CS561 HW12

November 27, 2023

1 Problem 1

A mixture model assumes that distribution of the data is composed of a mixture of several base distributions:

$$p(\mathbf{x}) = \sum_{k=1}^{K} \pi_k p_k(\mathbf{x})$$

Let μ_k and Σ_k for $k = 1, \dots, K$ be the mean and covariance matrix of the K base distributions.

1.a) We introduce a lable l that indicates which cluster $\mathbf{x}$ is sampled from (l takes the value of $1, \dots, k$). This means $\mathbb{P}(l = k) = \pi_k$ and $p(\mathbf{x}|l = k) = \mathcal{N}(\mu_k, \Sigma_k)$. We refer to the property of conditional expectations

$$\begin{aligned}
\mathbb{E}[\mathbf{x}] &= \mathbb{E}[\mathbb{E}[\mathbf{x}|l]] \\
&= \sum_{k=1}^{K} \mathbb{E}[\mathbf{x}|l = k]\mathbb{P}(l = k) \\
&= \sum_{k=1}^{K} \pi_k \mu_k
\end{aligned}$$

1.b) Similar to above, we refer to the property of conditional expectations

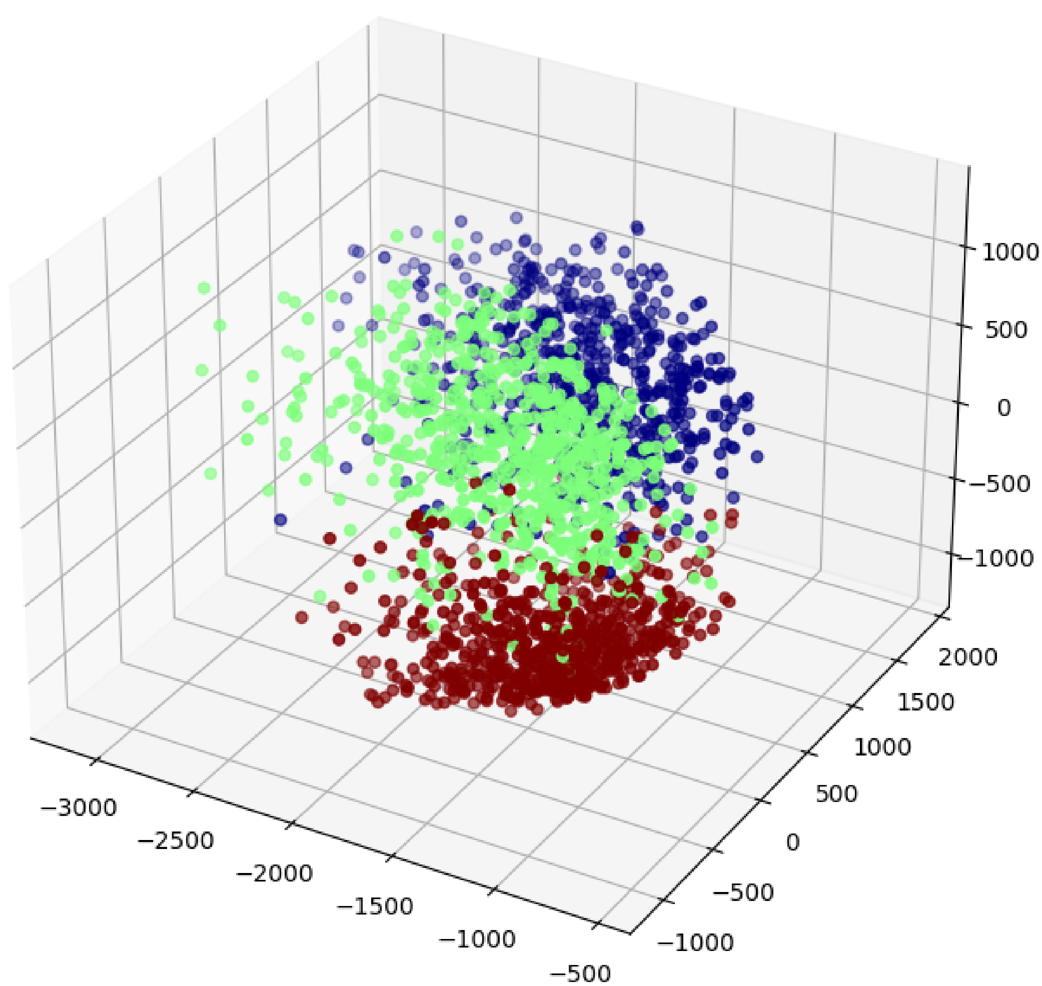
$$\begin{aligned}
\mathrm{Var}(\mathbf{x}) &= \mathbb{E}[\mathrm{Var}(\mathbf{x}|l)] + \mathrm{Var}(\mathbb{E}[\mathbf{x}|l]) \\
&= \sum_{k=1}^{K} \mathrm{Var}(\mathbf{x}|l = k)\mathbb{P}(l = k) + \sum_{k=1}^{K} \mathbb{P}(l = k)||\mathbb{E}[\mathbf{x}|l = k] - \mathbb{E}[\mathbb{E}[\mathbf{x}|l]]||_2^2 \\
&= \sum_{k=1}^{K} \pi_k \Sigma_k + \sum_{k=1}^{K} \pi_k \left(\mu_k - \sum_{k=1}^{K} \pi_k \mu_k\right)^T \left(\mu_k - \sum_{k=1}^{K} \pi_k \mu_k\right)
\end{aligned}$$

2 Problem 2

[1]:
```python
### Load the dataset (which is saved as a pickle file)
import numpy as np
import matplotlib.pyplot as plt
import pickle

with open('dataset.pkl', 'rb') as f:  # Python 3: open(..., 'rb')
    x_train, y_train, x_test, y_test  = pickle.load(f)
```

```python
# Note that each data point is a row
print('x_train has shape:', np.shape(x_train))
print('x_test has shape:', np.shape(x_test))

### Interactive scatter plot of dataset
from mpl_toolkits.mplot3d import Axes3D

fig = plt.figure(figsize=(8, 8))
ax = fig.add_subplot(111, projection='3d')
ax.scatter(x_train[:,0], x_train[:,1], x_train[:,2], c=y_train, cmap='jet')
plt.show()
```

```
x_train has shape: (2000, 3)
x_test has shape: (1018, 3)
```



[2]:
```python
### From an earlier activity
### compute the mean of the three classes, return a column vector

# complete the code below
# hint 1 -- x_train[y_train==1,:] for example will extract only the elements␣
↪from x_train that correspond to class 1
# hint 2 -- np.mean(blah, axis=0) will take the mean of each row
# hint 3 -- reshape your vector so that it's a column vector
mu_0 = np.mean(x_train[y_train==0,:], axis=0).reshape((-1,1))
mu_1 = np.mean(x_train[y_train==1,:], axis=0).reshape((-1,1))
mu_2 = np.mean(x_train[y_train==2,:], axis=0).reshape((-1,1))

### compute covariance of each class
### np.cov() expects each column to be a datapoint
cov_0 = np.cov(x_train[y_train==0,:].T)
cov_1 = np.cov(x_train[y_train==1,:].T)
cov_2 = np.cov(x_train[y_train==2,:].T)
```

[3]:
```python
### complete the code below to compute the log-likelihood ratio under all three␣
↪classes
def log_likelihood(_x, _mu, _cov):
    ## _x and _mu should be column vectors, and _cov should be an n \times n␣
↪matrix
    assert np.shape(_x) == np.shape(_mu)
    _log_likelihood = -1*np.linalg.slogdet(_cov)[1] - (_x-_mu).T@np.linalg.
↪inv(_cov)@(_x-_mu)
    return _log_likelihood[0,0]
```

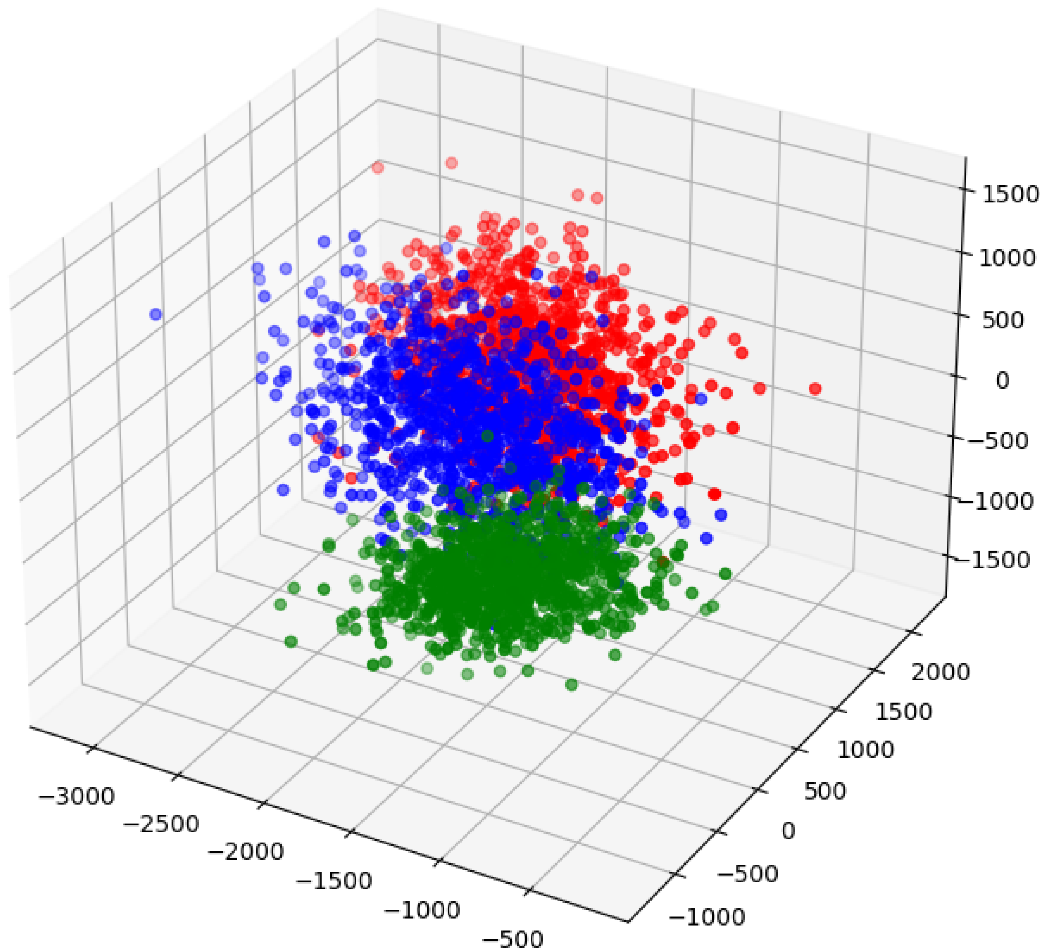
[4]:
```python
from sklearn.metrics import classification_report
### predict the class of the vectors in the test set
y_hat = []
for i, x in enumerate(x_test):
    x_column_vector = np.reshape(x,(-1,1))
    ll0 = log_likelihood(x_column_vector, mu_0, cov_0)
    ll1 = log_likelihood(x_column_vector, mu_1, cov_1)
    ll2 = log_likelihood(x_column_vector, mu_2, cov_2)
    y_hat.append(np.argmax([ll0, ll1, ll2]))

### compute the accuracy and print a classification report
print(classification_report(y_test, y_hat))
```

```
              precision    recall  f1-score   support

           0       0.98      0.92      0.95       341
           1       0.90      0.96      0.93       336
```

```
           2       0.97      0.95      0.96       341

    accuracy                           0.94      1018
   macro avg       0.95      0.95      0.95      1018
weighted avg       0.95      0.94      0.95      1018
```

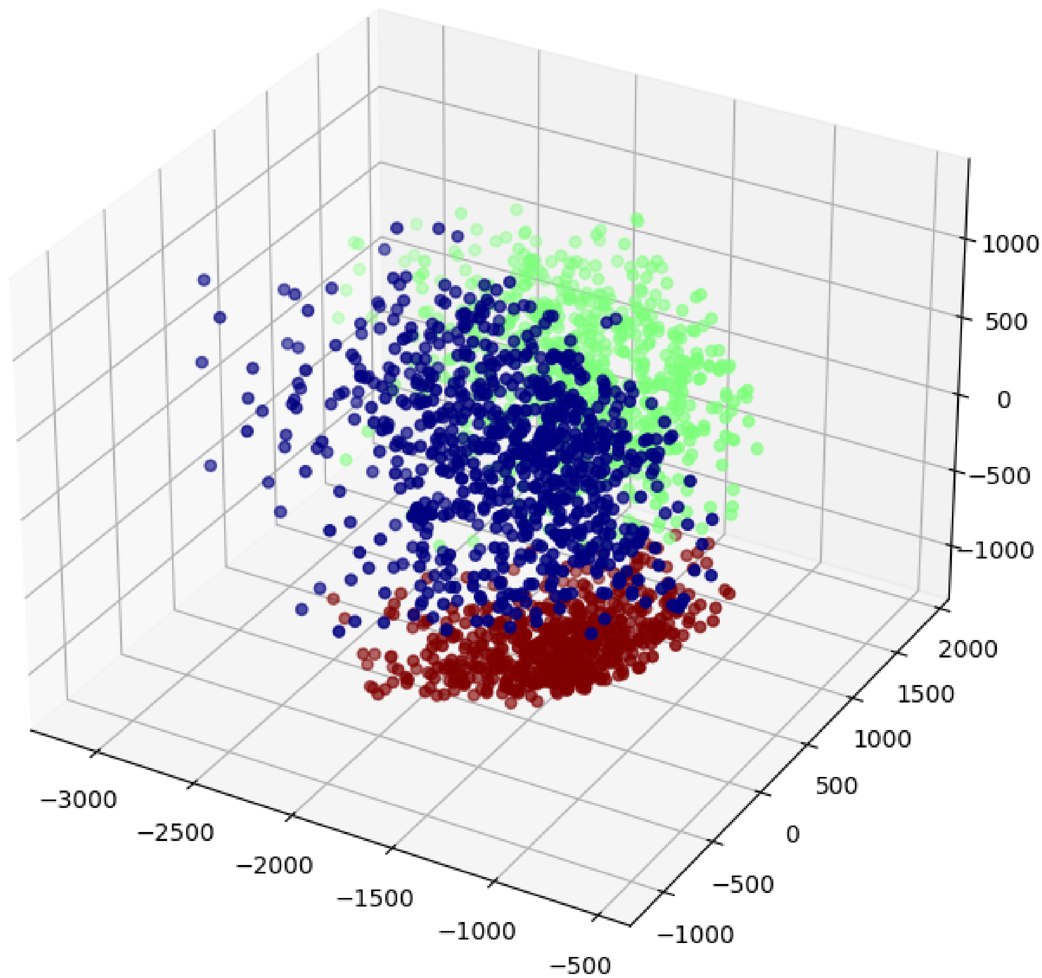
[5]:
```python
### create data points from three classes, and plot for comparison
x_0 = np.random.multivariate_normal(mu_0.squeeze(), cov_0, 1000)
x_1 = np.random.multivariate_normal(mu_1.squeeze(), cov_1, 1000)
x_2 = np.random.multivariate_normal(mu_2.squeeze(), cov_2, 1000)
print(np.shape(x_0))

# %matplotlib notebook #uncomment this line to make plot interactive
from mpl_toolkits.mplot3d import Axes3D

fig = plt.figure(figsize=(8, 8))
ax = fig.add_subplot(111, projection='3d')
ax.scatter(x_0[:,0], x_0[:,1], x_0[:,2], c='r')
ax.scatter(x_1[:,0], x_1[:,1], x_1[:,2], c='b')
ax.scatter(x_2[:,0], x_2[:,1], x_2[:,2], c='g')
plt.show()
```

```
(1000, 3)
```



[6]:
```
# 2a
from sklearn import mixture

# EM with sklearn
gm = mixture.GaussianMixture(n_components=3, random_state=69420).fit(x_train)
```

[7]:
```
#2b
fig = plt.figure(figsize=(8, 8))
ax = fig.add_subplot(111, projection='3d')
ax.scatter(x_train[:,0], x_train[:,1], x_train[:,2], c=gm.predict(x_train),␣
 ↪cmap='jet')
plt.show()
```



[8]:
```
#2c
#we can somewhat map means and covs around to one that's closest
#though they are out of order
#Here: label(0,1,2) = EM(1,0,2)

#get means and covariances
label_means = np.array([mu_0,mu_1,mu_2]).reshape(3,3)
label_covs = np.array([cov_0,cov_1,cov_2]).reshape(3, 3, 3)
gm_means = gm.means_
gm_covs = gm.covariances_

#compare means
print("Comparing means")
```

```
print(f"Means with labels:\n{label_means}")
print(f"Means with EM:\n{gm_means}")
print("\n")
#compare covs
print("Comparing covs")
print(f"Covs with labels:\n{label_covs}")
print(f"Covs with EM:\n{gm_covs}")
```

```
Comparing means
Means with labels:
[[-1713.81050084   944.31365051   281.16236007]
 [-1552.93107583   -89.19746319   477.87368097]
 [-1295.33434171  -205.02751531  -567.1897333 ]]
Means with EM:
[[-1537.70380674  -136.98245222   415.61331786]
 [-1699.53722651   966.80706526   266.41225178]
 [-1288.02906558  -172.93222688  -653.21398282]]



Comparing covs
Covs with labels:
[[[ 2.00183376e+05 -6.52227338e+04  1.36975829e+02]
  [-6.52227338e+04  1.37326374e+05  4.47994782e+04]
  [ 1.36975829e+02  4.47994782e+04  6.68313191e+04]]

 [[ 1.89841368e+05 -1.26359508e+04 -3.80146953e+04]
  [-1.26359508e+04  7.13122521e+04  3.46773546e+04]
  [-3.80146953e+04  3.46773546e+04  1.07049819e+05]]

 [[ 1.02911655e+05  8.78912053e+03  3.58852851e+04]
  [ 8.78912053e+03  1.39937256e+05 -4.35622433e+04]
  [ 3.58852851e+04 -4.35622433e+04  9.88917838e+04]]]
Covs with EM:
[[[191661.84327282 -23193.09527672 -38262.0234912 ]
  [-23193.09527672  97329.68665824  53356.24273249]
  [-38262.0234912   53356.24273249 123501.60250505]]

 [[204054.24552437 -72199.13588021  -4986.22273393]
  [-72199.13588021 112962.04708696  45979.65587803]
  [ -4986.22273393  45979.65587803  78381.36817658]]

 [[ 92901.27632136   7385.20253224  40801.77737756]
  [  7385.20253224 111596.41620463 -20832.82496708]
  [ 40801.77737756 -20832.82496708  54790.25840067]]]
```

[9]:

```
#2d
predictions = gm.predict(x_test)
```

```python
#find permutation with the least error
import itertools
permutations = list(itertools.permutations([0,1,2]))

best_permutation = permutations.pop(0)
permute = np.vectorize(lambda i: best_permutation[i])
best_accuracy = np.sum(permute(predictions)==y_test)/len(y_test)

while len(permutations)>0:
    permutation = permutations.pop(0)
    permute = np.vectorize(lambda i: permutation[i])
    accuracy = np.sum(permute(predictions)==y_test)/len(y_test)
    if accuracy>=best_accuracy:
        best_permutation = permutation
        best_accuracy = accuracy

print(f"Best accuracy = {best_accuracy}")
print(f"Best error rate = {1-best_accuracy}")

from sklearn.metrics import classification_report
permute = np.vectorize(lambda i: best_permutation[i])
print(classification_report(y_test, permute(predictions)))
```

```
Best accuracy = 0.9145383104125737
Best error rate = 0.08546168958742628
              precision    recall  f1-score   support

           0       0.97      0.91      0.94       341
           1       0.82      0.98      0.89       336
           2       0.98      0.86      0.92       341

    accuracy                           0.91      1018
   macro avg       0.92      0.91      0.92      1018
weighted avg       0.92      0.91      0.92      1018
```

[10]:
```python
#2e
#kmeans performs a bit better than EM
from sklearn import cluster
kmeans = cluster.KMeans(n_clusters=3, n_init='auto').fit(x_train)
predictions = kmeans.predict(x_test)

#find permutation with the least error
permutations = list(itertools.permutations([0,1,2]))
best_permutation = permutations.pop(0)
permute = np.vectorize(lambda i: best_permutation[i])
```

```python
best_accuracy = np.sum(permute(predictions)==y_test)/len(y_test)

while len(permutations)>0:
    permutation = permutations.pop(0)
    permute = np.vectorize(lambda i: permutation[i])
    accuracy = np.sum(permute(predictions)==y_test)/len(y_test)
    if accuracy>=best_accuracy:
        best_permutation = permutation
        best_accuracy = accuracy

print(f"Best accuracy = {best_accuracy}")
print(f"Best error rate = {1-best_accuracy}")

from sklearn.metrics import classification_report
permute = np.vectorize(lambda i: best_permutation[i])
print(classification_report(y_test, permute(predictions)))
```

```
Best accuracy = 0.9341846758349706
Best error rate = 0.06581532416502944
              precision    recall  f1-score   support

           0       0.98      0.89      0.94       341
           1       0.88      0.94      0.91       336
           2       0.95      0.97      0.96       341

    accuracy                           0.93      1018
   macro avg       0.94      0.93      0.93      1018
weighted avg       0.94      0.93      0.93      1018
```

2.f) We could use dimensionality reduction, preferably those that preserve distance like t-SNE, to visualize the distributions of the clusters in 2-3 dimensions and manually determine the number of clusters. Moreover, unsupervised clustering algorithms like DBSCAN learns to determine the number of clusters based on a set minimum distance one would consider two points to be from different clusters.

3 Problem 3

	$p(x,y)$	fish	cat	dog
		y		
	1	$\frac{1}{4}$	$\frac{1}{8}$	$\frac{1}{16}$
x	2	$\frac{1}{16}$	0	$\frac{1}{4}$
	3	0	$\frac{1}{8}$	$\frac{1}{16}$
	4	$\frac{1}{16}$	0	0

3.a) Recall Fano's inequality

$$\mathbb{P}(\hat{y} \neq y) \geq \frac{H(y|x) - 1}{\log(|\mathcal{Y}|)}$$

we can compute the conditional entropy as such

$$H(y|x) = \sum_{x,y} -p(x,y) \log_2 \frac{p(x,y)}{p(x)}$$

plugging in the numbers from the above, $H(y|x) \approxeq 1.001$, which means the lower bound is approximately $\frac{1.001-1}{12}$ (an extremely small magnitude).

3.b) We specify an MAP classifier under the condition $p(y|x)$ is maximized for each x:

- $x = 1$, the most likely y is "fish' '
- $x = 2$, the most likely y is "dog' '
- $x = 3$, the most likely y is "cat' '
- $x = 4$, the most likely y is "fish' '

the total error rate can be computed as follows: $\frac{1}{8} + \frac{1}{16} + \frac{1}{16} + \frac{1}{16} = \frac{5}{16}$ (way higher than the lower bound)

3.c) Instead of bounding $H(E|\hat{Y})$ by 1, we use the fact that $H(E|\hat{Y} \leq H(E))$, and the fact that

$$H(E) = (1 - p_e) \log_2 \frac{1}{1 - p_e} + p_e \log_2 \frac{1}{p_e}$$

Moreover,

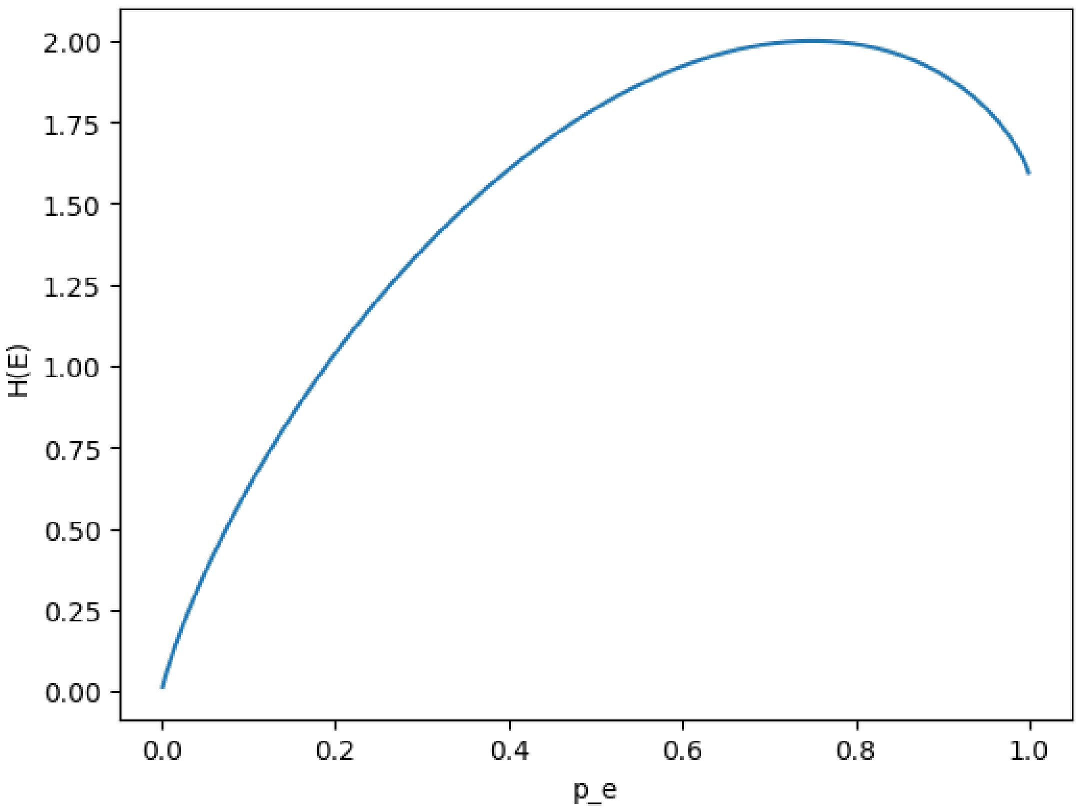
$$H(Y|E, \hat{Y}) = \mathbb{P}(Y \neq \hat{Y}) H(Y|E = 1, \hat{Y}) = p_e \log_2(\mathcal{Y}) = p_e \log_2(3)$$

and Fano's inequality sugggests

$$H(E|Y, \hat{Y}) = H(E|\hat{Y}) + H(Y|E, \hat{Y})$$
$$H(y|x) = (1 - p_e) \log_2 \frac{1}{1 - p_e} + p_e \log_2 \frac{1}{p_e} + p_e \log_2(3)$$

we can express the values of $H(E)$ for each p_e as follows

[8]:
```python
import numpy as np
def entropy(pe):
    return (1-pe)*np.log2(1/(1-pe))+pe*np.log2(1/pe)+pe*np.log2(3)
import matplotlib.pyplot as plt
pe = np.linspace(0,1,1000)[1:-1]
h = entropy(pe)
plt.plot(pe,h)
plt.xlabel("p_e")
plt.ylabel("H(E)")
plt.show()
```



[9]:
```python
print(f"Tightest bound is at pe = {round(pe[np.argmax(h)],4)}")
```

```
Tightest bound is at pe = 0.7497
```

[ ]:
```python

```