```
import pandas as pd
import numpy as np
import keras
from keras import layers
```

Upload the labels.csv and processed_counts.csv files to colab or your local workspace.

Copied from Part 1: This data associates a cell barcode, such as "AAAGCCTGGCTAAC-1", to a certain cell type label, such as "CD14+ Monocyte". For each cell barcode, there are also log RNA seq counts of 765 different genes, such as HES4.

label.csv stores the association between a cell barcode and a cell type label.

processed_counts.csv stores the normalized log read counts for each cell, where each row represents a single cell, and each column represents a gene.

```
labels_pd = pd.read_csv("labels.csv")
counts_pd = pd.read_csv("processed_counts.csv")
processed_pd = pd.read_csv("encoded_counts.csv")
```

```
df1 = processed_pd.merge(labels_pd,left_index=True, right_index=True)
df1.drop("Unnamed: 0", axis=1, inplace=True)
df1.drop("index", axis=1, inplace=True)

df1
```

	0	1	2	3	4	5	6	7	8	
0	0.000000	0.000000	0.000000	8.503382	0.000000	0.0	0.000000	0.0	0.000000	12.0071

```
labels_pd.index = labels_pd['index']
labels_pd.drop("index", axis=1, inplace=True)
counts_pd.index = counts_pd['Unnamed: 0']
counts_pd.drop("Unnamed: 0", axis=1, inplace=True)

df = counts_pd.merge(labels_pd, left_index=True, right_index=True).dropna()
df
```

One-hot encode the cell-type.

Shuffle your data. Make sure your labels and the counts are shuffled together.

Split into train and test sets (80:20 split)

```
#splitting up the total bulk data
categories = df['bulk_labels'].unique()
print(categories)

#one-hot encoding
y = np.zeros((len(df), len(categories)))
for i in range(len(df)):
    cell_type = df.iloc[i]['bulk_labels']
    pos = np.where(categories == cell_type)[0]
    y[i, pos] = 1

#remove label when processing input data
X = df.drop('bulk_labels', axis=1).values

#shufle and 80:20 split
np.random.seed(100)
permutation = np.random.permutation(len(X))
X, y = X[permutation], y[permutation]

X_train, y_train = X[:int(len(X)*0.8)], y[:int(len(y)*0.8)]
X_test, y_test = X[int(len(X)*0.8):], y[int(len(y)*0.8):]
```

```
['CD14+ Monocyte' 'Dendritic' 'CD56+ NK' 'CD4+/CD25 T Reg' 'CD19+ B'
 'CD8+ Cytotoxic T' 'CD4+/CD45RO+ Memory' 'CD8+/CD45RA+ Naive Cytotoxic'
 'CD4+/CD45RA+/CD25- Naive T' 'CD34+']
```

```
#splitting up the enbedded data
categories = df1['bulk_labels'].unique()
print(categories)

#one-hot encoding
y = np.zeros((len(df1), len(categories)))
for i in range(len(df1)):
    cell_type = df1.iloc[i]['bulk_labels']
    pos = np.where(categories == cell_type)[0]
    y[i, pos] = 1

#remove label when processing input data
X = df1.drop('bulk_labels', axis=1).values

encoded_clean = X
encoded_clean_labels = y

#shufle and 80:20 split
np.random.seed(100)
permutation = np.random.permutation(len(X))
X, y = X[permutation], y[permutation]
```

```
X_train1, y_train1 = X[:int(len(X)*0.8)], y[:int(len(y)*0.8)]
X_test1, y_test1 = X[int(len(X)*0.8):], y[int(len(y)*0.8):]
```

```
['CD14+ Monocyte' 'Dendritic' 'CD56+ NK' 'CD4+/CD25 T Reg' 'CD19+ B'
 'CD8+ Cytotoxic T' 'CD4+/CD45RO+ Memory' 'CD8+/CD45RA+ Naive Cytotoxic'
```

```
#print(X_train)
#print(y_train) #one hot encoded for different cell types

print(encoded_clean.shape)
print(encoded_clean_labels.shape)
```

```
(700, 32)
(700, 10)
```

```
#Visualize the One-hot encoded Prediction Labels
import matplotlib.pyplot as plt
plt.figure(figsize=(9,3), dpi=300)
plt.imshow(y_train[:50])
```

Apply classification algorithms to the training data, tune on validation data (if present), and evaluate on test data.

You can also apply classification downstream of last week's autoencoder latent space representation.

```
# Decision trees: https://scikit-learn.org/stable/modules/tree.html
# Ensemble methods: https://scikit-learn.org/stable/modules/ensemble.html
```

```
     (560, 765)
```

```
#Decision Tree####
##################

#have to be mindful of decision trees as they tend to overfit to their paritulcar data
#and if you had variance to a dataset you are likley to get an entirly different tree

#Could do PCA before on all the data or just use the embedded data

from sklearn import tree
#use embedding for the decision tree and ensemble method

from sklearn.model_selection import cross_val_score
from sklearn.tree import DecisionTreeClassifier
# 3,5 ,5

clf = DecisionTreeClassifier(random_state=0,max_depth = 20, min_samples_split = 7, mir

# max depth is how deep the tree is allowed to be (lower)
# min_samples split is how many samples at each split node,  (higher)
# min_samples_leaf is how many samples per leaf --> if low then very complex. (higher)

cross_val_score(clf, encoded_clean, encoded_clean_labels, cv=5).mean()
```

```
0.667142857142857
```

```
#random forest####
##################


#randomly pick a subset of features to work with
from sklearn.ensemble import RandomForestClassifier


#n estimators is the number of decision trees you are spitting up
#max_features = 'sqrt',5 --> lower their will be less variance
#min_samples_split


clf = RandomForestClassifier(n_estimators=15)#,min_samples_split=2)#,max_features = 5)
clf = clf.fit(X_train1,y_train1)

#take average of predctions

clf.score(X_test1, y_test1)
```

```
0.7071428571428572
```

```
#Ensemble bagging and random forest#
####################################

#average the predictions for multiple models
#each working with a subset of the data

from sklearn.ensemble import BaggingClassifier

#max samples is what proportion of the data is it going to train on
#max_features is the max num of features each will train on

bagging = BaggingClassifier(clf, max_samples=0.5, max_features=0.5)


clf = clf.fit(X_train1,y_train1)
clf.score(X_test1, y_test1)
```

```
    0.7357142857142858
```

```
# FFNN hints:

# Use softmax at the end, reLU for the rest
# Add layers until desired loss

# Categorical cross-entropy for loss func

# Add dropout layers to avoid overfitting

# Can also do bagging with FFNNs (but probably not necessary): https://machinelearning

#use all the data here
import keras
from keras import layers
from keras import regularizers
import tensorflow as tf

model = keras.Sequential()

model.add(tf.keras.Input(shape=(765)))


model.add(tf.keras.layers.Dense(100, input_dim=32, activation='relu'))
model.add(tf.keras.layers.Dropout(0.3))


model.add(tf.keras.layers.Dense(100, input_dim=32, activation='relu'))
model.add(tf.keras.layers.Dropout(0.3))

#model.add(tf.keras.layers.Dense(200, input_dim=32, activation='relu'))
#model.add(tf.keras.layers.Dropout(0.2))

model.add(tf.keras.layers.Dense(100, input_dim=32, activation='relu'))
model.add(tf.keras.layers.Dropout(0.3))


model.add(tf.keras.layers.Dense(10, activation='softmax'))
model.compile(loss="categorical_crossentropy", optimizer='adam', metrics=['accuracy']


model.fit(X_train, y_train, epochs=15)
```

```
    Epoch 1/15
    18/18 [==============================] - 1s 4ms/step - loss: 2.1427 - accuracy:
    Epoch 2/15
    18/18 [==============================] - 0s 4ms/step - loss: 1.3655 - accuracy:
    Epoch 3/15
    18/18 [==============================] - 0s 4ms/step - loss: 1.0768 - accuracy:
    Epoch 4/15
```

```
18/18 [==============================] - 0s 4ms/step - loss: 0.8016 - accuracy: 
Epoch 5/15
18/18 [==============================] - 0s 4ms/step - loss: 0.6433 - accuracy: 
Epoch 6/15
18/18 [==============================] - 0s 4ms/step - loss: 0.5296 - accuracy: 
Epoch 7/15
18/18 [==============================] - 0s 4ms/step - loss: 0.4043 - accuracy: 
Epoch 8/15
18/18 [==============================] - 0s 4ms/step - loss: 0.3434 - accuracy: 
Epoch 9/15
18/18 [==============================] - 0s 4ms/step - loss: 0.2942 - accuracy: 
Epoch 10/15
18/18 [==============================] - 0s 5ms/step - loss: 0.2411 - accuracy: 
Epoch 11/15
18/18 [==============================] - 0s 4ms/step - loss: 0.1919 - accuracy: 
Epoch 12/15
18/18 [==============================] - 0s 4ms/step - loss: 0.1982 - accuracy: 
Epoch 13/15
18/18 [==============================] - 0s 4ms/step - loss: 0.1431 - accuracy: 
Epoch 14/15
18/18 [==============================] - 0s 4ms/step - loss: 0.1366 - accuracy: 
Epoch 15/15
18/18 [==============================] - 0s 3ms/step - loss: 0.1050 - accuracy: 
<keras.callbacks.History at 0x7fb983f67d50>
```

```
# evaluate the model

#X_train, y_train
#X_test, y_test


#0.9 achevable

_, train_acc = model.evaluate(X_train, y_train, verbose=0)
_, test_acc = model.evaluate(X_test, y_test, verbose=0)
print('Train: %.3f, Test: %.3f' % (train_acc, test_acc))
```

```
Train: 1.000, Test: 0.843
```