumar Katariya (jagdishkumarkatariya@my.unt.edu)

Theophilus Medeiros (theophilusmedeiros@my.unt.edu)

Ryan Moye (ryanmoye@my.unt.edu)

Justin Kim (sunghukim@my.unt.edu)

Sections :

1. Project Name, Participants

1.1 Project Name:

Sensor embedding to improve activity recognition

1.2 The roles for this project are :

- Group 1 Theo & Jagdish:
    - Research on PCA model.
    - Implementation of PCA model.
    - Testing and prediction on new dataset.
    - Confusion Matrix.
    - Work on data visualization.

- Group 2 Ryan & Justin:
    - Research autoencoder applications.
        - https://blog.keras.io/building-autoencoders-in-keras.html
    - Work on data visualization.
    - Confusion Matrix.
    - Overfit/underfit graphs

2. Abstract

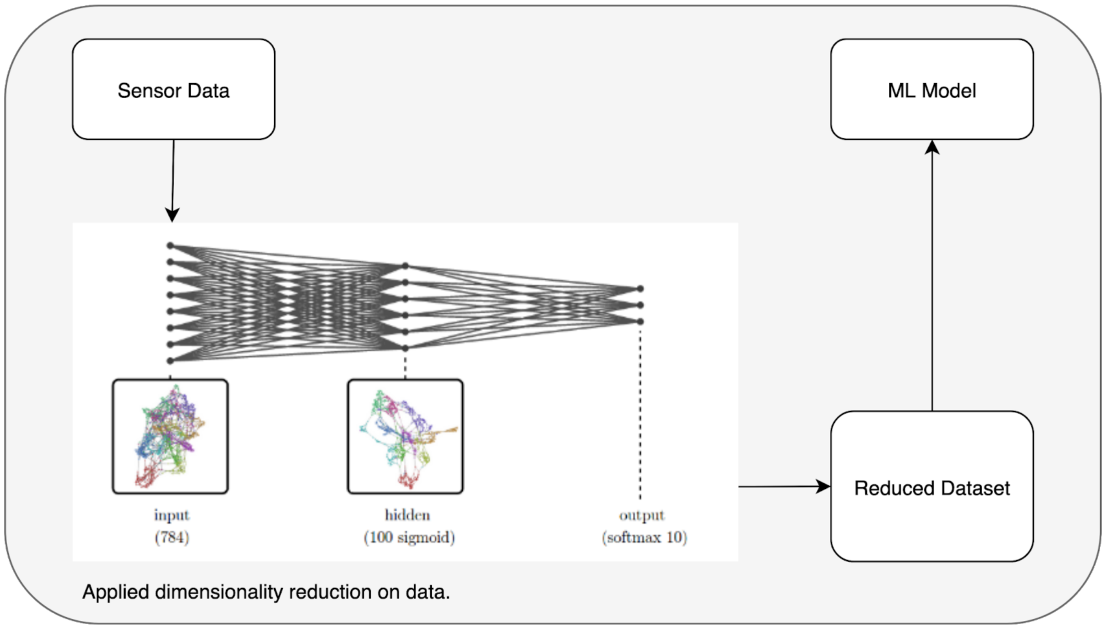
Data collection and retrieval has come a long way in the past few years. Wearable technology, such as smart watches, and even mobile phones collect massive amounts of user data. However, this data can be difficult to work with due to the size and complexity.

One option is to use dimensionality reduction in order to reduce the features represented in the data without reducing the efficiency of our model. We will use the *Human Activity Recognition* (HAR) dataset to test different forms of dimensionality reduction.

Our initial plan was to use Principal Components Analysis (PCA) and Autoencoders for feature reduction and test each method on the Human Activity Recognition Dataset. After completing both models, we compared the results and decided which model is best for the Human activity recognition dataset.

3. Data specification

We used the Human Activity Recognition (HAR) dataset. It has 561 features and 7351 observations. The data is separated into a test and train files which contains the x, y, and z accelerometer and gyroscope data. Each person performed six activities standing, walking, walking upstairs and downstairs, laying and sitting wearing a smartphone on the waist. The movements were collected using an android smartphone using its accelerometer and gyroscope data to gather the test and the train data.



3.1 Goals

1. Find an efficient way to reduce the HAR dataset using PCA
2. Build an autoencoder for dimensionality reduction.
3. Test the efficiency of the reduced dataset and compare it with the original dataset to ensure there is no loss of accuracy.

4. Project Design

4.1 Technology we used :

- Pandas and tensorflow for data manipulations and data pipelines.
- Scikit-learn for data cleaning and preprocessing.
- Scikit-learn and tensorflow 2.0+ for modeling
- Tensorflow and Keras for deep learning
- Jupyter Notebooks for coding and running experiments
- Github for sharing jupyter notebook

4.2 The major areas of the project are :

- PCA
- Autoencoder
- Data Cleaning
- Data Visualizations and Exploration
- Dimensionality Reduction
- Model Selection

We implemented two approaches for feature reduction or dimension reduction one is machine learning model Principal Component Analysis (PCA) and second one is Deep Learning approach Autoencoder.

4.3 Principal Component Analysis (PCA)

Principal Component Analysis or PCA is a widely used technique for dimensionality reduction of the large data set. Reducing the number of components or features costs some accuracy and on the other hand, it makes the large data set simpler, easy to explore and visualize. Also, it reduces the computational complexity of the model which makes machine learning algorithms run faster.

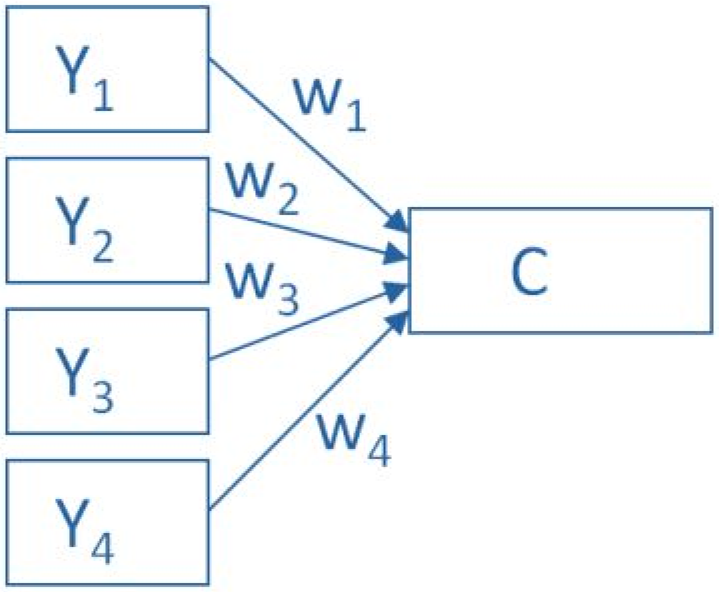
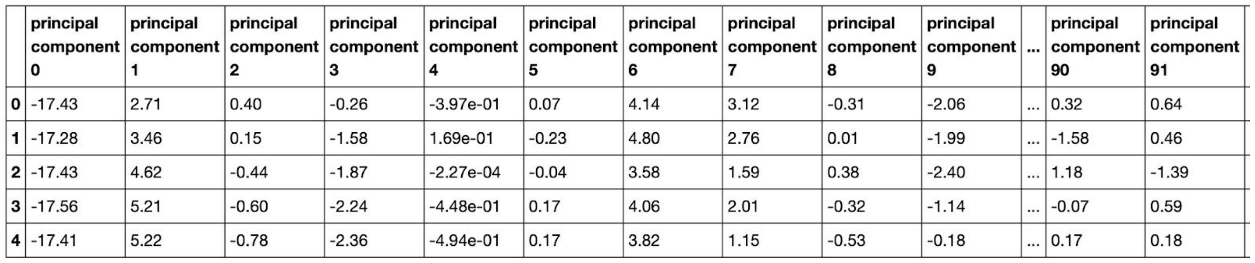


Image 1 : PCA

We use existing dataset and after preprocessing data we applied PCA. Our objective is to reduce the 561 features in the HAR dataset into smaller features while retaining as much as possible the variation present in the original data set. After reducing some features, we started using 2 principle components but we did not get a good result of the confusion matrix so we increased the number of components to 100 and we have gotten good results thus far.

Image 2 is DataFrame of principal components from 1 to 100.



	principal component 0	principal component 1	principal component 2	principal component 3	principal component 4	principal component 5	principal component 6	principal component 7	principal component 8	principal component 9	...	principal component 90	principal component 91
0	-17.43	2.71	0.40	-0.26	-3.97e-01	0.07	4.14	3.12	-0.31	-2.06	...	0.32	0.64
1	-17.28	3.46	0.15	-1.58	1.69e-01	-0.23	4.80	2.76	0.01	-1.99	...	-1.58	0.46
2	-17.43	4.62	-0.44	-1.87	-2.27e-04	-0.04	3.58	1.59	0.38	-2.40	...	1.18	-1.39
3	-17.56	5.21	-0.60	-2.24	-4.48e-01	0.17	4.06	2.01	-0.32	-1.14	...	-0.07	0.59
4	-17.41	5.22	-0.78	-2.36	-4.94e-01	0.17	3.82	1.15	-0.53	-0.18	...	0.17	0.18

Image 2 : Principle components

Image 3 is just plotting the first two principal components. Notice on the graph that the classes seem well separated from each other. We have six activities and in those six activities, three (walking, walking upstairs and downstairs) activities are similar to each other while the remaining three activities (standing, laying and sitting) are also similar to each other are similar. It is worth to note that the in-motion activities are correlated to each other while the static activities are also correlated.

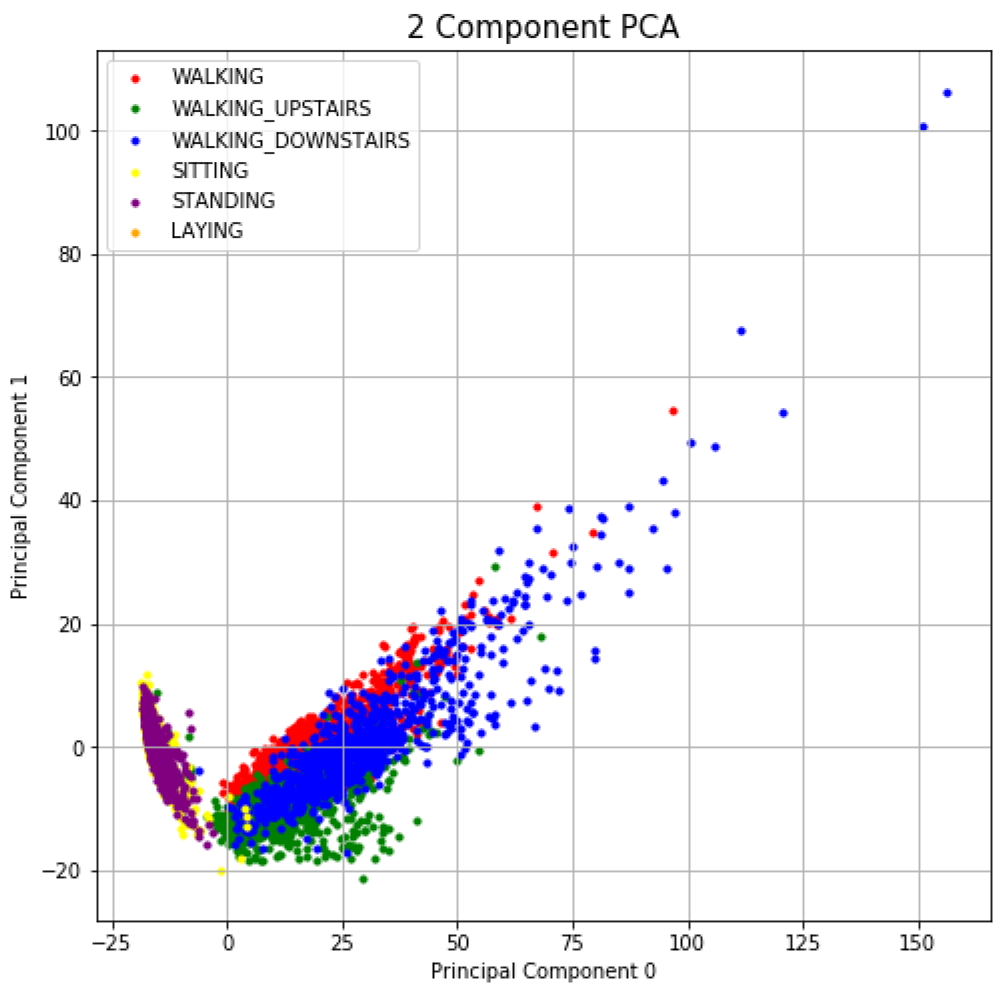


Image 3 : PCA Representation

Image 4 is a curve that quantifies how much of the total, 561-dimensional variance is contained within the first 100 components. For example, we see that the first 20 components contain approximately 80% of the variance, while you need around 100 components to describe close to 98% of the variance.

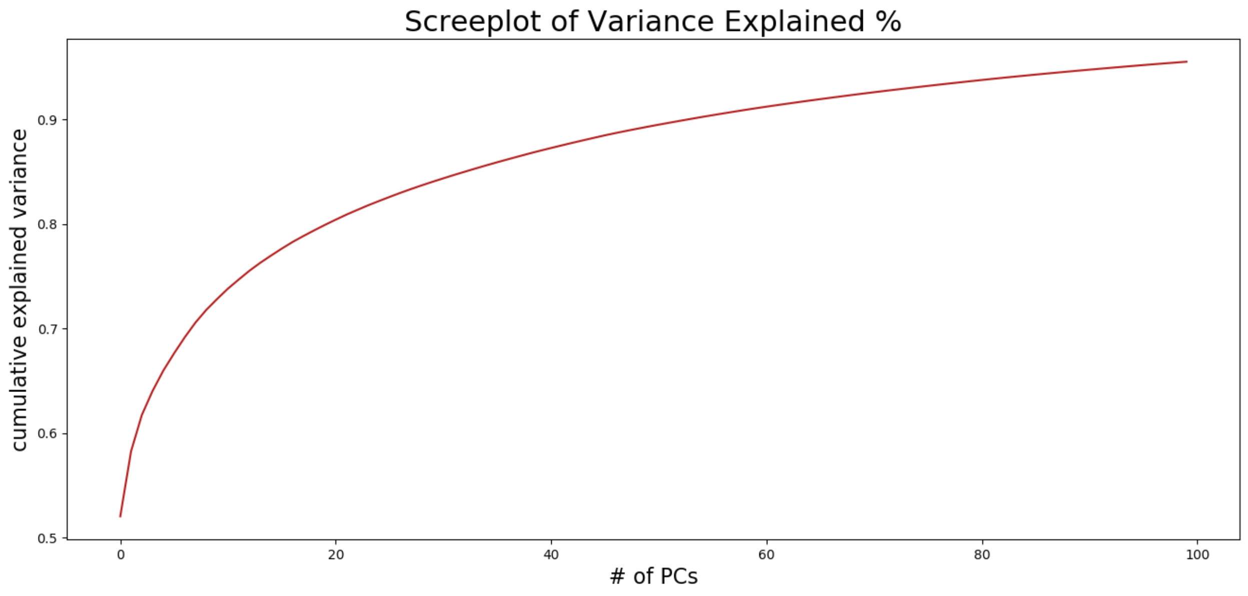
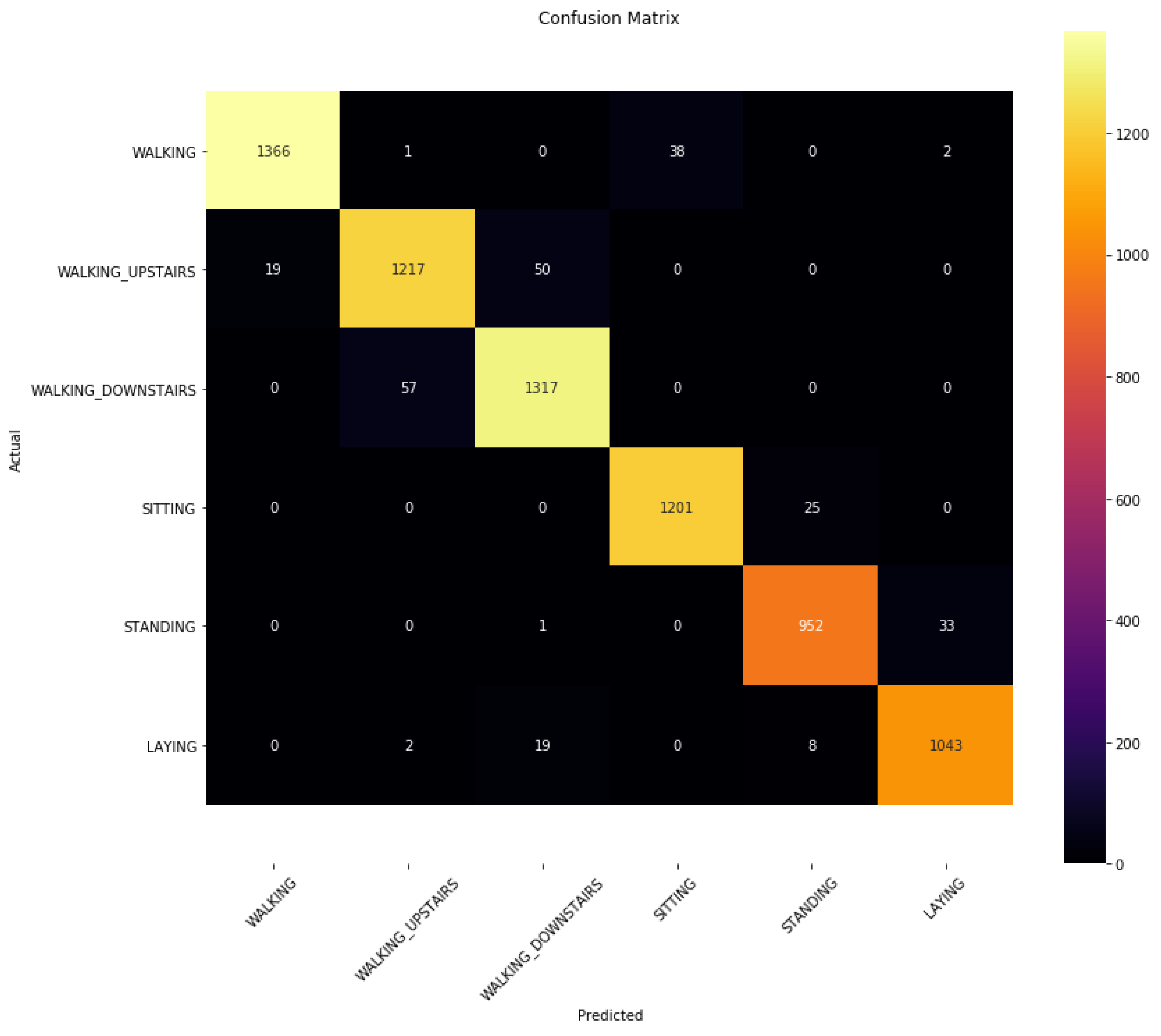


Image 4: Scree Plot of Variance

Image 5 is a confusion matrix that shows that the sensitivity and specificity are very high compared to the false negatives and false positives. This also shows that there is very minimal confusion in the confusion matrix.



Actual \ Predicted	WALKING	WALKING_UPSTAIRS	WALKING_DOWNSTAIRS	SITTING	STANDING	LAYING
WALKING	1366	1	0	38	0	2
WALKING_UPSTAIRS	19	1217	50	0	0	0
WALKING_DOWNSTAIRS	0	57	1317	0	0	0
SITTING	0	0	0	1201	25	0
STANDING	0	0	1	0	952	33
LAYING	0	2	19	0	8	1043

Image 5 : Confusion Matrix

```
from sklearn.metrics import accuracy_score

print('Accuracy ',accuracy_score(y_validation, yhat_validation)*100)
```

```
Accuracy  96.53108420623045
```

We are getting 96.53% accuracy in a simple model after reduction feature using the PCA model.

4.5 Auto Encoders

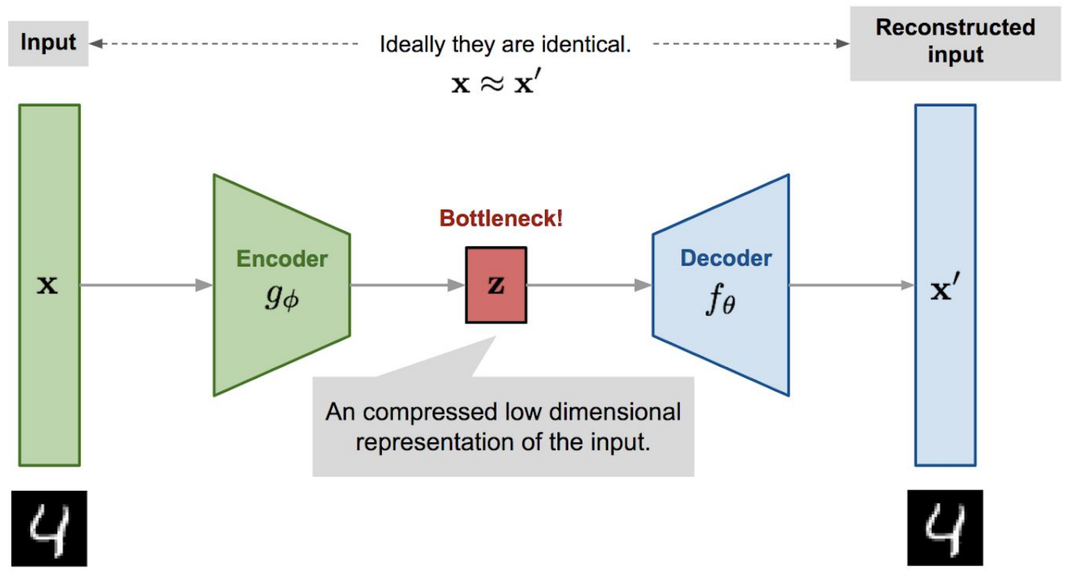
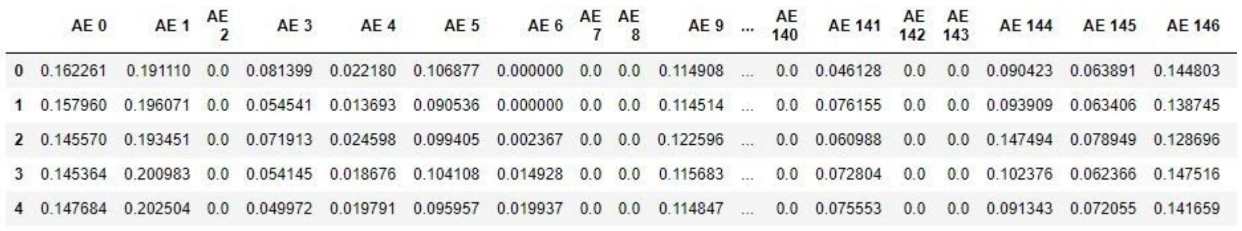

Input
Ideally they are identical.
$\mathbf{x} \approx \mathbf{x}'$
Reconstructed input
$\mathbf{x}$
Encoder
g_ϕ
Bottleneck!
$\mathbf{z}$
Decoder
f_θ
$\mathbf{x}'$
An compressed low dimensional representation of the input.

Image 6 : Autoencoders

The auto encoder was created by using keras to form a basic autoencoder. Autoencoders encode data into a smaller dimensionality and decode it while trying to reduce loss. First, we used a normalizer from sklearn in order to normalize the dense dataset so that it can be encoded. After normalizing the data, we created a two layer encoder. The activation of the encoder is relu. We would then set up the model by using the compile function while using the metrics: mean_squared_error.

After setting up the model, it was tested using 500 epochs to ensure that it will have enough datasets to test upon. It was concluded that the accuracy was around 79%. This told us that the auto encoder did work as the model received 79% accuracy when reducing the original feature set from 561 dimensions to 150 dimensions.



	AE 0	AE 1	AE 2	AE 3	AE 4	AE 5	AE 6	AE 7	AE 8	AE 9	...	AE 140	AE 141	AE 142	AE 143	AE 144	AE 145	AE 146
0	0.162261	0.191110	0.0	0.081399	0.022180	0.106877	0.000000	0.0	0.0	0.114908	...	0.0	0.046128	0.0	0.0	0.090423	0.063891	0.144803
1	0.157960	0.196071	0.0	0.054541	0.013693	0.090536	0.000000	0.0	0.0	0.114514	...	0.0	0.076155	0.0	0.0	0.093909	0.063406	0.138745
2	0.145570	0.193451	0.0	0.071913	0.024598	0.099405	0.002367	0.0	0.0	0.122596	...	0.0	0.060988	0.0	0.0	0.147494	0.078949	0.128696
3	0.145364	0.200983	0.0	0.054145	0.018676	0.104108	0.014928	0.0	0.0	0.115683	...	0.0	0.072804	0.0	0.0	0.102376	0.062366	0.147516
4	0.147684	0.202504	0.0	0.049972	0.019791	0.095957	0.019937	0.0	0.0	0.114847	...	0.0	0.075553	0.0	0.0	0.091343	0.072055	0.141659

Figure 7: Autoencoder Dimensions

Above in figure 7 we see the weights associated with the encoded feature set. Below in figure 8 we see the scatterplot showing the first two encoded features. This scatterplot is set up in the same manner as the PCA plot in figure 3.

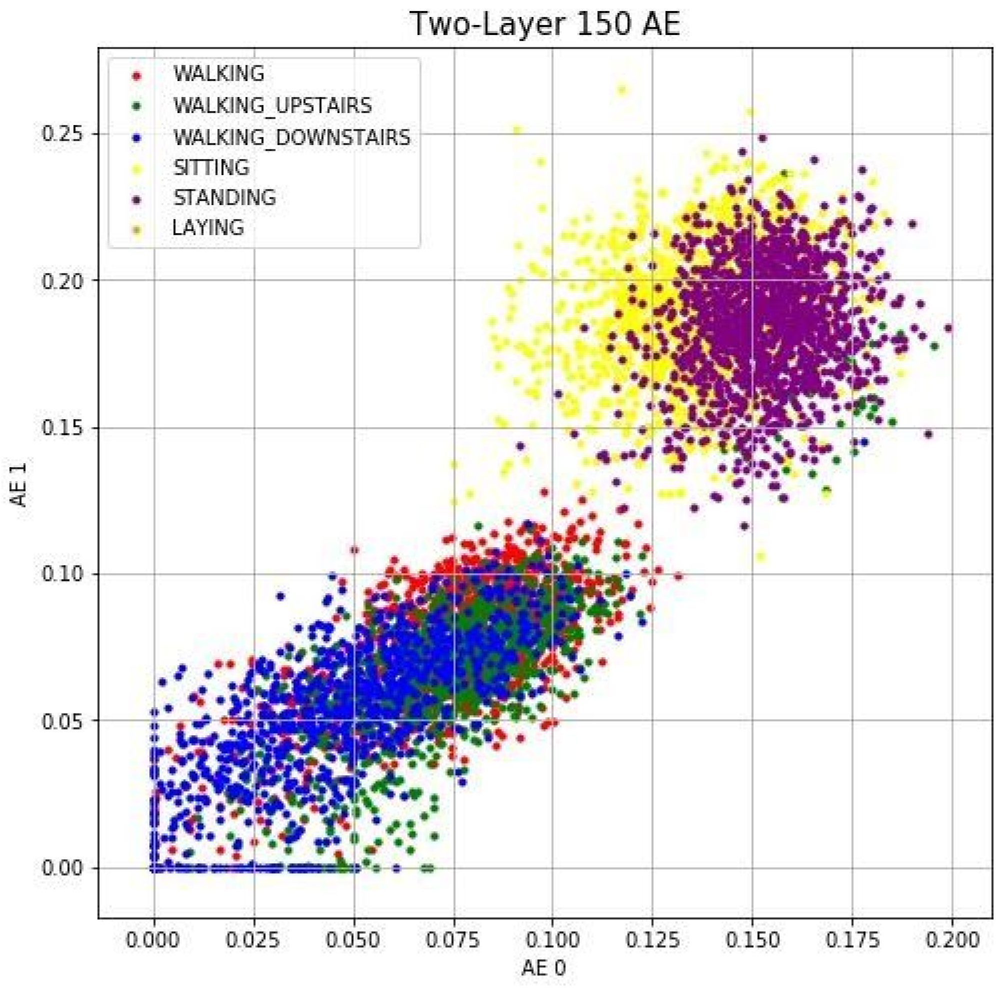


Figure 8: Scatterplot Representation of the Autoencoder

5. Milestones

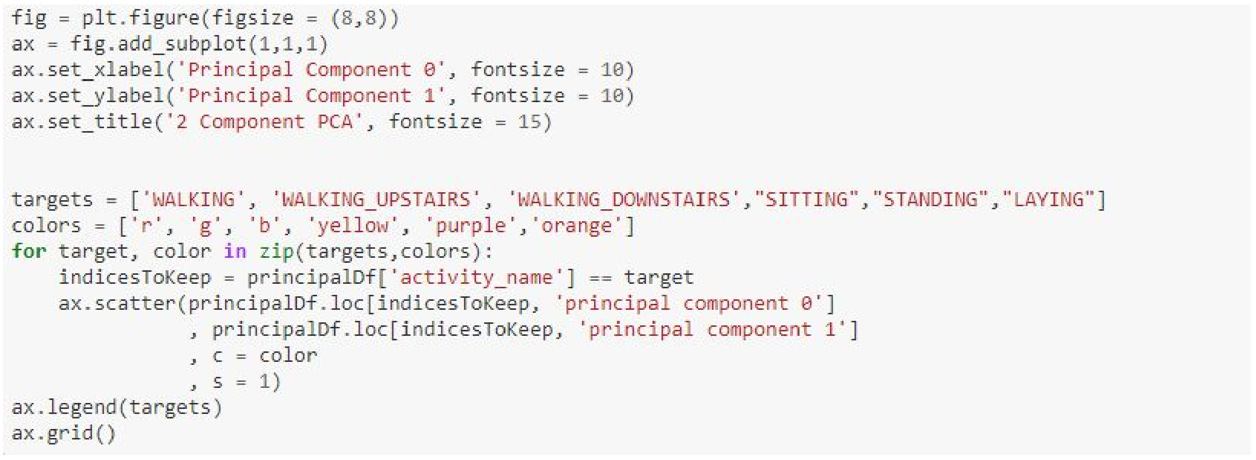
Name	Due Date	Meeting
Found Dataset of accelerometer and gyroscope (HAR Dataset)	08/27/2020	Zoom meeting
Project ideas discussed with professor and found some resources	08/31/2020	Zoom meeting
Complete Project proposal	09/02/2020	Zoom meeting
Data Cleaning and finding appropriate model	09/2/2020	Zoom meeting
Implementation of PCA	09/9/2020	Zoom meeting
Complete Project progress report and video	09/9/2020	Zoom meeting
Implementation of Confusion Matrix and Scree plot of variance	09/11/2020	Zoom meeting
Implementation of Autoencoder	09/16/2020	Zoom meeting
● Project Presentation ● Project report	09/16/2020	Zoom meeting

● Project video		

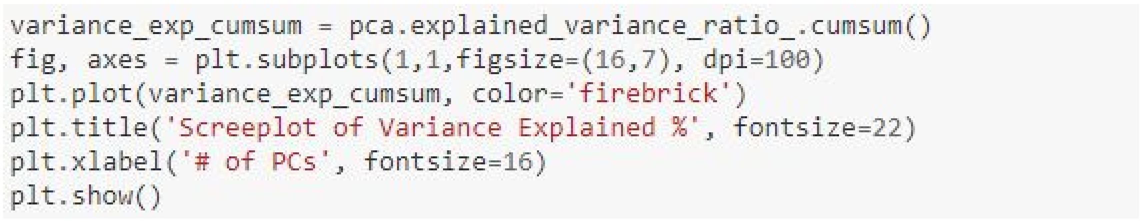
6. Repository

- Github Repository containing all of our code.
    - https://github.com/idaf-ai/sensor-embeddings

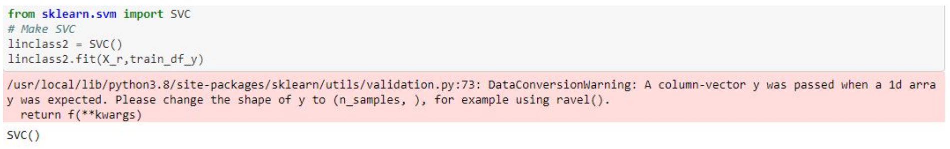
Code Pictures:



```python
fig = plt.figure(figsize = (8,8))
ax = fig.add_subplot(1,1,1)
ax.set_xlabel('Principal Component 0', fontsize = 10)
ax.set_ylabel('Principal Component 1', fontsize = 10)
ax.set_title('2 Component PCA', fontsize = 15)


targets = ['WALKING', 'WALKING_UPSTAIRS', 'WALKING_DOWNSTAIRS',"SITTING","STANDING","LAYING"]
colors = ['r', 'g', 'b', 'yellow', 'purple','orange']
for target, color in zip(targets,colors):
    indicesToKeep = principalDf['activity_name'] == target
    ax.scatter(principalDf.loc[indicesToKeep, 'principal component 0']
               , principalDf.loc[indicesToKeep, 'principal component 1']
               , c = color
               , s = 1)
ax.legend(targets)
ax.grid()
```

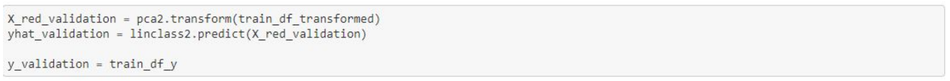


```python
variance_exp_cumsum = pca.explained_variance_ratio_.cumsum()
fig, axes = plt.subplots(1,1,figsize=(16,7), dpi=100)
plt.plot(variance_exp_cumsum, color='firebrick')
plt.title('Screeplot of Variance Explained %', fontsize=22)
plt.xlabel('# of PCs', fontsize=16)
plt.show()
```

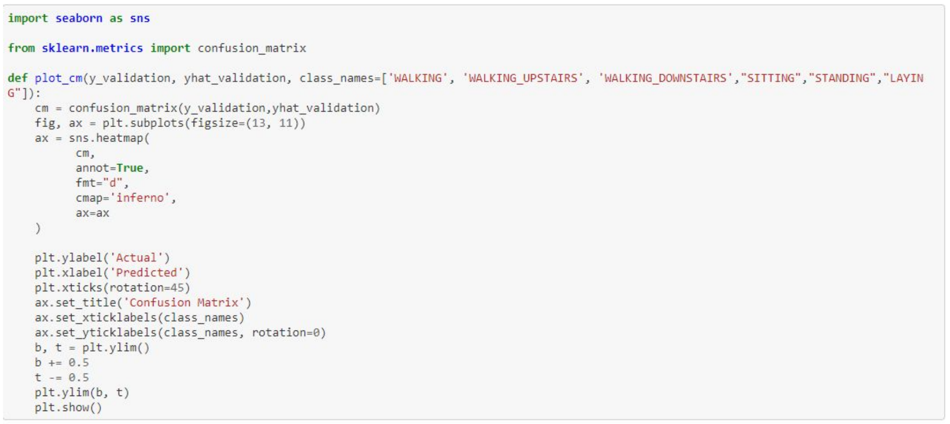


```python
from sklearn.svm import SVC
# Make SVC
linclass2 = SVC()
linclass2.fit(X_r,train_df_y)
```

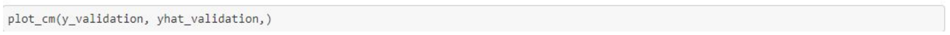
```
/usr/local/lib/python3.8/site-packages/sklearn/utils/validation.py:73: DataConversionWarning: A column-vector y was passed when a 1d arra
y was expected. Please change the shape of y to (n_samples, ), for example using ravel().
  return f(**kwargs)
SVC()
```



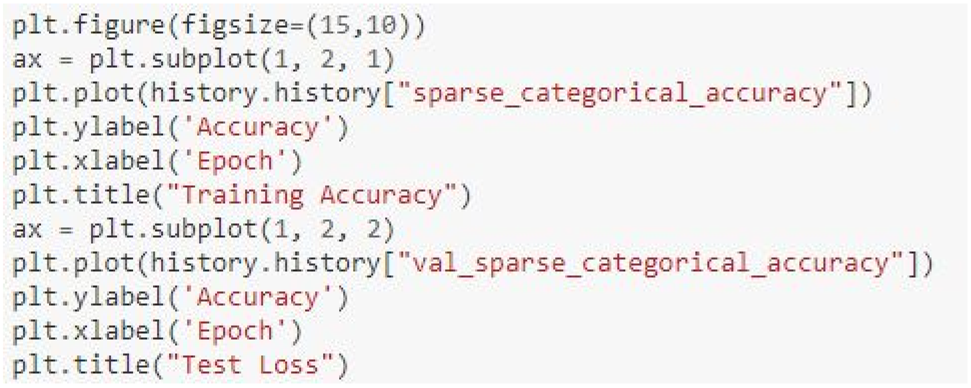
```python
X_red_validation = pca2.transform(train_df_transformed)
yhat_validation = linclass2.predict(X_red_validation)

y_validation = train_df_y
```



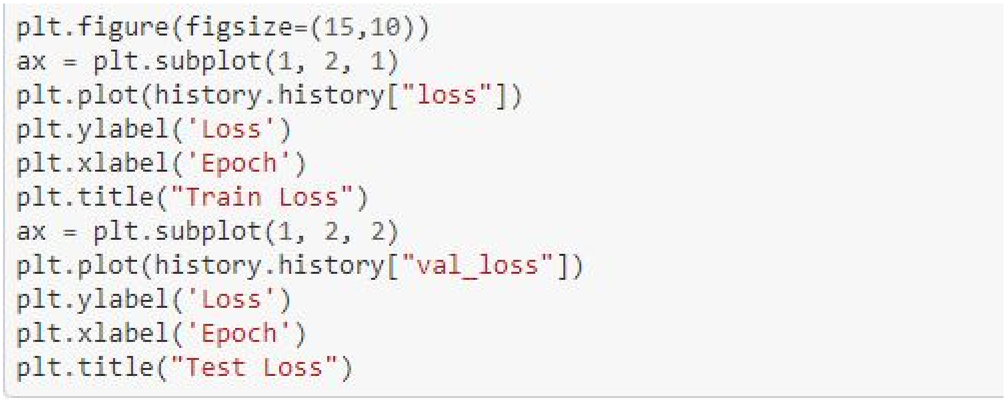
```python
import seaborn as sns

from sklearn.metrics import confusion_matrix

def plot_cm(y_validation, yhat_validation, class_names=['WALKING', 'WALKING_UPSTAIRS', 'WALKING_DOWNSTAIRS',"SITTING","STANDING","LAYIN
G"]):
    cm = confusion_matrix(y_validation,yhat_validation)
    fig, ax = plt.subplots(figsize=(13, 11))
    ax = sns.heatmap(
          cm,
          annot=True,
          fmt="d",
          cmap='inferno',
          ax=ax
    )

    plt.ylabel('Actual')
    plt.xlabel('Predicted')
    plt.xticks(rotation=45)
    ax.set_title('Confusion Matrix')
    ax.set_xticklabels(class_names)
    ax.set_yticklabels(class_names, rotation=0)
    b, t = plt.ylim()
    b += 0.5
    t -= 0.5
    plt.ylim(b, t)
    plt.show()
```



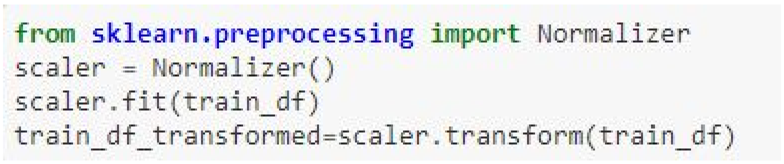
```python
plot_cm(y_validation, yhat_validation,)
```



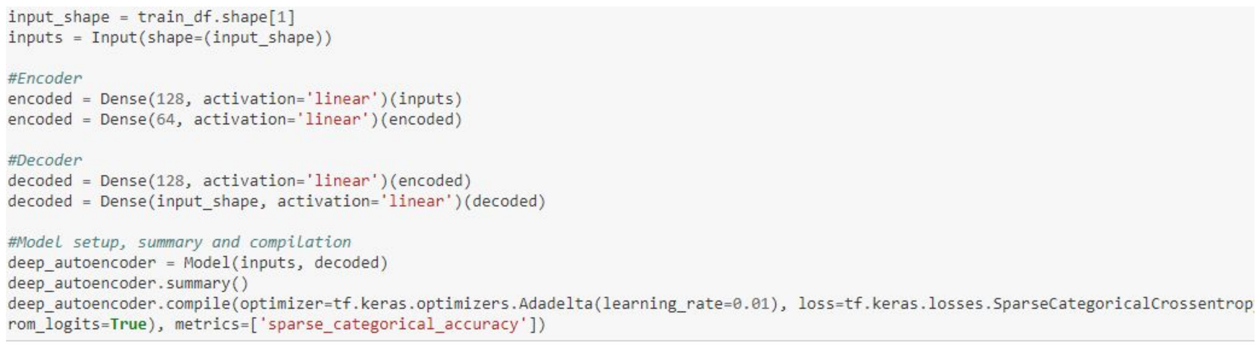
```
plt.figure(figsize=(15,10))
ax = plt.subplot(1, 2, 1)
plt.plot(history.history["sparse_categorical_accuracy"])
plt.ylabel('Accuracy')
plt.xlabel('Epoch')
plt.title("Training Accuracy")
ax = plt.subplot(1, 2, 2)
plt.plot(history.history["val_sparse_categorical_accuracy"])
plt.ylabel('Accuracy')
plt.xlabel('Epoch')
plt.title("Test Loss")
```



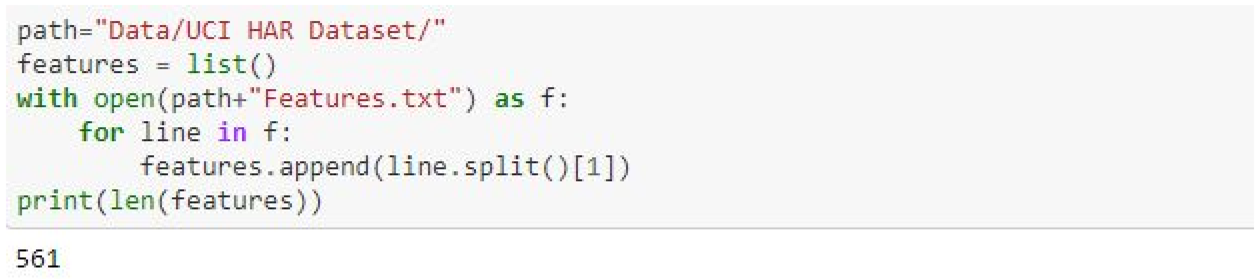
```
plt.figure(figsize=(15,10))
ax = plt.subplot(1, 2, 1)
plt.plot(history.history["loss"])
plt.ylabel('Loss')
plt.xlabel('Epoch')
plt.title("Train Loss")
ax = plt.subplot(1, 2, 2)
plt.plot(history.history["val_loss"])
plt.ylabel('Loss')
plt.xlabel('Epoch')
plt.title("Test Loss")
```



```
from sklearn.preprocessing import Normalizer
scaler = Normalizer()
scaler.fit(train_df)
train_df_transformed=scaler.transform(train_df)
```

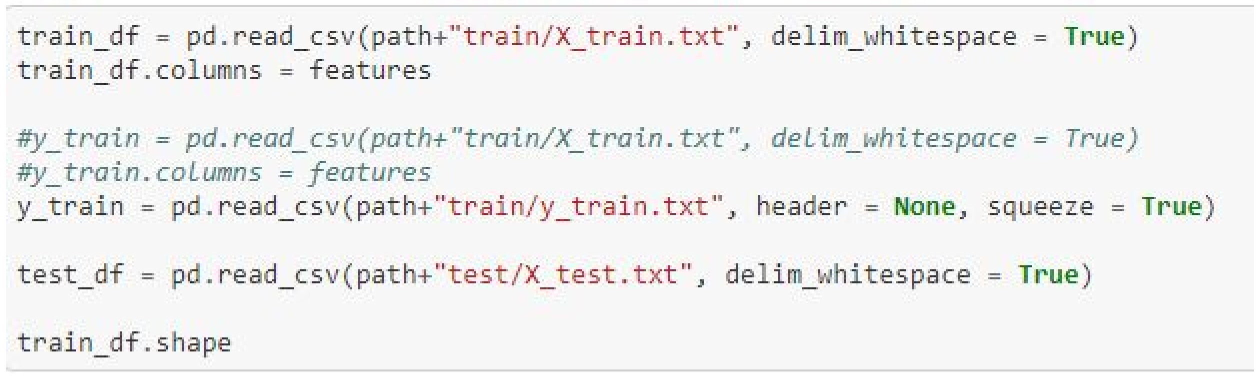


```
input_shape = train_df.shape[1]
inputs = Input(shape=(input_shape))

#Encoder
encoded = Dense(128, activation='linear')(inputs)
encoded = Dense(64, activation='linear')(encoded)

#Decoder
decoded = Dense(128, activation='linear')(encoded)
decoded = Dense(input_shape, activation='linear')(decoded)

#Model setup, summary and compilation
deep_autoencoder = Model(inputs, decoded)
deep_autoencoder.summary()
deep_autoencoder.compile(optimizer=tf.keras.optimizers.Adadelta(learning_rate=0.01), loss=tf.keras.losses.SparseCategoricalCrossentrop
rom_logits=True), metrics=['sparse_categorical_accuracy'])
```



```
path="Data/UCI HAR Dataset/"
features = list()
with open(path+"Features.txt") as f:
    for line in f:
        features.append(line.split()[1])
print(len(features))
```

```
561
```



```
train_df = pd.read_csv(path+"train/X_train.txt", delim_whitespace = True)
train_df.columns = features

#y_train = pd.read_csv(path+"train/X_train.txt", delim_whitespace = True)
#y_train.columns = features
y_train = pd.read_csv(path+"train/y_train.txt", header = None, squeeze = True)

test_df = pd.read_csv(path+"test/X_test.txt", delim_whitespace = True)

train_df.shape
```

7. Reference Material

- This tutorial provides basic introduction to PCA

    - https://towardsdatascience.com/pca-using-python-scikit-learn-e653f8989e60
- This project is about Human Activity Recognition (HAR) database built from the recordings of 30 subjects performing activities of daily living (ADL) while carrying a waist-mounted smartphone with embedded inertial sensors.
    - http://archive.ics.uci.edu/ml/datasets/Human+Activity+Recognition+Using+Smartphones#
- This tutorial shows how to use embeddings and also teaches how to train meaningful embeddings (using word2vec, for example).
    - https://developers.google.com/machine-learning/crash-course/embeddings/video-lecture
    - https://www.jeremyjordan.me/autoencoders/
        - Autoencoder introduction
- This tutorial shines more light into using embeddings through the lens of word2vec
    - https://towardsdatascience.com/introduction-to-word-embedding-and-word2vec-652d0c2060fa
- This project aims to build a model that predicts the human activities
    - https://github.com/gauravtheP/Human-Activity-Recognition
- The goal of this project is to classify the actions taken by the user (walking, climbing stairs, and descending stairs) from the 3D accelerometer data.
    - https://github.com/srane96/Activity-Recognition-Using-Accelerometer-Data
    - https://github.com/SunghuJustinKim/GestureRecognition
        - The TalkMotion model from summer