1. From the perspective of a social scientist, which models did we learn this semester that are useful for ruling out alternative explanations through control variables AND that allow us to observe substantively meaningful information from model coefficients?

1. Linear Regression
2. Logistic Regression
3. Lasso Regression
4. Ridge Regression

2. Describe the main differences between supervised and unsupervised learning.

In supervised learning, the algorithm is trained on a labeled dataset, where input data is associated with corresponding output labels. The primary goal is to predict or classify new instances based on the patterns learned from the labeled examples. On the other hand, unsupervised learning involves working with unlabeled data to identify inherent patterns or groupings within the dataset. The algorithm aims to discover insights without explicit guidance in the form of predefined labels.

3. Is supervised or unsupervised learning the primary approach that is used by machine learning practitioners? For whatever approach you think is secondary, why would you use this approach (what's a good reason to use these kinds of models?)

Supervised learning is the primary approach for many machine learning practitioners. This prevalence may be due to the abundance of labeled data in various domains, facilitating the training of accurate prediction and classification models. Unsupervised learning serves as a valuable secondary approach, offering insights when the data lacks clear labels or when the goal is to uncover hidden structures within the dataset. It can be a helpful approach for data visualization and data pre-processing as well.

4. Which unsupervised learning modeling approaches did we cover this semester? What are the major differences between these techniques?

1. **Principal Components Analysis (PCA)**: transforms high-dimensional data into low-dimensional one; finds a sequence of linear combinations of the variables that have maximal variance and are mutually uncorrelated; used for data visualization or data pre-processing before supervised techniques are applied

2. **Clustering**: discovers unknown subgroups in data, with K-means clustering partitioning data into a pre-specified number of clusters based on Euclidean distances between data points, and hierarchical clustering having no predefined number of clusters but providing a tree-like graph, dendrogram, showing a hierarchy of clusters obtained
3. **Manifold Learning**: visualizes and uncovers the underlying structure of high-dimensional data; reduces the dimensionality of complex datasets; better on preserve nonlinear relationships in data compared to PCA

5. What are the main benefits of using Principal Components Analysis?

1. PCA can effectively reduce the dimensionality of data and visualize it
2. PCA lets you find the output dimension based on the explained variance
3. PCA naturally filters noise from the most important components
4. PCA has straightforward iterative approaches for missing data
5. PCA does not involve choosing the number of neighbors

6. Thinking about neural networks, what are three major differences between a deep multilayer perceptron network and a convolutional neural network model? Be sure to define any key terms in your explanation.

1. **Connectivity**: A deep multilayer perceptron (MLP) is characterized by a fully connected structure, where neurons in one layer are connected to every neuron in the next layer. In contrast, for a convolutional neural network (CNN), each output value depends only on a small number of inputs, which means it has a sparsity of connection.

2. **Parameter Sharing**: Deep MLPs learn features independently across neurons, necessitating a large number of parameters to capture intricate patterns. In contrast, CNNs implement parameter sharing through convolutional kernels or filters, enabling the detection of common features across different regions of the input. This parameter sharing contributes to the efficiency of CNNs in tasks requiring spatial hierarchies, like image recognition.

3. **Weight Initialization**: In deep MLPs, weights are typically initialized randomly, which can lead to different validation accuracy. In contrast, CNNs employ convolutional layers with shared weights and local receptive fields, and weight initialization in CNNs is often performed with consideration for the size of the convolutional kernel.

7. Write the tf.keras code for a multilayer perceptron neural network with the following structure: Three hidden layers. 50 hidden units in the first hidden layer, 100 in the second, and 150 in the third. Activate all hidden layers with relu. The output layer

should be built to classify to five categories. Further, your optimization technique should be stochastic gradient descent. (This code should simply build the architecture of the model and your approach to compile the model. You will not run it on real data.)

In [ ]:
```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Activation

model = Sequential([
    Dense(50, input_shape=input_shape),
    Activation('relu'),
    Dense(100),
    Activation('relu'),
    Dense(150),
    Activation('relu'),
    Dense(5),
    Activation('softmax'),
])

model.compile(loss='categorical_crossentropy',
              optimizer='sgd',
              metrics=['accuracy'])
```

8. Write the tf.keras code for a multilayer perceptron neural network with the following structure: Two hidden layers. 75 hidden units in the first hidden layer and 150 in the second. Activate all hidden layers with relu. The output layer should be built to classify a binary dependent variable. Further, your optimization technique should be stochastic gradient descent. (This code should simply build the architecture of the model and your approach to compile the model. You will not run it on real data.)

In [ ]:
```
model = Sequential([
    Dense(75, input_shape=input_shape),
    Activation('relu'),
    Dense(150),
    Activation('relu'),
    Dense(1),
    Activation('sigmoid'),
])

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

9. Write the tf.keras code for a convolutional neural network with the following structure: Two convolutional layers. 16 filters in the first layer and 28 in the second. Activate all convolutional layers with relu. Use max pooling after each convolutional layer with a 2 by 2 filter. The output layer should be built to classify to ten categories. Further, your optimization technique should be

stochastic gradient descent. (This code should simply build the architecture of the model and your approach to compile the model. You will not run it on real data.)

In [ ]:
```
from keras.layers import Conv2D
from keras.layers import MaxPooling2D
from keras.layers import Flatten

model = Sequential()

model.add(Conv2D(16, (3, 3), activation='relu', input_shape=input_shape))
model.add(MaxPooling2D((2, 2)))

model.add(Conv2D(28, (3, 3), activation='relu'))
model.add(MaxPooling2D((2, 2)))

model.add(Flatten())

model.add(Dense(10, activation='softmax'))

model.compile(loss='categorical_crossentropy',
              optimizer='sgd',
              metrics=['accuracy'])
```

10. Write the keras code for a convolutional neural network with the following structure: Two convolutional layers. 32 filters in the first layer and 32 in the second. Activate all convolutional layers with relu. Use max pooling after each convolutional layer with a 2 by 2 filter. Add two fully connected layers with 128 hidden units in each layer and relu activations. The output layer should be built to classify to six categories. Further, your optimization technique should be stochastic gradient descent. (This code should simply build the architecture of the model and your approach to compile the model. You will not run it on real data.)

In [ ]:
```
model = Sequential()

model.add(Conv2D(32, (3, 3), activation='relu', input_shape=input_shape))
model.add(MaxPooling2D((2, 2)))

model.add(Conv2D(32, (3, 3), activation='relu'))
model.add(MaxPooling2D((2, 2)))

model.add(Flatten())

model.add(Dense(128, activation='relu'))
model.add(Dense(128, activation='relu'))

model.add(Dense(6, activation='softmax'))

model.compile(loss='categorical_crossentropy',
              optimizer='sgd',
              metrics=['accuracy'])
```