Homework 5

In [180…

```
#imports
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
```

In [181…

```
wage_raw = pd.read_csv('Wage.csv')
```

1. K-Means

1. Instead of just having an indicator for job class we add an indicator for each of the categorical variables. The categorical variables are:

- sex
- maritl
- race
- education
- region
- jobclass
- health
- health_ins

In [214…

```
cat_var = ['sex', 'maritl', 'race', 'education', 'region', 'jobclass', 'health', 'h
wage = pd.get_dummies(wage_raw, columns=cat_var, drop_first=True)
```

In [102…

```
wage
```

Out[102…

	year	age	logwage	wage	maritl_2. Married	maritl_3. Widowed	maritl_4. Divorced	maritl_5. Separated	race_2. Black
0	2006	18	4.318063	75.043154	False	False	False	False	False
1	2004	24	4.255273	70.476020	False	False	False	False	False
2	2003	45	4.875061	130.982177	True	False	False	False	False
3	2003	43	5.041393	154.685293	True	False	False	False	False
4	2005	50	4.318063	75.043154	False	False	True	False	False
...	...	...	...	...	...	...	...	...	...
2995	2008	44	5.041393	154.685293	True	False	False	False	False
2996	2007	30	4.602060	99.689464	True	False	False	False	False
2997	2005	27	4.193125	66.229408	True	False	False	False	True
2998	2005	27	4.477121	87.981033	False	False	False	False	False
2999	2009	55	4.505150	90.481913	False	False	False	True	False

3000 rows × 18 columns

2. Code

In [215…

```
rng = np.random.default_rng()
train_idx = rng.binomial(1, 0.8, size = wage.shape[0]).astype(bool)
wage_train = wage[train_idx]
wage_test = wage[~train_idx]
```

3. We assume that everyone has completed middle school for a minimum of 9 years of schooling. If someone is a High School Grad that counts for 4 more years. If they have some college, we assume that's an associate degree. The cases where someone drops out at 1 year or 3 years average out to 2 years (most of the time). Some college should count for 2 years. College grad counts for 4 years. Advanced degreees can be masters or phds or professional degrees. Masters usually are 2 years, and Phds are usually 5-6, but often take much longer. We assume it adds a bit more years of schooling, say 4 years on average.

4. Code

In [104…

```
# make education years column
education_years = 9 + wage['education_2. HS Grad'] * 4 + wage['education_3. Some Co

# make kmeans data_frame
```

```
wage_kmeans = pd.DataFrame({'jobclass': wage['jobclass_2. Information'], 'age': wag

# standardize
wage_kmeans[['age', 'education_years', 'logwage']] = (wage_kmeans[['age', 'educatio

# split
wage_kmeans_train = wage_kmeans[train_idx]
wage_kmeans_test = wage_kmeans[~train_idx]
wage_kmeans
```

Out[104…

	jobclass	age	education_years	logwage
0	False	-2.115215	-1.888180	-0.954767
1	True	-1.595392	0.577880	-1.133275
2	False	0.223986	-0.038635	0.628727
3	True	0.050712	0.577880	1.101591
4	True	0.657171	-0.655150	-0.954767
...	...	...	...	...
2995	False	0.137349	-0.038635	1.101591
2996	False	-1.075570	-0.655150	-0.147391
2997	False	-1.335481	-1.888180	-1.309956
2998	False	-1.335481	-0.038635	-0.502580
2999	False	1.090356	-0.655150	-0.422897

3000 rows × 4 columns

5. Code

In [105…

```
import sklearn.cluster
num_clusters = 8
kmeans_clf = sklearn.cluster.KMeans(n_clusters = num_clusters)
kmeans_clf.fit(wage_kmeans_train)
```

Out[105…

```
KMeans
KMeans()
```

6. Predict

In [106…

```
kmeans_weights = np.zeros(num_clusters)
for i in range(num_clusters):
    kmeans_weights[i] = np.mean(wage_kmeans_train.iloc[kmeans_clf.labels_ == i]['jo
print(kmeans_weights)
```

```
[0.66666667 0.50743494 0.8        0.29299363 0.46285714 0.39732143
 0.51923077 0.24096386]
```

7. Here is some code to investigate this:

In [107…
```
print(kmeans_clf.cluster_centers_)
```

```
[[ 0.66812227  1.25038882  0.95478894  0.2731773 ]
 [ 0.50743494 -0.77362873 -0.03634307 -0.0313303 ]
 [ 0.8        -0.09379251  1.81091033  1.06612075]
 [ 0.29299363  0.09320234 -1.88818023 -0.5036751 ]
 [ 0.46285714  0.37102111 -0.43672755 -1.42916273]
 [ 0.39732143  0.92133658 -0.49138822 -0.02732775]
 [ 0.51910828  0.08602857  0.29514711  1.09067936]
 [ 0.24096386 -1.44717001 -0.62543847 -1.19975317]]
```

As we can see, the first coordinate of each of the cluster centers (the part corresponding to the job) is exactly the fraction we calculated prior. This makes sense since each cluster center's should just be the mean of the group, otherwise the variance would increase.

8. Code

In [108…
```
kmeans_weights = kmeans_weights > 0.5
kmeans_preds_train = np.eye(num_clusters)[kmeans_clf.predict(wage_kmeans_train)] @
kmeans_err_train = np.mean(kmeans_preds_train != wage_kmeans_train['jobclass'])
print(f"The training error for the k-means classifier is {kmeans_err_train}")
```

```
The training error for the k-means classifier is 0.3831578947368421
```

9. Code

In [109…
```
kmeans_preds_test = np.eye(num_clusters)[kmeans_clf.predict(wage_kmeans_test)] @ km
```

10. Code

In [110…
```
kmeans_err_test = np.mean(kmeans_preds_test != wage_kmeans_test['jobclass'])
print(f"The test error for the k-means classifier is {kmeans_err_test}")
```

```
The test error for the k-means classifier is 0.384
```

2. Neural Networks

1. Code

In [111…

```
import torch
import torch.nn as nn
import torch.optim as optim
import torch.functional as F
import torch.utils.data as data
```

In [112…

```
device = torch.accelerator.current_accelerator().type if torch.accelerator.is_avail
print(f'Using device: {device}')
```

```
Using device: cuda
```

In [113…

```
class Net(nn.Module):
    def __init__(self):
        super(Net, self).__init__()
        self.linear_relu_logit_stack = nn.Sequential(
            nn.Linear(3, 16),
            nn.ReLU(),
            nn.Linear(16, 16),
            nn.ReLU(),
            nn.Linear(16, 1),
            nn.Sigmoid()
        )
    def forward(self, x):
        return self.linear_relu_logit_stack(x)
```

In [114…

```
model = Net().to(device)
print(model)
```

```
Net(
  (linear_relu_logit_stack): Sequential(
    (0): Linear(in_features=3, out_features=16, bias=True)
    (1): ReLU()
    (2): Linear(in_features=16, out_features=16, bias=True)
    (3): ReLU()
    (4): Linear(in_features=16, out_features=1, bias=True)
    (5): Sigmoid()
  )
)
```

In [115…

```
loss_fn = nn.BCELoss()
optimizer = optim.Adam(model.parameters(), lr=10e-4)
```

In [116…

```
wage_nn_train = torch.tensor(wage_kmeans_train[['age', 'education_years', 'logwage'
jobclass_nn_train = torch.tensor(wage_kmeans_train['jobclass'].values, dtype=torch.
jobclass_nn_train = jobclass_nn_train.view(-1, 1)
wage_nn_test = torch.tensor(wage_kmeans_test[['age', 'education_years', 'logwage']]
jobclass_nn_test = torch.tensor(wage_kmeans_test['jobclass'].values, dtype=torch.fl
jobclass_nn_test = jobclass_nn_test.view(-1, 1)
```

In [150…

```
def train(X,y, model, loss_fn, optimizer):
    model.train()
    for i in np.random.permutation(range(len(X))):
        Xi, yi = X[i], y[i]
        Xi, yi = Xi.to(device), yi.to(device)
        # pred
```

```python
        pred = model(Xi)
        loss = loss_fn(pred, yi)
        # backprop
        loss.backward()
        optimizer.step()
        optimizer.zero_grad()

        # if i % 100 == 0:
        #     loss = loss.item()
        #     print(f"loss: {loss:>7f} [{i:>5d}/{len(X)}]")
```

In [119…

```python
def test(X, y, model, loss_fn):
    model.eval()
    size = len(X)
    test_loss, correct = 0, 0
    with torch.no_grad():
        for Xi, yi in zip(X, y):
            Xi, yi = Xi.to(device), yi.to(device)
            pred = model(Xi)
            loss = loss_fn(pred, yi)
            test_loss += loss.item()
            correct += (pred.round() == yi).type(torch.float).item()
    test_loss /= size
    correct /= size
    print(f"Test Error: \n Accuracy: {(100*correct):>0.1f}%, Avg loss: {test_loss:>
```

In [120…

```python
epochs = 5
for t in range(epochs):
    print(f"Epoch {t+1}\n-------------------------------")
    train(wage_nn_train, jobclass_nn_train, model, loss_fn, optimizer)
    test(wage_nn_train, jobclass_nn_train, model, loss_fn)
```

```
Epoch 1
-------------------------------
loss: 0.812750 [    0/2375]
loss: 0.746969 [  100/2375]
loss: 0.719210 [  200/2375]
loss: 0.767752 [  300/2375]
loss: 0.660637 [  400/2375]
loss: 0.588416 [  500/2375]
loss: 0.458372 [  600/2375]
loss: 0.949831 [  700/2375]
loss: 0.394190 [  800/2375]
loss: 0.539203 [  900/2375]
loss: 0.386170 [ 1000/2375]
loss: 0.609884 [ 1100/2375]
loss: 0.383120 [ 1200/2375]
loss: 0.563736 [ 1300/2375]
loss: 0.398944 [ 1400/2375]
loss: 0.876582 [ 1500/2375]
loss: 0.573143 [ 1600/2375]
loss: 0.538475 [ 1700/2375]
loss: 0.820063 [ 1800/2375]
loss: 0.761384 [ 1900/2375]
loss: 0.695942 [ 2000/2375]
loss: 0.564975 [ 2100/2375]
loss: 0.765927 [ 2200/2375]
loss: 0.652676 [ 2300/2375]
Test Error: 
 Accuracy: 63.4%, Avg loss: 0.641555

Epoch 2
-------------------------------
loss: 0.210156 [    0/2375]
loss: 1.319807 [  100/2375]
loss: 0.639556 [  200/2375]
loss: 0.883472 [  300/2375]
loss: 0.550339 [  400/2375]
loss: 0.645365 [  500/2375]
loss: 0.297408 [  600/2375]
loss: 0.978406 [  700/2375]
loss: 0.312536 [  800/2375]
loss: 0.493559 [  900/2375]
loss: 0.265217 [ 1000/2375]
loss: 0.656701 [ 1100/2375]
loss: 0.341239 [ 1200/2375]
loss: 0.505777 [ 1300/2375]
loss: 0.353160 [ 1400/2375]
loss: 0.948211 [ 1500/2375]
loss: 0.579311 [ 1600/2375]
loss: 0.542754 [ 1700/2375]
loss: 0.892295 [ 1800/2375]
loss: 0.833782 [ 1900/2375]
loss: 0.703336 [ 2000/2375]
loss: 0.573368 [ 2100/2375]
loss: 0.643396 [ 2200/2375]
loss: 0.668942 [ 2300/2375]
Test Error: 
```

```
 Accuracy: 63.3%, Avg loss: 0.639839

Epoch 3
-------------------------------
loss: 0.197121 [    0/2375]
loss: 1.361679 [  100/2375]
loss: 0.602423 [  200/2375]
loss: 0.872241 [  300/2375]
loss: 0.527855 [  400/2375]
loss: 0.687093 [  500/2375]
loss: 0.291176 [  600/2375]
loss: 0.968406 [  700/2375]
loss: 0.314803 [  800/2375]
loss: 0.478695 [  900/2375]
loss: 0.256262 [ 1000/2375]
loss: 0.670204 [ 1100/2375]
loss: 0.338398 [ 1200/2375]
loss: 0.477620 [ 1300/2375]
loss: 0.352963 [ 1400/2375]
loss: 0.971770 [ 1500/2375]
loss: 0.587948 [ 1600/2375]
loss: 0.553294 [ 1700/2375]
loss: 0.931062 [ 1800/2375]
loss: 0.866287 [ 1900/2375]
loss: 0.728667 [ 2000/2375]
loss: 0.576758 [ 2100/2375]
loss: 0.640255 [ 2200/2375]
loss: 0.661310 [ 2300/2375]
Test Error:
 Accuracy: 62.8%, Avg loss: 0.638866

Epoch 4
-------------------------------
loss: 0.195810 [    0/2375]
loss: 1.353064 [  100/2375]
loss: 0.576642 [  200/2375]
loss: 0.875407 [  300/2375]
loss: 0.515771 [  400/2375]
loss: 0.709095 [  500/2375]
loss: 0.287706 [  600/2375]
loss: 0.964781 [  700/2375]
loss: 0.322479 [  800/2375]
loss: 0.477383 [  900/2375]
loss: 0.250386 [ 1000/2375]
loss: 0.679712 [ 1100/2375]
loss: 0.337413 [ 1200/2375]
loss: 0.459359 [ 1300/2375]
loss: 0.352589 [ 1400/2375]
loss: 0.995823 [ 1500/2375]
loss: 0.601121 [ 1600/2375]
loss: 0.577461 [ 1700/2375]
loss: 0.958010 [ 1800/2375]
loss: 0.889349 [ 1900/2375]
loss: 0.739539 [ 2000/2375]
loss: 0.584371 [ 2100/2375]
loss: 0.627556 [ 2200/2375]
```

```
loss: 0.667797 [ 2300/2375]
Test Error:
 Accuracy: 63.1%, Avg loss: 0.638044

Epoch 5
-------------------------------
loss: 0.195758 [    0/2375]
loss: 1.338283 [  100/2375]
loss: 0.564895 [  200/2375]
loss: 0.876744 [  300/2375]
loss: 0.506485 [  400/2375]
loss: 0.718385 [  500/2375]
loss: 0.284253 [  600/2375]
loss: 0.968909 [  700/2375]
loss: 0.323961 [  800/2375]
loss: 0.477185 [  900/2375]
loss: 0.249726 [ 1000/2375]
loss: 0.676968 [ 1100/2375]
loss: 0.332026 [ 1200/2375]
loss: 0.453932 [ 1300/2375]
loss: 0.347524 [ 1400/2375]
loss: 1.008308 [ 1500/2375]
loss: 0.610335 [ 1600/2375]
loss: 0.592940 [ 1700/2375]
loss: 0.972017 [ 1800/2375]
loss: 0.903162 [ 1900/2375]
loss: 0.749905 [ 2000/2375]
loss: 0.587641 [ 2100/2375]
loss: 0.618064 [ 2200/2375]
loss: 0.675766 [ 2300/2375]
Test Error:
 Accuracy: 63.0%, Avg loss: 0.637513
```

2. Code

In [121…

```
model.eval()
wage_nn_train_err = ((model(wage_nn_train) > 0.5) == jobclass_nn_train).sum().item(
print(f"The training error for the neural network is {1 - wage_nn_train_err}")
```

```
The training error for the neural network is 0.3701052631578947
```

3. Code

In [122…

```
model.eval()
wage_nn_test_err = ((model(wage_nn_test) > 0.5) == jobclass_nn_test).sum().item()/l
print(f"The test error for the neural network is {1 - wage_nn_test_err}")
```

```
The test error for the neural network is 0.35840000000000005
```

3. Regression Trees

1. Code

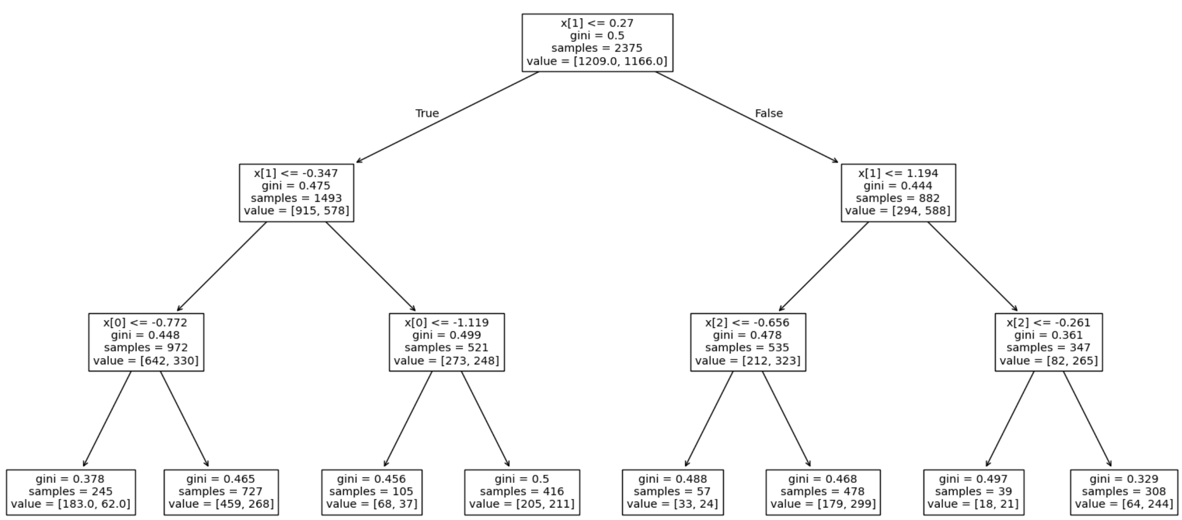
In [123…
```
wage_dt_train = wage_kmeans_train[['age', 'education_years', 'logwage']]
jobclass_dt_train = wage_kmeans_train['jobclass']
wage_dt_test = wage_kmeans_test[['age', 'education_years', 'logwage']]
jobclass_dt_test = wage_kmeans_test['jobclass']
```

In [124…
```
import sklearn.tree
dt_clf = sklearn.tree.DecisionTreeClassifier(max_depth = 3)
dt_clf.fit(wage_dt_train, jobclass_dt_train)
```

Out[124…
```
DecisionTreeClassifier
DecisionTreeClassifier(max_depth=3)
```

2. Code

In [125…
```
plt.figure(figsize=(20,10))
sklearn.tree.plot_tree(dt_clf)
plt.show()
```



3. Code

In [126…
```
wage_dt_train_err = 1 - dt_clf.score(wage_dt_train, jobclass_dt_train)
print(f"The training error for the decision tree is {wage_dt_train_err}")
```

```
The training error for the decision tree is 0.36084210526315785
```

4. Code

In [127…

```
wage_dt_test_err = 1 - dt_clf.score(wage_dt_test, jobclass_dt_test)
print(f"The test error for the decision tree is {wage_dt_test_err}")
```

```
The test error for the decision tree is 0.36639999999999995
```

5. Code

In [128…

```
wage_dt = wage_kmeans[['age', 'education_years', 'logwage']]
jobclass_dt = wage_kmeans['jobclass']
dt_full_clf = sklearn.tree.DecisionTreeClassifier(max_depth = 3)
dt_full_clf.fit(wage_dt, jobclass_dt)
```

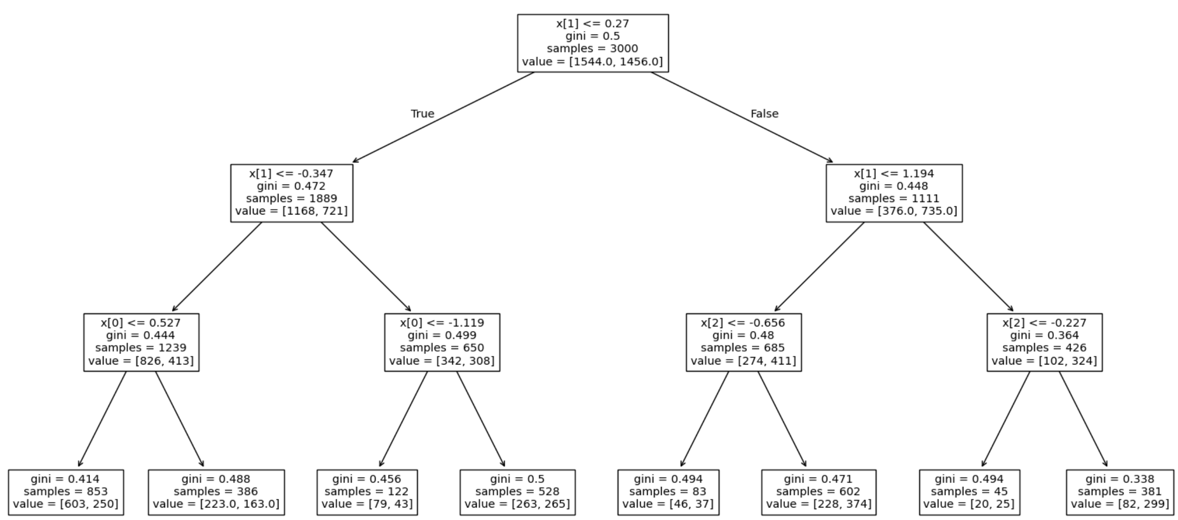
Out[128…

```
DecisionTreeClassifier
DecisionTreeClassifier(max_depth=3)
```

6. Code

In [129…

```
plt.figure(figsize=(20,10))
sklearn.tree.plot_tree(dt_full_clf)
plt.show()
```



7. Code

In [130…

```
wage_dt_full_train_err = 1 - dt_full_clf.score(wage_dt, jobclass_dt)
print(f"The error on the training set for the decision tree is {wage_dt_full_train_
```

```
The error on the training set for the decision tree is 0.362
```

8. Code

In [131…

```
wage_dt_full_test_err = 1 - dt_full_clf.score(wage_dt_test, jobclass_dt_test)
print(f"The error on the test set for the decision tree is {wage_dt_full_test_err}"
```

```
The error on the test set for the decision tree is 0.36639999999999995
```

4. Compare All Methods

1. Code

In [133…

```
results = pd.DataFrame({
    'kmeans': [kmeans_err_train, kmeans_err_test],
    'neural_network': [1 - wage_nn_train_err, 1 - wage_nn_test_err],
    'decision_tree': [wage_dt_train_err, wage_dt_test_err],
    'decision_tree_full': [wage_dt_full_train_err, wage_dt_full_test_err]
}, index = ['train', 'test'])
results
```

Out[133…

	kmeans	neural_network	decision_tree	decision_tree_full
train	0.383158	0.370105	0.360842	0.3620
test	0.384000	0.358400	0.366400	0.3664

2. All of the methods performed incredibly close to each other. They were all within 3e-2 of each other. The neural network did the best, which was expected, but all methods performed well. In theory, we should have expected the full decision tree to perform better than the standard decision tree, however, we only had 3 features, with max depth of 8, so the decision tree was quite constrained. The neural network performed the best, but not by as well as I expected. I chalk this up to there not being enough data and not using enough features.

3. (Optional) We are going to train a neural net with one more hidden layer and more intermediary nodes. We are also going to cross validate for the learning rate in the SGD optimizer and we are going to use the entire data set with the one-hot encoding.

In [ ]:

```
wage = wage.astype('float64')
wage_train = wage_train.astype('float64')
wage_test = wage_test.astype('float64')
wage
```

In [137…

```
class Net(nn.Module):
    def __init__(self):
        super(Net, self).__init__()
        self.linear_relu_logit_stack = nn.Sequential(
            nn.Linear(17, 32),
            nn.ReLU(),
            nn.Linear(32, 32),
```

```
            nn.ReLU(),
            nn.Linear(32, 32),
            nn.ReLU(),
            nn.Linear(32, 1),
            nn.Sigmoid()
        )
    def forward(self, x):
        return self.linear_relu_logit_stack(x)
model = Net().to(device)
print(model)
```

```
Net(
  (linear_relu_logit_stack): Sequential(
    (0): Linear(in_features=17, out_features=32, bias=True)
    (1): ReLU()
    (2): Linear(in_features=32, out_features=32, bias=True)
    (3): ReLU()
    (4): Linear(in_features=32, out_features=32, bias=True)
    (5): ReLU()
    (6): Linear(in_features=32, out_features=1, bias=True)
    (7): Sigmoid()
  )
)
```

In [145…

```
wage_nn_train = torch.tensor(wage_train.drop(columns = 'jobclass_2. Information').v
jobclass_nn_train = torch.tensor(wage_train['jobclass_2. Information'].values, dtyp
jobclass_nn_train = jobclass_nn_train.view(-1, 1)
wage_nn_test = torch.tensor(wage_test.drop(columns = 'jobclass_2. Information').val
jobclass_nn_test = torch.tensor(wage_test['jobclass_2. Information'].values, dtype=
jobclass_nn_test = jobclass_nn_test.view(-1, 1)
```

In [151…

```
optimizer = optim.Adam(model.parameters(), lr=10e-5)
epochs = 25
```

In [152…

```
for t in range(epochs):
    print(f"Epoch {t+1}\n-------------------------------")
    train(wage_nn_train, jobclass_nn_train, model, loss_fn, optimizer)
    test(wage_nn_train, jobclass_nn_train, model, loss_fn)
```

```
Epoch 1
-------------------------------
Test Error:
 Accuracy: 57.2%, Avg loss: 0.677338

Epoch 2
-------------------------------
Test Error:
 Accuracy: 57.1%, Avg loss: 0.673603

Epoch 3
-------------------------------
Test Error:
 Accuracy: 59.4%, Avg loss: 0.667718

Epoch 4
-------------------------------
Test Error:
 Accuracy: 57.2%, Avg loss: 0.672373

Epoch 5
-------------------------------
Test Error:
 Accuracy: 59.5%, Avg loss: 0.668015

Epoch 6
-------------------------------
Test Error:
 Accuracy: 56.8%, Avg loss: 0.676829

Epoch 7
-------------------------------
Test Error:
 Accuracy: 59.7%, Avg loss: 0.668202

Epoch 8
-------------------------------
Test Error:
 Accuracy: 57.7%, Avg loss: 0.668597

Epoch 9
-------------------------------
Test Error:
 Accuracy: 57.5%, Avg loss: 0.670710

Epoch 10
-------------------------------
Test Error:
 Accuracy: 59.6%, Avg loss: 0.667262

Epoch 11
-------------------------------
Test Error:
 Accuracy: 57.4%, Avg loss: 0.673629

Epoch 12
```

```
-------------------------------
Test Error: 
 Accuracy: 59.2%, Avg loss: 0.668041 

Epoch 13
-------------------------------
Test Error: 
 Accuracy: 59.7%, Avg loss: 0.671024 

Epoch 14
-------------------------------
Test Error: 
 Accuracy: 59.5%, Avg loss: 0.666076 

Epoch 15
-------------------------------
Test Error: 
 Accuracy: 58.1%, Avg loss: 0.668596 

Epoch 16
-------------------------------
Test Error: 
 Accuracy: 54.1%, Avg loss: 0.718041 

Epoch 17
-------------------------------
Test Error: 
 Accuracy: 58.5%, Avg loss: 0.667357 

Epoch 18
-------------------------------
Test Error: 
 Accuracy: 56.8%, Avg loss: 0.675292 

Epoch 19
-------------------------------
Test Error: 
 Accuracy: 60.0%, Avg loss: 0.665352 

Epoch 20
-------------------------------
Test Error: 
 Accuracy: 58.4%, Avg loss: 0.670422 

Epoch 21
-------------------------------
Test Error: 
 Accuracy: 56.4%, Avg loss: 0.676622 

Epoch 22
-------------------------------
Test Error: 
 Accuracy: 60.3%, Avg loss: 0.664770 

Epoch 23
-------------------------------
```

```
Test Error:
 Accuracy: 59.6%, Avg loss: 0.664290

Epoch 24
-------------------------------
Test Error:
 Accuracy: 59.4%, Avg loss: 0.664646

Epoch 25
-------------------------------
Test Error:
 Accuracy: 56.8%, Avg loss: 0.675412
```

In [153…

```python
model.eval()
wage_nn_train_err = ((model(wage_nn_train) > 0.5) == jobclass_nn_train).sum().item(
print(f"The training error for the neural network is {1 - wage_nn_train_err}")
wage_nn_test_err = ((model(wage_nn_test) > 0.5) == jobclass_nn_test).sum().item()/l
print(f"The test error for the neural network is {1 - wage_nn_test_err}")
```

```
The training error for the neural network is 0.43200000000000005
The test error for the neural network is 0.40480000000000005
```

It's clear that our nn is suffering. We will drop wages because they are codependent with logwages. We also demean year and age, since only the relative age and years matter

In [216…

```python
# education_years = 9 + wage['education_2. HS Grad'] * 4 + wage['education_3. Some
# wage.drop(columns=['education_2. HS Grad', 'education_3. Some College', 'educatio
# wage['education_years'] = education_years
wage = wage.astype('float64')
wage = wage.drop(columns = 'wage')
wage['year'] = wage['year'] - wage['year'].min()
# wage['age'] = wage['age'] - wage['age'].min()
wage_train = wage[train_idx]
wage_test = wage[~train_idx]
wage
```

Out[216...

	year	age	logwage	maritl_2. Married	maritl_3. Widowed	maritl_4. Divorced	maritl_5. Separated	race_2. Black	race_3. Asian	rac O
0	3.0	18.0	4.318063	0.0	0.0	0.0	0.0	0.0	0.0	
1	1.0	24.0	4.255273	0.0	0.0	0.0	0.0	0.0	0.0	
2	0.0	45.0	4.875061	1.0	0.0	0.0	0.0	0.0	0.0	
3	0.0	43.0	5.041393	1.0	0.0	0.0	0.0	0.0	1.0	
4	2.0	50.0	4.318063	0.0	0.0	1.0	0.0	0.0	0.0	
...	...	...	...	...	...	...	...	...	...	
2995	5.0	44.0	5.041393	1.0	0.0	0.0	0.0	0.0	0.0	
2996	4.0	30.0	4.602060	1.0	0.0	0.0	0.0	0.0	0.0	
2997	2.0	27.0	4.193125	1.0	0.0	0.0	0.0	1.0	0.0	
2998	2.0	27.0	4.477121	0.0	0.0	0.0	0.0	0.0	0.0	
2999	6.0	55.0	4.505150	0.0	0.0	0.0	1.0	0.0	0.0	

3000 rows × 17 columns

In [217...

```
wage_nn_train = torch.tensor(wage_train.drop(columns = 'jobclass_2. Information').v
jobclass_nn_train = torch.tensor(wage_train['jobclass_2. Information'].values, dtyp
jobclass_nn_train = jobclass_nn_train.view(-1, 1)
wage_nn_test = torch.tensor(wage_test.drop(columns = 'jobclass_2. Information').val
jobclass_nn_test = torch.tensor(wage_test['jobclass_2. Information'].values, dtype=
jobclass_nn_test = jobclass_nn_test.view(-1, 1)
```

In [218...

```
wage_nn_train.shape
```

Out[218...

```
torch.Size([2396, 16])
```

In [219...

```
device = torch.accelerator.current_accelerator().type if torch.accelerator.is_avail
class Net(nn.Module):
    def __init__(self, d_in = 17):
        super(Net, self).__init__()
        self.linear_relu_logit_stack = nn.Sequential(
            nn.Linear(d_in, 32),
            nn.ReLU(),
            nn.Linear(32, 32),
            nn.ReLU(),
            nn.Linear(32, 1),
            nn.Sigmoid()
        )
    def forward(self, x):
        return self.linear_relu_logit_stack(x)
model = Net(d_in = 16).to(device)
print(model)
```

```
Net(
  (linear_relu_logit_stack): Sequential(
    (0): Linear(in_features=16, out_features=32, bias=True)
    (1): ReLU()
    (2): Linear(in_features=32, out_features=32, bias=True)
    (3): ReLU()
    (4): Linear(in_features=32, out_features=1, bias=True)
    (5): Sigmoid()
  )
)
```

In [220…

```
optimizer = optim.Adam(model.parameters(), lr=10e-4)
epochs = 25
```

In [221…

```
for t in range(epochs):
    print(f"Epoch {t+1}\n-------------------------------")
    train(wage_nn_train, jobclass_nn_train, model, loss_fn, optimizer)
    test(wage_nn_train, jobclass_nn_train, model, loss_fn)
```

```
Epoch 1
-------------------------------
Test Error:
 Accuracy: 61.7%, Avg loss: 0.670983

Epoch 2
-------------------------------
Test Error:
 Accuracy: 62.4%, Avg loss: 0.646089

Epoch 3
-------------------------------
Test Error:
 Accuracy: 62.4%, Avg loss: 0.643450

Epoch 4
-------------------------------
Test Error:
 Accuracy: 63.4%, Avg loss: 0.647053

Epoch 5
-------------------------------
Test Error:
 Accuracy: 63.9%, Avg loss: 0.640471

Epoch 6
-------------------------------
Test Error:
 Accuracy: 61.6%, Avg loss: 0.651038

Epoch 7
-------------------------------
Test Error:
 Accuracy: 63.8%, Avg loss: 0.638435

Epoch 8
-------------------------------
Test Error:
 Accuracy: 63.4%, Avg loss: 0.639214

Epoch 9
-------------------------------
Test Error:
 Accuracy: 63.6%, Avg loss: 0.638327

Epoch 10
-------------------------------
Test Error:
 Accuracy: 63.9%, Avg loss: 0.641005

Epoch 11
-------------------------------
Test Error:
 Accuracy: 63.8%, Avg loss: 0.636597

Epoch 12
```

```
-------------------------------
Test Error: 
 Accuracy: 63.3%, Avg loss: 0.643140 

Epoch 13
-------------------------------
Test Error: 
 Accuracy: 64.7%, Avg loss: 0.633007 

Epoch 14
-------------------------------
Test Error: 
 Accuracy: 63.2%, Avg loss: 0.638376 

Epoch 15
-------------------------------
Test Error: 
 Accuracy: 61.6%, Avg loss: 0.661419 

Epoch 16
-------------------------------
Test Error: 
 Accuracy: 63.9%, Avg loss: 0.633139 

Epoch 17
-------------------------------
Test Error: 
 Accuracy: 63.4%, Avg loss: 0.638465 

Epoch 18
-------------------------------
Test Error: 
 Accuracy: 64.0%, Avg loss: 0.634474 

Epoch 19
-------------------------------
Test Error: 
 Accuracy: 64.5%, Avg loss: 0.634486 

Epoch 20
-------------------------------
Test Error: 
 Accuracy: 64.5%, Avg loss: 0.630806 

Epoch 21
-------------------------------
Test Error: 
 Accuracy: 63.8%, Avg loss: 0.637442 

Epoch 22
-------------------------------
Test Error: 
 Accuracy: 64.7%, Avg loss: 0.630708 

Epoch 23
-------------------------------
```

```
Test Error: 
 Accuracy: 64.1%, Avg loss: 0.632921 

Epoch 24
-------------------------------
Test Error: 
 Accuracy: 64.1%, Avg loss: 0.634543 

Epoch 25
-------------------------------
Test Error: 
 Accuracy: 64.9%, Avg loss: 0.632953 
```

In [222…

```
wage_nn_train_err = ((model(wage_nn_train) > 0.5) == jobclass_nn_train).sum().item(
print(f"The training error for the neural network is {1 - wage_nn_train_err}")
wage_nn_test_err = ((model(wage_nn_test) > 0.5) == jobclass_nn_test).sum().item()/l
print(f"The test error for the neural network is {1 - wage_nn_test_err}")
```

```
The training error for the neural network is 0.3505843071786311
The test error for the neural network is 0.3178807947019867
```

This test error beats our other test error by more than 4%!