Ethics & Deep Learning

Professor Jeff Saltz
Professor Jeff Stanton

ischool.syr.edu
School of Information Studies
SYRACUSE UNIVERSITY

Summary of Previous Learning

What you should know and be able to do at this point :

1. List major skills needed by data scientists and describe the development of a DS project with domain analysis, SMEs, data and modeling
2. Use data frames in R as well as more complex data structures; use multiple strategies for accessing external data from R; use SQL facilities from within R; automate with functions
3. Use and interpret the most common descriptive statistics; describe the effects of randomness on sampling; create and interpret a sampling distribution
4. Use plot and ggplot to visualize data and create maps
5. Create and interpret a multiple regression model using lm()
6. Run a "market basket" analysis using arules; coerce a set of factor variables into a transaction matrix
7. Define, run, and interpret a "supervised" data mining model such as a classification/regression tree (CART) or support vector machine (SVM)
8. Create and use a term-document matrix to conduct simple analyses of unstructured data such as text

Deep Learning

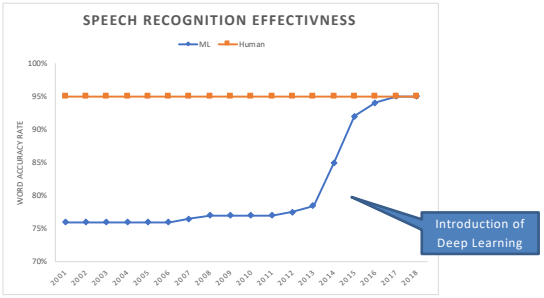
ischool.syr.edu
School of Information Studies
SYRACUSE UNIVERSITY

Example Deep Learning Applications

- Self-driving cars
- Voice Search & Voice-Activated Assistants
- Automatically Add Sounds To Silent Movies
- Automatic Machine Translation
- Sentiment Analysis
- Automatic Text Generation
- Automatic Handwriting Generation
- Image Recognition
- Automatic Image Caption Generation
- Predicting Earthquakes
- Brain Cancer Detection
- Energy Market Price Forecasting

https://medium.com/breathe-publication/top-15-deep-learning-applications-that-will-rule-the-world-in-2018-and-beyond-7c6130c43b01

Deep Learning Impact: Speech recognition



Source: https://www.slideshare.net/kleinerperkins/internet-trends-2017-report
https://sonix.ai/history-of-speech-recognition

Deep Learning Impact: Image Recognition

ImageNet Large Scale Visual Recognition Challenge:

- When deep learning was first used (2012), the deep neural network was 41% better than the previous best solution

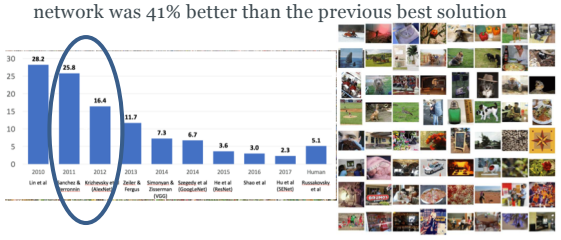
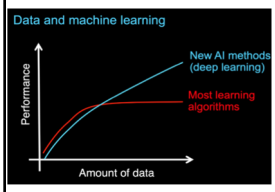


Image credit: https://www.researchgate.net/figure/Examples-in-the-ImageNet-dataset_fig7_314646236
Imagenet info: https://cudeo.io/olo-imagenet/
http://cs231n.stanford.edu/slides/2019/cs231n_2019_lecture01.pdf

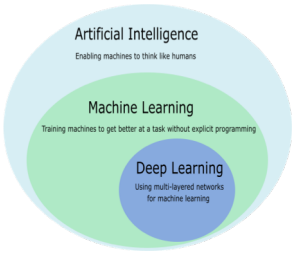
Why is deep learning useful?



- Deep learning:
  - Utilizes large amounts of training data
  - Provides a **flexible**, framework for representing world, visual and linguistic information.
- Feature Engineering:
  - Manually designed features are often **over-specified**, **incomplete** and take a **long time to design** and validate
  - Machine Learned Features are **easy to adapt**, **fast** to learn

Reference,
http://cs229.stanford.edu/materials/CS229-DeepLearning.pdf

What is deep learning?



- An artificial intelligence capability that uses machine learning to imitate the human brain
- Creates a network of "neurons" capable of learning from data
- Also known as:
  - Deep neural learning
  - Deep neural network.

Reference
https://www.datamation.com/artificial-intelligence/ai-vs-machine-learning-vs-deep-learning.html

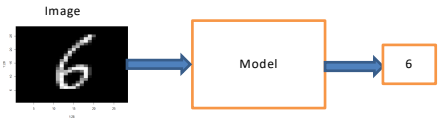
Deep Learning vs SVMs

- SVMs are non-parametric models:
  - **Complexity grows as the training size increases** (i.e. training the model can be expensive computationally)
  - In a complex dataset worst-case scenario, we can end up with as many support vectors as we have samples in the training set.
- SVMs and other simpler models are typically preferred for relatively small data sets with fewer outliers.
- Deep learning needs relatively large datasets (and infrastructure) to train the model.

Reference:
Kurtz, K. (2016, April 22). When Does Deep Learning Work Better Than SVMs or Random Forests? Retrieved from http://www.kdnuggets.com/2016/04/deep-learning-versus-svm-random-forest.html

Typical Deep Learning Application

Predict what it is the digit in an image

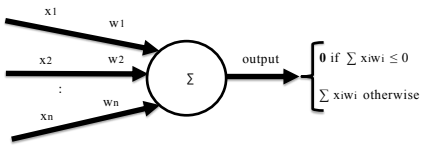


Basic Deep Learning Vocabulary

- **Epoch**: an iteration of learning (one pass of the training data)
- **Accuracy**: % of correct predictions (e.g., based on confusion matrix)
- **Loss**: how inaccurate is the predictions by looking at prediction % (predicting 90% for the correct choice is better than predicting 70%)
- **Tensor:** a generalization of vectors and matrices (i.e., a multidimensional array)
- Two main parameters control the architecture (or topology) of the Deep Learning network:
  - The number of **layers** in the network
  - The number of **neurons (nodes)** in each layer

Node / Neuron / Perceptron

- A computational unit that has one or more weighted input connections, a transfer/activation function that combines the inputs in some way, and an output connection
- An example ***Activation Function*** for a Neuron
  -> Rectified linear unit (ReLU) – the value or 0 if negative



Neuron / Perceptron Example:

Which car should I buy?

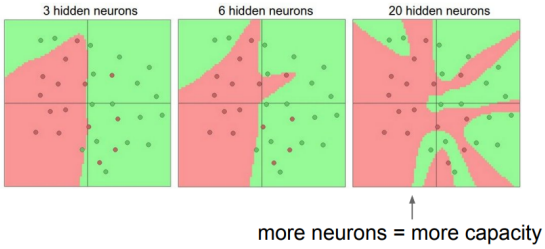
Factor #	Factor description	Factor Weight
X1	Is the miles per gallon (mpg) above 30 mpg?	W1 = 3 (high MPG is important)
X2	Is the horsepower above 200?	W2 = 1 (high horsepower is somewhat important)
X3	Is the transmission type auto?	W3 = 5 (the car really needs to be an auto transmission)

→ Each car will have a score (output) from the neuron
→ The neuron has 3 inputs (factors) to determine the output
→ A car with 32 mpg (X1=1)
190 hp (X2=0)
auto trans (X3=1), will have an output of 8:

Output = X1*W1 + X2*W2 + X3*W3
= 3 + 0 + 5
= 8

The Impact of More Neurons

Setting the number of layers and their sizes



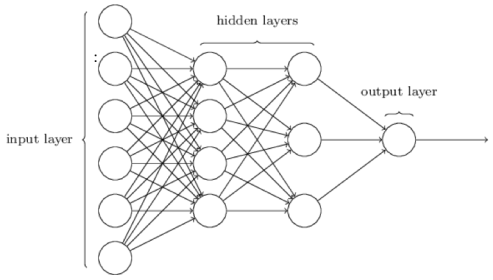
more neurons = more capacity

Image credit: http://cs231n.stanford.edu/slides/2019/cs231n_2019_lecture04.pdf

Why multiple layers of Neurons?

- A *single-layer* neural network can only be used to represent ***linearly separable*** functions:
  - Problems such as when the two classes in a classification problem can be neatly separated by a line
- However, most problems are not linearly separable
- A *multilayer* network can be used to represent *convex* regions:
  - The networks can learn to draw shapes around examples in a high-dimensional space that can separate and classify the regions
  - This overcomes the limitation of linear separability

A Deep Multi Layer Network



Example: Layers Used for Face Recognition

Layers progressively add more insight as the data passes from input layers toward output layers.

- *First layer*: recognizes edges.
- *Second layer*: recognizes facial features like a nose or an ear.
- *Third layer*: recognizes faces.
- *Fourth Layer*: The full face is recognized

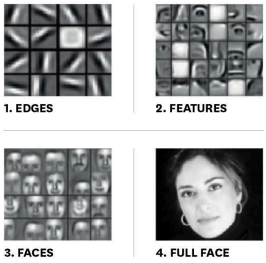
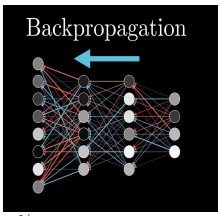


Image credit: https://www.simplilearn.com/deep-learning-tutorial

Backpropagation

Back-propagation is the essence of neural net training.

- The practice of *fine-tuning the weights* of a neural net
- *Based on the error rate (i.e. loss)* obtained in the previous epoch (i.e. iteration).
- Proper tuning of the weights ensures lower error rate
- Makes the model reliable by increasing its generalization.



References:
https://www.youtube.com/watch?v=Ilg3gGewQ5U

Simple Example: loading the data

```
library(tidyverse)
library(caret)

df <- read_csv("testData.csv")
head(df, 5)
# A tibble: 5 x 4
     X1     X2      X3        Y
  <dbl>  <dbl>   <dbl>   <dbl>
1 0.308  0.665  0.882     1
2 0.258  0.0175 0.318     0
3 0.552  0.409  0.454     0
4 0.0564 0.416  0.380     0
5 0.469  0.657  0.390     1
```

Can you figure out the "rule" for y_data?

Deep Learning in R using Keras

```
# Keras provides an R interface
#       to the Python deep learning package Keras

# Keras uses Tensorflow at backend
# to install both Keras and Tensorflow run
#      following the following commands.
Install.packages ("keras")
library(keras)
install_keras()
library(keras)
```

Simple Example: make output binary

```
#change y into a matrix with a 0 or 1 for each choice
df$y_matrix <- to_categorical(df$Y, num_classes = 2)

df[1,]
# A tibble: 1 x 5
     X1     X2      X3      Y          y_matrix[,1]  [,2]
  <dbl>  <dbl>   <dbl>  <dbl>              <dbl> <dbl>
1 0.308  0.665  0.882     1                   0      1

df[2,]
# A tibble: 1 x 5
     X1      X2      X3        Y     y_matrix[,1]  [,2]
  <dbl>   <dbl>   <dbl>  <dbl>                <dbl> <dbl>
1 0.258  0.0175 0.318     0                     1     0
```

Simple Example: Create test & train

```
#create test and train datasets
trainList <-
      createDataPartition(y=df$Y, p=.80, list=FALSE)

trainData <- df[trainList,]
testData <- df[-trainList,]
```

Simple Example: define a model

```
# units is the number of neurons
# ReLU (rectified linear unit) is linear (identity)
#         for positive values, and zero for negative
# softmax is used in the final layer for categorical analysis:
#         it  maps the output to a [0,1] range
#         in such a way that the total sum is 1.
#         hence, is a probability distribution
# input_shape is the number of X inputs (we have 3 inputs)
model = keras_model_sequential() %>%
    layer_dense(units = 64, activation = "relu",
                                    input_shape = 3) %>%
    layer_dense(units = ncol(df$y_matrix),
                                activation = "softmax")
```

Simple Example: compile the model

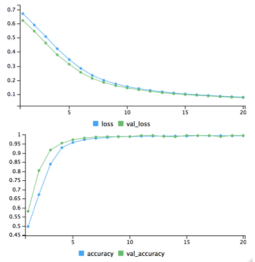
```
#  loss == how to measure error.
#      Use categorical crossentropy in classification problems
#            where only one result can be correct.
#  optimizer == how to optimize during iterations
#       optimizer_rmsprop is a good default
#  metrics == List of metrics to be evaluated by the model
#       often use metrics='accuracy'
compile(model,
      loss = "categorical_crossentropy",
      optimizer = optimizer_rmsprop(),
      metrics = "accuracy")
```

Simple Example: run the model

```
# epochs: # times that the learning algorithm will work
#             through the entire training dataset.
# batch_size: the number of samples to work through before
#               updating the internal model parameters.
#             --> 64, 128, 256 are typical default numbers
# validation_split: The % to validate the model – avoid overfitting
#                       (the rest is to train the model)
x_data <- as.matrix(trainData[,1:3])
history <- fit(model,
        x_data,
        trainData$y_matrix,
        epochs = 20,
        batch_size = 128,
        validation_split = 0.2)
```

Simple Example: run the model

```
...
Epoch 19/20
loss: 0.0709 - accuracy: 0.9945 - val_loss: 0.0698 - val_accuracy: 0.9956
Epoch 20/20
loss: 0.0685 - accuracy: 0.9945 - val_loss: 0.0669 - val_accuracy: 0.9969
```



Simple Example: Use the model

```
#do prediction
x_data_test <- as.matrix(testData[,1:3])
y_data_pred=predict_classes(model, x_data_test)

#see how good the prediction was
confusionMatrix(as.factor(y_data_pred),
                as.factor(testData$Y))
Confusion Matrix and Statistics
          Reference
Prediction  0    1
     0    993   1
     1     10  997

      Accuracy : 0.995
```

Example 2: Use a binary model

```
# units is the number of neurons
# ReLU is linear (identity) for positive values,
#              and zero for negative
# sigmoid is typically used in the final layer for binary
# analysis:
#         it  maps the output to a [0,1] range
# input_shape is the number of X inputs (we have 3)
model = keras_model_sequential() %>%
    layer_dense(units = 64, activation = "relu",
                                    input_shape = 3) %>%
    layer_dense(units =1,  activation = "sigmoid")
```

Example 2: compile the binary model

```
#  loss == how to measure error.
#      Use binary_crossentropy in binary problems
#             where one of two choices is correct.
#  optimizer == how to optimize during iterations
#       optimizer_rmsprop is a good default
#  metrics == List of metrics to be evaluated
#       often use metrics='accuracy'
compile(model,
      loss = "binary_crossentropy",
      optimizer = optimizer_rmsprop(),
      metrics = "accuracy")
```

Example 2: run the binary model

```
#  epochs: # times that the learning algorithm will work
#             through the entire training dataset.
#  batch_size: the number of samples to work through before
#                updating the internal model parameters.
#              --> 64, 128, 256 are typical default numbers
#  validation_split: The % to validate the model – avoid overfitting
#                        (the rest is to train the model)
x_data <- as.matrix(trainData[,1:3])
history = fit(model,
          x_data,
          trainData$Y,
          epochs = 20,
          batch_size = 128,
          validation_split = 0.2)
```

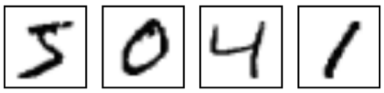
Simple Example 3: use SVM

```
#run the SVM
trainData$Y <- as.factor(trainData$Y)
svm.model <- train(Y ~ X1+X2+X3, data = trainData, method = "svmRadial",
    trControl=trainControl(method = "none"),  preProcess = c("center", "scale"))

#use the model on the test data
testData$Y <- as.factor(testData$Y)
predictValues <- predict(svm.model, newdata=testData)
confusionMatrix(predictValues, testData$Y)
```

```
Confusion Matrix and Statistics
          Reference
Prediction   0    1
         0 992    8
         1  11  989
      Accuracy : 0.9905
```

MNIST: A Slightly More Advanced Problem

- Predict the digit in the image
- MNIST consists of 28 x 28 grayscale images of handwritten digits like below:



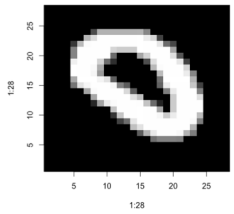
Reference:
Falbel D., Allaire JJ., Chollet F., RStudio, Google (n.d). Keras (n.d). Retrieved from https://keras.rstudio.com/.

Loading MNIST dataset

```
library(tidyverse)
library(keras)
library(caret)

#load the images of digits:
#   from - https://storage.googleapis.com/tensorflow/tf-keras-datasets/mnist.npz
#   note: if loading a local file, need full path
mnist <- dataset_mnist(path = "/.../../mnist.npz")
```

Exploring the MNIST dataset

```
#look at an images (only 28x28) and the known value
img <- mnist$train$x[400,,]
image(1:28, 1:28, img, col = gray.colors(start=0, end=1, n=256))

mnist$train$y[400]
[1] 0
```



Define a MNIST Model

```
model <- keras_model_sequential() %>%
   #first "flatten" each 28x28 image into one vector of 784 numbers (28*28)
   #      Think of this layer as unstacking rows of pixels in the image and
   #      lining them up - this layer only reformats the data.
   layer_flatten(input_shape=c(28, 28)) %>%

   #After the pixels are flattened, the network consists of two layers.
   #start with 128 units (neurons)
   layer_dense(units = 128, activation = "relu") %>%

   #The second (last) layer is a 10-node softmax layer (since 10 possibilities)
   # the second layer returns an array of 10 probability scores that sum to 1.
   layer_dense(units = 10, activation = 'softmax')
```

Compile & Run the MNIST Model

```
#same code as previous example to compile the model
model %>% compile(
    loss = 'categorical_crossentropy',
    optimizer = optimizer_rmsprop(),
    metrics = c('accuracy')
)

#same code as previous example to run the model
y_train <- to_categorical(mnist$train$y, 10)
history <- model %>% fit(
   mnist$train$x, y_train,
   epochs = 30, batch_size = 128,
   validation_split = 0.2
)
```

Evaluating the Model

```
#Explore model accuracy
pred.digit <- model %>% predict_classes(mnist$test$x)
confusionMatrix(as.factor(pred.digit), as.factor(mnist$test$y))
```

```
Overall Statistics
        Accuracy : 0.9777
```

Defining a better MNIST Model

```
#now improve the model
#    add a third layer of nodes (neurons)
#    and add dropout layers
#    (helps with overtraining, sets % of neuron weights to 0 randomly)
model <- keras_model_sequential() %>%
  layer_flatten(input_shape=c(28, 28)) %>%
  layer_dense(units = 256, activation = 'relu') %>%
  layer_dropout(rate = 0.4) %>%
  layer_dense(units = 128, activation = 'relu') %>%
  layer_dropout(rate = 0.3) %>%
  layer_dense(units = 10, activation = 'softmax')
```

Compile & Run the MNIST Model

```
#same code as previous example to compile the model
model %>% compile(
    loss = 'categorical_crossentropy',
    optimizer = optimizer_rmsprop(),
    metrics = c('accuracy')
)

#same code as previous example to run the model
y_train <- to_categorical(mnist$train$y, 10)
history <- model %>% fit(
    mnist$train$x, y_train,
    epochs = 30, batch_size = 128,
    validation_split = 0.2
)
```

Evaluating the Better Model

```
# Explore model accuracy
pred.digit <- model %>% predict_classes(mnist$test$x)
confusionMatrix(as.factor(pred.digit), as.factor(mnist$test$y))
```

Overall Statistics

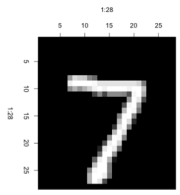
Accuracy : 0.9809

(previous accuracy was: 0.9777)

→ while not a significant better result for this "simple" example, this approach is used for more complicated datasets.

Using the Model

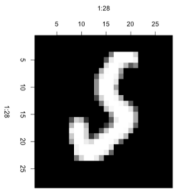
```
pred.digit[1]
[1] 7
mnist$test$y[1]
[1] 7
```



```
img <- mnist$test$x[1,,]
image(1:28, 1:28, img, col = gray.colors(start=0, end=1, n=256))
```

Where did the model mess up?

```
which(pred.digit != mnist$test$y)
 [1]  248  322  341  446  …. 9983

pred.digit[9983]
[1] 6

mnist$test$y[9983]
[1] 5
```



```
img <- mnist$test$x[9983,,]
image(1:28, 1:28, img, col = gray.colors(start=0, end=1, n=256))
```

Could do an SVM – but takes MUCH longer

```
#try SVM
library(caret)
library(e1071)
set.seed(1000)

#test and train data (28x28 is 784 pixels)
x_train <- array_reshape(mnist$train$x, c(nrow(mnist$train$x), 784))
x_test <- array_reshape(mnist$test$x, c(nrow(mnist$test$x), 784))
trainData <- as.data.frame(x_train)
trainData <- cbind(trainData, digit=mnist$train$y)

#run the SVM -  takes about 20-90 minutes on a high end laptop
trainData$digit <- as.factor(trainData$digit)
svm.model <- train(digit ~ ., data = trainData,
                   method = "svmRadial",
                   trControl=trainControl(method = "none"),
                   preProcess = c("center", "scale"))

#Actual resulting model is good (close to previous model), but clearly not scalable
```

Saving / Loading a model

```
# After training completes, you can save your model
model %>% save_model_hdf5("my_model.h5")

# Call load_model_hdf5 to load the model
new_model <- load_model_hdf5("my_model.h5")

# Viewing the loaded model summary
new_model %>% summary()
```

ImageNet

- Contains millions of images with thousands of categories
- Training for this would need a lot of resources and time (requires GPUs)
- But prebuilt models are available to use

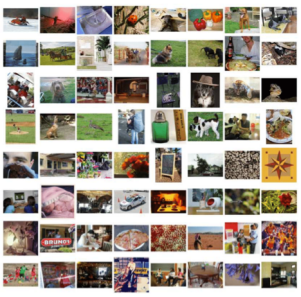


Image credit: https://www.researchgate.net/figure/Examples-in-the-ImageNet-dataset_fig7_314646236

Loading a pre-trained model

Xception: an imageNet model
- Trained on 1,000 classes from ImageNet
- Trained using 60 NVIDIA K80 GPUs
- Training took 3 days!!!!!!!!!

```
# Call load_model_hdf5 to load the model if you have model file
model <- load_model_hdf5("xception.h5")

# Alternatively, one could download and load the trained model using
# "application_xception" function from keras
model <- application_xception(include_top = TRUE,
                  weights = "imagenet", input_tensor = NULL,
                   input_shape = NULL, pooling = NULL, classes = 1000)
```

Reference:
Chollet, F. (2017). Xception: Deep learning with depthwise separable convolutions. In Proceedings of the IEEE conference on computer vision and pattern recognition (pp. 1251-1258).
https://cran.rstudio.com/web/packages/keras/vignettes/applications.html

Get ready to use the model

```
# Compile the model
model %>% compile(
    loss = 'categorical_crossentropy',
    optimizer = optimizer_rmsprop(),
    metrics = c('accuracy')
)

# be able to decode predictions to animal names
imageNames <- fromJSON("imagenet_class_index.json")
imageNamesDF <- as.data.frame(imageNames,
                                        stringsAsFactors = FALSE)
predictedImageNames <- as.character(imageNamesDF[2,1:1000])

predictedImageNames[1:3]
[1] "tench"          "goldfish"        "great_white_shark"
```

Use the pre-trained model (see if the model can identify a cat)

```
#use the model on a test image (of a cat)
testImage <- "cat.jpg"

# first just display image using magick package
library(magick)
im <- image_read(imageName)
plot(im)
```

Image from:
https://pixabay.com/photos/cat-small-kitten-domestic-cat-pet-4611189/

Use model to make a prediction

```
# Next load image, convert to array and reshape for our model
img <- image_load(testImage, target_size = c(299,299))
x <- image_to_array(img)
x <- array_reshape(x, c(1, dim(x)))

#  xception_preprocess_input() should be used for keras image preprocessing
x <- xception_preprocess_input(x)

# predict the type of image
predictions <- model %>% predict(x)

# create a dataframe of prediction % and the name of that animal
predsDF <-
        data.frame(predPercent=predictions, predictedImageNames)
```

What did the model predict?

```
# sort and show the predictions
predsDF <- predsDF %>% arrange(desc(predPercent))
predsDF[1:5,]
```

	predPercent	predictedImageNames
1	0.555413783	tabby
2	0.215213165	tiger_cat
3	0.047023088	Egyptian_cat
4	0.031442393	lynx
5	0.002431044	sock

Use the pre-trained model (see if the model can identify a ball)

```
#use the model on a test image (of a ball)
testImage <- "soccerBall.jpg"

# first just display image using magick package
library(magick)
im <- image_read(imageName)
plot(im)
```

Image from:
https://pixabay.com/photos/football-duel-rush-ball-sport-1331838/

Use model to make a prediction

```
# Next load image, convert to array and reshape for our model
img <- image_load(testImage, target_size = c(299,299))
x <- image_to_array(img)
x <- array_reshape(x, c(1, dim(x)))

#  xception_preprocess_input() should be used for keras image preprocessing
x <- xception_preprocess_input(x)

# predict the type of image
predictions <- model %>% predict(x)

# create a dataframe of prediction % and the name of that animal
predsDF <-
    data.frame(predPercent=predictions,predictedImageNames)
```

What did the model predict?

```
# sort and show the predictions
predsDF <- predsDF %>% arrange(desc(predPercent))
predsDF[1:5,]
```

	predPercent	predictedImageNames
1	0.8440486789	soccer_ball
2	0.0065183444	rugby_ball
3	0.0035442952	football_helmet
4	0.0010038349	baseball
5	0.0009614553	croquet_ball

Deep Learning Sentiment Analysis

- Learn from sentences with a known sentiment
- First, convert each input (sentence) into a vector
  - Machine Learning works on numbers, not text strings
  - Text Embedding: Mapping text to a vector of numbers
    → converts text (words or sentences) into numerical vectors
  - A vector of a 0 or 1 for each possible word is one approach
    → but it would be very long vectors and it doesn't take into account words close to each provides additional info
- Then, use the vectors to train a model

Sentiment Analysis: Text Embedding

- How to embed? **Universal Sentence Encoder** from Google:
  - Encodes text into high dimensional vectors (512 length numbers)
  - Encodes the meaning of the sentence, fine-tuned for text similarity
  - Works best with long text (as opposed to short sentences)
  - These vectors can be used for sentiment analysis, semantic similarity, clustering, and other natural language tasks.
  - A pre-trained Encoder is available: Tensorflow-hub.

https://amitness.com/2020/06/universal-sentence-encoder

Sentiment Analysis: Text Embedding

```
# load tensor flow hub (only do installs once!)
install.packages("tfhub")
library(tfhub)
install_tfhub()

# file is at: https://tfhub.dev/google/universal-sentence-encoder/4
model_use <- hub_load("universal-sentence-encoder_4")

test <- c("this was great", "this was bad")
model_results <- model_use(test)

# get the actual vectors of numbers
mapped_vectors <- model_results$numpy()
str(mapped_vectors )
num [1:2, 1:512] -1.49e-03 -1.51e-05 2.91e-02 1.16e-02 3.69e-02 ...
```

Sentiment Analysis: Load Training Data

```
# read movie review dataset - see
#    www.kaggle.com/lakshmi25npathi/imdb-dataset-of-50k-movie-reviews
#    https://ai.stanford.edu/~amaas/data/sentiment/
#    https://github.com/laxmimerit/IMDB-Movie-Reviews-Large-Dataset-50k
data = read_csv('IMDB Dataset.csv')

# reduce data file to 2/3 of it's size (train and the test)
ind<-createDataPartition(data$sentiment,p=2/3,list = FALSE)
train<-data[ind,]

# have sentiment be either 0 or 1
train$sentiment <- as.integer(train$sentiment=="positive")
sentiment <- train$sentiment

# map sentences to a vector (NOTE: THIS WILL TAKE ~30 minutes)
model_results <- model_use(train$review)
mappedSentences <- model_results$numpy()
```

Sentiment Analysis: Define the Model

```
# create a binary model – same as previous DL models
#   input shape is the length of sentence vector - 512
#   units is the length input data

model <- keras_model_sequential() %>%
  layer_dense(units = length(sentiment), activation = "relu",
              input_shape = 512, dtype="float32") %>%
  layer_dense(units = 128, activation = 'relu') %>%
  layer_dropout(rate = 0.3) %>%
  layer_dense(units = 64, activation = 'relu') %>%
  layer_dropout(rate = 0.3) %>%
  layer_dense(units = 32, activation = 'relu') %>%
  layer_dense(units =1,  activation = "sigmoid")
```

Sentiment Analysis: Train the Model

```
# similar to other deep learning models -
compile(model,
        loss = "binary_crossentropy",
        optimizer = optimizer_rmsprop(),
        metrics = "accuracy")

num_epochs <- 10
b_size <- 32

# Fit the model – this will take ~30 minutes
history <- model %>% fit(
    mappedSentences,
    sentiment,
    epochs = num_epochs, batch_size = b_size,
    validation_split = 0.1 )
```

Sentiment Analysis: Test the Model

```
# See how we do with the test data
test<-data[-ind,]

# map each sentence to a vector – this will take some time
model_results_test  <- model_use(test$review)
mappedSentences_test <- model_results_test$numpy()

# Make the predictions
prediction_test <- predict(model, mappedSentences_test)
prediction_test1 <- round(prediction_test, digits=0)

# map each sentence to a vector – this will take some time
confusionMatrix(as.factor(prediction_test1), as.factor(test$sentiment))
```

Sentiment Analysis: Test Results

```
# map each sentence to a vector – this will take some time
confusionMatrix(as.factor(prediction_test1), as.factor(test$sentiment))
```

```
Confusion Matrix and Statistics

          Reference
Prediction    0    1
         0 7285 1262
         1 1048 7071

      Accuracy : 0.8614
        95% CI : (0.8561, 0.8666)
   No Information Rate : 0.5
   P-Value [Acc > NIR] : < 2.2e-16
```

Sentiment Analysis: Test Results

```
# map each sentence to a vector – this will take some time
confusionMatrix(as.factor(prediction_test1), as.factor(test$sentiment))
```

```
Confusion Matrix and Statistics

          Reference
Prediction    0    1
         0 7285 1262
         1 1048 7071

      Accuracy : 0.8614
        95% CI : (0.8561, 0.8666)
   No Information Rate : 0.5
   P-Value [Acc > NIR] : < 2.2e-16
```

```
paste("prediction=",
prediction_test1[100],
      "test=", test$sentiment[100])
[1] "prediction= 1 test= 1"

paste(substring(test$review[100], 1,200))
[1] "I would like to know if anyone know
how I can get a copy of the movie,
\"That's the way of the World\". It's been
about 30 years since I've seen this movie,
and I would like to see it again. Earth Wind
"
```

Sentiment Analysis: Explore the Model

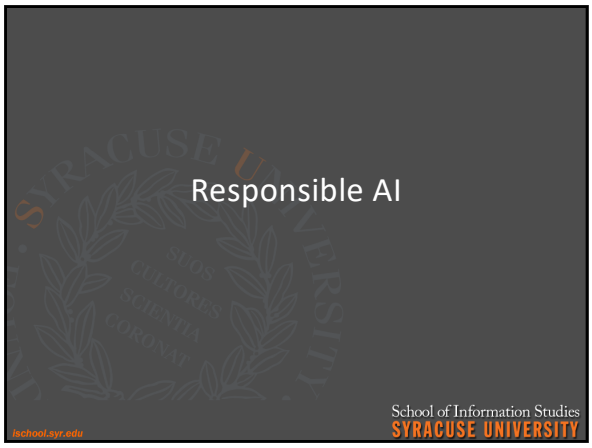
```
# Some example sentences
test_sentances <- c("this is wonderful",
          "your package is ready for pickup",
          "this is  terrible!", "your shipment has been delayed",
          "this is not good", "this is not bad",
           "This movie is horrible, but I love it", "this is green")

# Map/embed the sentences
model_output <- model_use(test_sentances)
mapped_test <- model_output$numpy()

# Predict Positive or Negative, using the Deep Learning Model
prediction <- predict(model, mapped_test)
prediction <- round(prediction, digits=2)
data.frame(prediction, test_sentances)
```

Sentiment Analysis: The Results

```
# Some example sentences
data.frame(prediction, test_sentances)
```

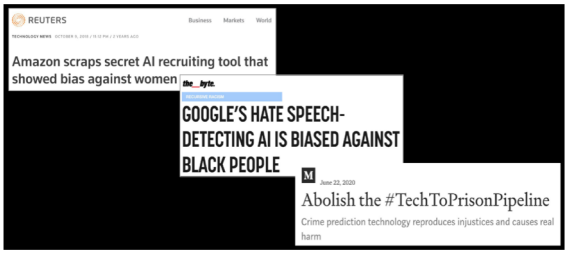
	prediction	test_sentences
1	1	this is wonderful
2	1	your package is ready for pickup
3	0	this is terrible!
4	0	your shipment has been delayed
5	0	this is not good
6	0	this is not bad
7	0	This movie is horrible, but I love it
8	1	this is green

Responsible AI

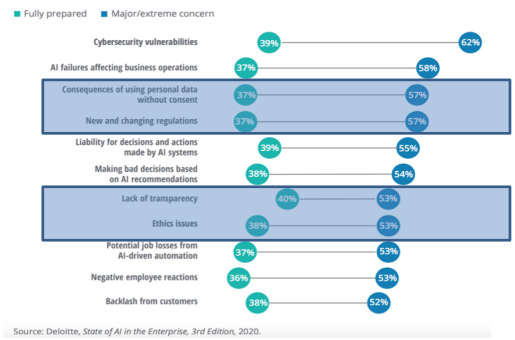


Examples of Models "Gone Wrong"

- **Why focus/think about ethics?**



Ethics / Risks: Industry Perceptions



Focus of Ethics Discussion

- **Avoid potential issues in the creation and use of the model**
  - Help ensure an ethical analysis
  - Key questions during the life of the project
  - Not focused on societal issues
    -> But on what the team should consider
- **Examples of "things to think about"**
  - Is the model biased?
  - Is the model fair?

What is Bias?

In data science, bias is a deviation from expectation in the data or outcome.

What is Bias?

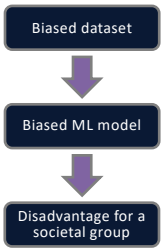
In data science, bias is a deviation from expectation in the data or outcome.

It is actually very common.

But the implications could be serious in some circumstances.

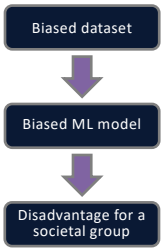
Types of Bias

- **Historical bias**:
  existing bias in the data
  and socio-technical issues in the world



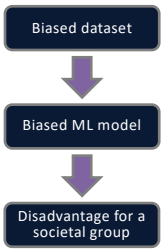
Types of Bias

- **Historical bias**:
  existing bias in the data
  and socio-technical issues in the world
- **Representation bias**:
  comes from the way we define and
  sample a population (or dataset)



Types of Bias

- **Historical bias**: existing bias in the data and socio-technical issues in the world
- **Representation bias**: comes from the way we define and sample a population (or dataset)
- **Measurement bias**: comes from the way we choose, utilize, and measure a particular feature



What is Fair?

In everyday life, there's no easy answer

What is Fair?

In everyday life, there's no easy answer → In data science, there's no easy answer

Question

Which is most fair to give a loan?

A) Ensure loans are made at the same rate to two different groups?

B) Focus on each person's expected payback rate & group attribute?

C) Focus on each person's expected payback rate, ignoring the group attribute?

D) None of the above

Question

Which is most fair to give a loan?

A) Ensure loans are made at the same rate to two different groups?

B) Focus on each person's expected payback rate & group attribute?

C) Focus on each person's expected payback rate, ignoring the group attribute?

D) None of the above

Answer

Well … there's no definitive answer

- Lack consensus about which fairness to apply
- Each type of fairness requires both technical & non-technical decisions and trade-offs

Four Types of Fairness

Max Profit

Most profitable, since there are no constraints.

But the groups have different thresholds, meaning they are held to different standards

Four Types of Fairness

Max Profit

Most profitable, since there are no constraints.

But the groups have different thresholds, meaning they are held to different standards

Group Unaware

Groups have the same threshold

One group will have fewer actions

There might be bias in the training data

Four Types of Fairness

Max Profit

Most profitable, since there are no constraints.

But the groups have different thresholds, meaning they are held to different standards

Group Unaware

Groups have the same threshold

One group will have fewer actions

There might be bias in the training data

Demographic Parity

Number of actions for every group is the same

But among people who qualify one group is at a disadvantage

Four Types of Fairness

Max Profit

Most profitable, since there are no constraints.

But the groups have different thresholds, meaning they are held to different standards

Group Unaware

Groups have the same threshold

One group will have fewer actions

There might be bias in the training data

Demographic Parity

Number of actions for every group is the same

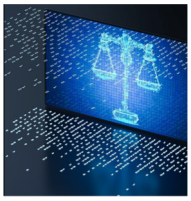
But among people who qualify one group is at a disadvantage

Equal Opportunity

Among people who meet the threshold, all groups do equally well

Fairness Is Hard

- Lack consensus about which fairness to apply.
- Each type of fairness requires both technical and nontechnical decisions and trade-offs.



Example Ethics Questions

Project initiation and management-related challenges

Q1: Which laws and regulations might be applicable to our project?

It is important to consider which laws and regulations might be relevant.

Q2: How are we achieving ethical accountability?

It should be clear who will be accountable to minimize the potential harm.

Example Ethics Questions

Model-related challenges

How have we identified and minimized any bias in the data or in the model?
Models built using biased data can also be biased.

How was any potential modeler bias identified, and if appropriate, mitigated?
There could be subjectivity within the model building process.

How transparent does the model need to be and how is transparency achieved?
How important it is that the model can be explained and understood.

What are likely misinterpretations of the results and what can be done to prevent those misinterpretations?
The decisions made via an ML model should reflect the scale, accuracy, and precision of the data that was used and the resulting model.

Example Ethics Questions (Cont.)

Data-related challenges

How might the legal rights of an individual be impinged by our use of data?

The project must have the right to use the data for that purpose.

How might individuals' privacy and anonymity be impinged?

How to ensure anonymity must be reexamined due to aggregations and linking.

How do we know that the data is ethically available for its intended use?

Being able to access data does not mean that it is ethical to use that data.

How do we know that the data is valid for its intended use?

This includes data accuracy and imputing missing values or excluding records.

Ethics Question for Discussion

What should I do if my manager asks me to do something—not use data incorrectly or not check for bias in a model?

Learning resources

- Free online tutorial, blogs, and videos available
  - LinkedIn Learning at Syracuse University (for SU students, faculty and Staff) (https://answers.syr.edu/display/lnkdlrn/LinkedIn+Learning+at+Syracuse+University)
  - *10 Free Courses for Machine Learning and Data Science* (https://www.kdnuggets.com/2018/11/10-free-must-see-courses-machine-learning-data-science.html)
  - *R Markdown Notebooks for "Deep Learning with R"* (https://github.com/jjallaire/deep-learning-with-r-notebooks)

Appendix: Tensors