



Understanding and Visualizing Data Iteration in Machine Learning

Fred Hohman, Georgia Tech, @fredhohman

Kanit Wongsuphasawat, Apple, @kanitw

Mary Beth Kery, Carnegie Mellon University, @mbkery

Kayur Patel, Apple, @foil

```
[15]: '''Trains a simple convnet on the MNIST dataset.
Gets to 99.25% test accuracy after 12 epochs
(there is still a lot of margin for parameter tuning).
16 seconds per epoch on a GRID K520 GPU.
'''

from __future__ import print_function
import keras
from keras.datasets import mnist
from keras.models import Sequential
from keras.layers import Dense, Dropout, Flatten
from keras.layers import Conv2D, MaxPooling2D
from keras import backend as K

batch_size = 128
num_classes = 10
epochs = 10

# input image dimensions
img_rows, img_cols = 28, 28

# the data, split between train and test sets
(x_train, y_train), (x_test, y_test) = mnist.load_data()

if K.image_data_format() == 'channels_first':
    x_train = x_train.reshape(x_train.shape[0], 1, img_rows, img_cols)
    x_test = x_test.reshape(x_test.shape[0], 1, img_rows, img_cols)
    input_shape = (1, img_rows, img_cols)
else:
    x_train = x_train.reshape(x_train.shape[0], img_rows, img_cols, 1)
    x_test = x_test.reshape(x_test.shape[0], img_rows, img_cols, 1)
    input_shape = (img_rows, img_cols, 1)
```

```
[16]: x_train = x_train.astype('float32')
x_test = x_test.astype('float32')
x_train /= 255
x_test /= 255
x_train = x_train[0:600]
x_test = x_test[0:100]
y_train = y_train[0:600]
y_test = y_test[0:100]
# print('x_train shape:', x_train.shape)
# print('x_test shape:', x_test.shape)
# print('y_train shape:', y_train.shape)
# print('y_test shape:', y_test.shape)
```

```
[17]: # convert class vectors to binary class matrices
y_train = keras.utils.to_categorical(y_train, num_classes)
y_test = keras.utils.to_categorical(y_test, num_classes)

model = Sequential()
model.add(Conv2D(32, kernel_size=(3, 3), activation='relu', input_shape=input_shape))
model.add(Conv2D(64, kernel_size=(3, 3), activation='relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Flatten())
model.add(Dense(128, activation='relu'))
model.add(Dropout(0.5))
model.add(Dense(num_classes, activation='softmax'))

model.compile(loss=keras.losses.categorical_crossentropy,
              optimizer=keras.optimizers.Adadelta(),
              metrics=['accuracy'])

model.fit(x_train, y_train,
          batch_size=batch_size,
          epochs=epochs,
          verbose=1,
          validation_data=(x_test, y_test))
score = model.evaluate(x_test, y_test, verbose=0)
print('Test loss:', score[0])
print('Test accuracy:', score[1])
```

Test accuracy: 0.81

```
Train on 600 samples, validate on 100 samples
Epoch 1/10
600/600 [=====] - 1s 2ms/step - loss: 2.0781 - accuracy: 0.2733 - val_loss: 1.6467 - val_accuracy: 0.4900
Epoch 2/10
600/600 [=====] - 1s 2ms/step - loss: 1.6297 - accuracy: 0.4983 - val_loss: 1.4862 - val_accuracy: 0.4600
Epoch 3/10
600/600 [=====] - 1s 2ms/step - loss: 1.2831 - accuracy: 0.5967 - val_loss: 0.9324 - val_accuracy: 0.6700
Epoch 4/10
600/600 [=====] - 1s 2ms/step - loss: 0.8546 - accuracy: 0.7250 - val_loss: 0.7551 - val_accuracy: 0.7800
Epoch 5/10
600/600 [=====] - 1s 2ms/step - loss: 0.6299 - accuracy: 0.8000 - val_loss: 0.5108 - val_accuracy: 0.8100
Epoch 6/10
600/600 [=====] - 1s 2ms/step - loss: 0.5073 - accuracy: 0.8367 - val_loss: 0.4397 - val_accuracy: 0.8700
Epoch 7/10
600/600 [=====] - 1s 2ms/step - loss: 0.4291 - accuracy: 0.8700 - val_loss: 0.4042 - val_accuracy: 0.8700
Epoch 8/10
600/600 [=====] - 1s 2ms/step - loss: 0.3493 - accuracy: 0.8783 - val_loss: 0.3459 - val_accuracy: 0.8500
Epoch 9/10
600/600 [=====] - 1s 2ms/step - loss: 0.3295 - accuracy: 0.8950 - val_loss: 0.4374 - val_accuracy: 0.8500
Epoch 10/10
600/600 [=====] - 1s 2ms/step - loss: 0.2969 - accuracy: 0.9083 - val_loss: 0.4780 - val_accuracy: 0.8100
Test loss: 0.4779697322845459
Test accuracy: 0.8100000023841858
```

Convolutional Neural Network on MNIST

```
[15]: '''Trains a simple convnet on the MNIST dataset.
Gets to 99.25% test accuracy after 12 epochs
(there is still a lot of margin for parameter tuning).
16 seconds per epoch on a GRID K520 GPU.
'''

from __future__ import print_function
import keras
from keras.datasets import mnist
from keras.models import Sequential
from keras.layers import Dense, Dropout, Flatten
from keras.layers import Conv2D, MaxPooling2D
from keras import backend as K

batch_size = 128
num_classes = 10
epochs = 10

# input image dimensions
img_rows, img_cols = 28, 28

# the data, split between train and test sets
(x_train, y_train), (x_test, y_test) = mnist.load_data()

if K.image_data_format() == 'channels_first':
    x_train = x_train.reshape(x_train.shape[0], 1, img_rows, img_cols)
    x_test = x_test.reshape(x_test.shape[0], 1, img_rows, img_cols)
    input_shape = (1, img_rows, img_cols)
else:
    x_train = x_train.reshape(x_train.shape[0], img_rows, img_cols, 1)
    x_test = x_test.reshape(x_test.shape[0], img_rows, img_cols, 1)
    input_shape = (img_rows, img_cols, 1)
```

```
[16]: x_train = x_train.astype('float32')
x_test = x_test.astype('float32')
x_train /= 255
x_test /= 255
x_train = x_train[0:600]
x_test = x_test[0:100]
y_train = y_train[0:600]
y_test = y_test[0:100]
# print('x_train shape:', x_train.shape)
# print('x_test shape:', x_test.shape)
# print('y_train shape:', y_train.shape)
# print('y_test shape:', y_test.shape)
```

```
[17]: # convert class vectors to binary class matrices
y_train = keras.utils.to_categorical(y_train, num_classes)
y_test = keras.utils.to_categorical(y_test, num_classes)

model = Sequential()
model.add(Conv2D(32, kernel_size=(3, 3), activation='relu', input_shape=input_shape))
model.add(Conv2D(64, kernel_size=(3, 3), activation='relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Flatten())
model.add(Dense(128, activation='relu'))
model.add(Dropout(0.5))
model.add(Dense(num_classes, activation='softmax'))

model.compile(loss=keras.losses.categorical_crossentropy,
              optimizer=keras.optimizers.Adadelta(),
              metrics=['accuracy'])

model.fit(x_train, y_train,
        batch_size=batch_size,
        epochs=epochs,
        verbose=1,
        validation_data=(x_test, y_test))
score = model.evaluate(x_test, y_test, verbose=0)
print('Test loss:', score[0])
print('Test accuracy:', score[1])
```

Test accuracy: 0.81

```
Train on 600 samples, validate on 100 samples
Epoch 1/10
600/600 [=====] - 1s 2ms/step - loss: 2.0781 - accuracy: 0.2733 - val_loss: 1.6467 - val_accuracy: 0.4900
Epoch 2/10
600/600 [=====] - 1s 2ms/step - loss: 1.6297 - accuracy: 0.4983 - val_loss: 1.4862 - val_accuracy: 0.4600
Epoch 3/10
600/600 [=====] - 1s 2ms/step - loss: 1.2831 - accuracy: 0.5967 - val_loss: 0.9324 - val_accuracy: 0.6700
Epoch 4/10
600/600 [=====] - 1s 2ms/step - loss: 0.8546 - accuracy: 0.7250 - val_loss: 0.7551 - val_accuracy: 0.7800
Epoch 5/10
600/600 [=====] - 1s 2ms/step - loss: 0.6299 - accuracy: 0.8000 - val_loss: 0.5108 - val_accuracy: 0.8100
Epoch 6/10
600/600 [=====] - 1s 2ms/step - loss: 0.5073 - accuracy: 0.8367 - val_loss: 0.4397 - val_accuracy: 0.8700
Epoch 7/10
600/600 [=====] - 1s 2ms/step - loss: 0.4291 - accuracy: 0.8700 - val_loss: 0.4042 - val_accuracy: 0.8700
Epoch 8/10
600/600 [=====] - 1s 2ms/step - loss: 0.3493 - accuracy: 0.8783 - val_loss: 0.3459 - val_accuracy: 0.8500
Epoch 9/10
600/600 [=====] - 1s 2ms/step - loss: 0.3295 - accuracy: 0.8950 - val_loss: 0.4374 - val_accuracy: 0.8500
Epoch 10/10
600/600 [=====] - 1s 2ms/step - loss: 0.2969 - accuracy: 0.9083 - val_loss: 0.4780 - val_accuracy: 0.8100
Test loss: 0.4779697322845459
Test accuracy: 0.8100000023841858
```

Convolutional Neural Network on MNIST

How to improve performance?

```
[15]: '''Trains a simple convnet on the MNIST dataset.
Gets to 99.25% test accuracy after 12 epochs
(there is still a lot of margin for parameter tuning).
16 seconds per epoch on a GRID K520 GPU.
'''

from __future__ import print_function
import keras
from keras.datasets import mnist
from keras.models import Sequential
from keras.layers import Dense, Dropout, Flatten
from keras.layers import Conv2D, MaxPooling2D
from keras import backend as K

batch_size = 128
num_classes = 10
epochs = 10

# input image dimensions
img_rows, img_cols = 28, 28

# the data, split between train and test sets
(x_train, y_train), (x_test, y_test) = mnist.load_data()

if K.image_data_format() == 'channels_first':
    x_train = x_train.reshape(x_train.shape[0], 1, img_rows, img_cols)
    x_test = x_test.reshape(x_test.shape[0], 1, img_rows, img_cols)
    input_shape = (1, img_rows, img_cols)
else:
    x_train = x_train.reshape(x_train.shape[0], img_rows, img_cols, 1)
    x_test = x_test.reshape(x_test.shape[0], img_rows, img_cols, 1)
    input_shape = (img_rows, img_cols, 1)
```

```
[16]: x_train = x_train.astype('float32')
x_test = x_test.astype('float32')
x_train /= 255
x_test /= 255
x_train = x_train[0:600]
x_test = x_test[0:100]
y_train = y_train[0:600]
y_test = y_test[0:100]
# print('x_train shape:', x_train.shape)
# print('x_test shape:', x_test.shape)
# print('y_train shape:', y_train.shape)
# print('y_test shape:', y_test.shape)
```

```
[17]: # convert class vectors to binary class matrices
y_train = keras.utils.to_categorical(y_train, num_classes)
y_test = keras.utils.to_categorical(y_test, num_classes)

model = Sequential()
model.add(Conv2D(32, kernel_size=(3, 3), activation='relu', input_shape=input_shape))
model.add(Conv2D(64, kernel_size=(3, 3), activation='relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.5))
model.add(Flatten())
model.add(Dense(128, activation='relu'))
model.add(Dropout(0.5))
model.add(Dense(num_classes, activation='softmax'))

model.compile(loss=keras.losses.categorical_crossentropy,
              optimizer=keras.optimizers.Adadelta(),
              metrics=['accuracy'])

model.fit(x_train, y_train,
          batch_size=batch_size,
          epochs=epochs,
          verbose=1,
          validation_data=(x_test, y_test))
score = model.evaluate(x_test, y_test, verbose=0)
print('Test loss:', score[0])
print('Test accuracy:', score[1])
```

Test accuracy: 0.81

```
Train on 600 samples, validate on 100 samples
Epoch 1/10
600/600 [=====] - 1s 2ms/step - loss: 2.0781 - accuracy: 0.2733 - val_loss: 1.6467 - val_accuracy: 0.4900
Epoch 2/10
600/600 [=====] - 1s 2ms/step - loss: 1.6297 - accuracy: 0.4983 - val_loss: 1.4862 - val_accuracy: 0.4600
Epoch 3/10
600/600 [=====] - 1s 2ms/step - loss: 1.2831 - accuracy: 0.5967 - val_loss: 0.9324 - val_accuracy: 0.6700
Epoch 4/10
600/600 [=====] - 1s 2ms/step - loss: 0.8546 - accuracy: 0.7250 - val_loss: 0.7551 - val_accuracy: 0.7800
Epoch 5/10
600/600 [=====] - 1s 2ms/step - loss: 0.6299 - accuracy: 0.8000 - val_loss: 0.5108 - val_accuracy: 0.8100
Epoch 6/10
600/600 [=====] - 1s 2ms/step - loss: 0.5073 - accuracy: 0.8367 - val_loss: 0.4397 - val_accuracy: 0.8700
Epoch 7/10
600/600 [=====] - 1s 2ms/step - loss: 0.4291 - accuracy: 0.8700 - val_loss: 0.4042 - val_accuracy: 0.8700
Epoch 8/10
600/600 [=====] - 1s 2ms/step - loss: 0.3493 - accuracy: 0.8783 - val_loss: 0.3459 - val_accuracy: 0.8500
Epoch 9/10
600/600 [=====] - 1s 2ms/step - loss: 0.3295 - accuracy: 0.8950 - val_loss: 0.4374 - val_accuracy: 0.8500
Epoch 10/10
600/600 [=====] - 1s 2ms/step - loss: 0.2969 - accuracy: 0.9083 - val_loss: 0.4780 - val_accuracy: 0.8100
Test loss: 0.4779697322845459
Test accuracy: 0.8100000023841858
```

Convolutional Neural Network on MNIST

How to improve performance?

A. Different architecture

```
[15]: '''Trains a simple convnet on the MNIST dataset.
Gets to 99.25% test accuracy after 12 epochs
(there is still a lot of margin for parameter tuning).
16 seconds per epoch on a GRID K520 GPU.
'''

from __future__ import print_function
import keras
from keras.datasets import mnist
from keras.models import Sequential
from keras.layers import Dense, Dropout, Flatten
from keras.layers import Conv2D, MaxPooling2D
from keras import backend as K

batch_size = 128
num_classes = 10
epochs = 10

# input image dimensions
img_rows, img_cols = 28, 28

# the data, split between train and test sets
(x_train, y_train), (x_test, y_test) = mnist.load_data()

if K.image_data_format() == 'channels_first':
    x_train = x_train.reshape(x_train.shape[0], 1, img_rows, img_cols)
    x_test = x_test.reshape(x_test.shape[0], 1, img_rows, img_cols)
    input_shape = (1, img_rows, img_cols)
else:
    x_train = x_train.reshape(x_train.shape[0], img_rows, img_cols, 1)
    x_test = x_test.reshape(x_test.shape[0], img_rows, img_cols, 1)
    input_shape = (img_rows, img_cols, 1)
```

```
[16]: x_train = x_train.astype('float32')
x_test = x_test.astype('float32')
x_train /= 255
x_test /= 255
x_train = x_train[0:600]
x_test = x_test[0:100]
y_train = y_train[0:600]
y_test = y_test[0:100]
# print('x_train shape:', x_train.shape)
# print('x_test shape:', x_test.shape)
# print('y_train shape:', y_train.shape)
# print('y_test shape:', y_test.shape)
```

```
[17]: # convert class vectors to binary class matrices
y_train = keras.utils.to_categorical(y_train, num_classes)
y_test = keras.utils.to_categorical(y_test, num_classes)

model = Sequential()
model.add(Conv2D(32, kernel_size=(3, 3), activation='relu', input_shape=input_shape))
model.add(Conv2D(64, kernel_size=(3, 3), activation='relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.5))
model.add(Flatten())
model.add(Dense(128, activation='relu'))
model.add(Dropout(0.5))
model.add(Dense(num_classes, activation='softmax'))

model.compile(loss=keras.losses.categorical_crossentropy,
              optimizer=keras.optimizers.Adadelta(),
              metrics=['accuracy'])

model.fit(x_train, y_train,
        batch_size=batch_size,
        epochs=epochs,
        verbose=1,
        validation_data=(x_test, y_test))
score = model.evaluate(x_test, y_test, verbose=0)
print('Test loss:', score[0])
print('Test accuracy:', score[1])
```

Test accuracy: 0.81

```
Train on 600 samples, validate on 100 samples
Epoch 1/10
600/600 [=====] - 1s 2ms/step - loss: 2.0781 - accuracy: 0.2733 - val_loss: 1.6467 - val_accuracy: 0.4900
Epoch 2/10
600/600 [=====] - 1s 2ms/step - loss: 1.6297 - accuracy: 0.4983 - val_loss: 1.4862 - val_accuracy: 0.4600
Epoch 3/10
600/600 [=====] - 1s 2ms/step - loss: 1.2831 - accuracy: 0.5967 - val_loss: 0.9324 - val_accuracy: 0.6700
Epoch 4/10
600/600 [=====] - 1s 2ms/step - loss: 0.8546 - accuracy: 0.7250 - val_loss: 0.7551 - val_accuracy: 0.7800
Epoch 5/10
600/600 [=====] - 1s 2ms/step - loss: 0.6299 - accuracy: 0.8000 - val_loss: 0.5108 - val_accuracy: 0.8100
Epoch 6/10
600/600 [=====] - 1s 2ms/step - loss: 0.5073 - accuracy: 0.8367 - val_loss: 0.4397 - val_accuracy: 0.8700
Epoch 7/10
600/600 [=====] - 1s 2ms/step - loss: 0.4291 - accuracy: 0.8700 - val_loss: 0.4042 - val_accuracy: 0.8700
Epoch 8/10
600/600 [=====] - 1s 2ms/step - loss: 0.3493 - accuracy: 0.8783 - val_loss: 0.3459 - val_accuracy: 0.8500
Epoch 9/10
600/600 [=====] - 1s 2ms/step - loss: 0.3295 - accuracy: 0.8950 - val_loss: 0.4374 - val_accuracy: 0.8500
Epoch 10/10
600/600 [=====] - 1s 2ms/step - loss: 0.2969 - accuracy: 0.9083 - val_loss: 0.4780 - val_accuracy: 0.8100
Test loss: 0.4779697322845459
Test accuracy: 0.8100000023841858
```

Convolutional Neural Network on MNIST

How to improve performance?

A. Different architecture

B. Tweak hyperparameter

```
[15]: '''Trains a simple convnet on the MNIST dataset.
Gets to 99.25% test accuracy after 12 epochs
(there is still a lot of margin for parameter tuning).
16 seconds per epoch on a GRID K520 GPU.
'''

from __future__ import print_function
import keras
from keras.datasets import mnist
from keras.models import Sequential
from keras.layers import Dense, Dropout, Flatten
from keras.layers import Conv2D, MaxPooling2D
from keras import backend as K

batch_size = 128
num_classes = 10
epochs = 10

# input image dimensions
img_rows, img_cols = 28, 28

# the data, split between train and test sets
(x_train, y_train), (x_test, y_test) = mnist.load_data()

if K.image_data_format() == 'channels_first':
    x_train = x_train.reshape(x_train.shape[0], 1, img_rows, img_cols)
    x_test = x_test.reshape(x_test.shape[0], 1, img_rows, img_cols)
    input_shape = (1, img_rows, img_cols)
else:
    x_train = x_train.reshape(x_train.shape[0], img_rows, img_cols, 1)
    x_test = x_test.reshape(x_test.shape[0], img_rows, img_cols, 1)
    input_shape = (img_rows, img_cols, 1)
```

```
[16]: x_train = x_train.astype('float32')
x_test = x_test.astype('float32')
x_train /= 255
x_test /= 255
x_train = x_train[0:600]
x_test = x_test[0:100]
y_train = y_train[0:600]
y_test = y_test[0:100]
# print('x_train shape:', x_train.shape)
# print('x_test shape:', x_test.shape)
# print('y_train shape:', y_train.shape)
# print('y_test shape:', y_test.shape)
```

```
[17]: # convert class vectors to binary class matrices
y_train = keras.utils.to_categorical(y_train, num_classes)
y_test = keras.utils.to_categorical(y_test, num_classes)

model = Sequential()
model.add(Conv2D(32, kernel_size=(3, 3), activation='relu', input_shape=input_shape))
model.add(Conv2D(64, kernel_size=(3, 3), activation='relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Flatten())
model.add(Dense(128, activation='relu'))
model.add(Dropout(0.5))
model.add(Dense(num_classes, activation='softmax'))

model.compile(loss=keras.losses.categorical_crossentropy,
              optimizer=keras.optimizers.Adadelta(),
              metrics=['accuracy'])

model.fit(x_train, y_train,
          batch_size=batch_size,
          epochs=epochs,
          verbose=1,
          validation_data=(x_test, y_test))
score = model.evaluate(x_test, y_test, verbose=0)
print('Test loss:', score[0])
print('Test accuracy:', score[1])
```

Test accuracy: 0.81

```
Train on 600 samples, validate on 100 samples
Epoch 1/10
600/600 [=====] - 1s 2ms/step - loss: 2.0781 - accuracy: 0.2733 - val_loss: 1.6467 - val_accuracy: 0.4900
Epoch 2/10
600/600 [=====] - 1s 2ms/step - loss: 1.6297 - accuracy: 0.4983 - val_loss: 1.4862 - val_accuracy: 0.4600
Epoch 3/10
600/600 [=====] - 1s 2ms/step - loss: 1.2831 - accuracy: 0.5967 - val_loss: 0.9324 - val_accuracy: 0.6700
Epoch 4/10
600/600 [=====] - 1s 2ms/step - loss: 0.8546 - accuracy: 0.7250 - val_loss: 0.7551 - val_accuracy: 0.7800
Epoch 5/10
600/600 [=====] - 1s 2ms/step - loss: 0.6299 - accuracy: 0.8000 - val_loss: 0.5108 - val_accuracy: 0.8100
Epoch 6/10
600/600 [=====] - 1s 2ms/step - loss: 0.5073 - accuracy: 0.8367 - val_loss: 0.4397 - val_accuracy: 0.8700
Epoch 7/10
600/600 [=====] - 1s 2ms/step - loss: 0.4291 - accuracy: 0.8700 - val_loss: 0.4042 - val_accuracy: 0.8700
Epoch 8/10
600/600 [=====] - 1s 2ms/step - loss: 0.3493 - accuracy: 0.8783 - val_loss: 0.3459 - val_accuracy: 0.8500
Epoch 9/10
600/600 [=====] - 1s 2ms/step - loss: 0.3295 - accuracy: 0.8950 - val_loss: 0.4374 - val_accuracy: 0.8500
Epoch 10/10
600/600 [=====] - 1s 2ms/step - loss: 0.2969 - accuracy: 0.9083 - val_loss: 0.4780 - val_accuracy: 0.8100
Test loss: 0.4779697322845459
Test accuracy: 0.8100000023841858
```

Convolutional Neural Network on MNIST

How to improve performance?

A. Different architecture

B. Tweak hyperparameter

C. Adjust loss function

```
[15]: '''Trains a simple convnet on the MNIST dataset.
Gets to 99.25% test accuracy after 12 epochs
(there is still a lot of margin for parameter tuning).
16 seconds per epoch on a GRID K520 GPU.
'''

from __future__ import print_function
import keras
from keras.datasets import mnist
from keras.models import Sequential
from keras.layers import Dense, Dropout, Flatten
from keras.layers import Conv2D, MaxPooling2D
from keras import backend as K

batch_size = 128
num_classes = 10
epochs = 10

# input image dimensions
img_rows, img_cols = 28, 28

# the data, split between train and test sets
(x_train, y_train), (x_test, y_test) = mnist.load_data()

if K.image_data_format() == 'channels_first':
    x_train = x_train.reshape(x_train.shape[0], 1, img_rows, img_cols)
    x_test = x_test.reshape(x_test.shape[0], 1, img_rows, img_cols)
    input_shape = (1, img_rows, img_cols)
else:
    x_train = x_train.reshape(x_train.shape[0], img_rows, img_cols, 1)
    x_test = x_test.reshape(x_test.shape[0], img_rows, img_cols, 1)
    input_shape = (img_rows, img_cols, 1)
```

```
[16]: x_train = x_train.astype('float32')
x_test = x_test.astype('float32')
x_train /= 255
x_test /= 255
x_train = x_train[0:600]
x_test = x_test[0:100]
y_train = y_train[0:600]
y_test = y_test[0:100]
# print('x_train shape:', x_train.shape)
# print('x_test shape:', x_test.shape)
# print('y_train shape:', y_train.shape)
# print('y_test shape:', y_test.shape)
```

```
[17]: # convert classes to zero-indexed
y_train = keras.utils.to_categorical(y_train, num_classes)
y_test = keras.utils.to_categorical(y_test, num_classes)

model = Sequential()
model.add(Conv2D(32, kernel_size=(3, 3), activation='relu', input_shape=input_shape))
model.add(Conv2D(64, kernel_size=(3, 3), activation='relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.5))
model.add(Flatten())
model.add(Dense(128, activation='relu'))
model.add(Dropout(0.5))
model.add(Dense(num_classes, activation='softmax'))

model.compile(loss=keras.losses.categorical_crossentropy,
              optimizer=keras.optimizers.Adadelta(),
              metrics=['accuracy'])

model.fit(x_train, y_train,
        batch_size=batch_size,
        epochs=epochs,
        verbose=1,
        validation_data=(x_test, y_test))
score = model.evaluate(x_test, y_test, verbose=0)
print('Test loss:', score[0])
print('Test accuracy:', score[1])
```

Test accuracy: 0.81

```
Train on 600 samples, validate on 100 samples
Epoch 1/10
600/600 [=====] - 1s 2ms/step - loss: 2.0781 - accuracy: 0.2733 - val_loss: 1.6467 - val_accuracy: 0.4900
Epoch 2/10
600/600 [=====] - 1s 2ms/step - loss: 1.6297 - accuracy: 0.4983 - val_loss: 1.4862 - val_accuracy: 0.4600
Epoch 3/10
600/600 [=====] - 1s 2ms/step - loss: 1.2831 - accuracy: 0.5967 - val_loss: 0.9324 - val_accuracy: 0.6700
Epoch 4/10
600/600 [=====] - 1s 2ms/step - loss: 0.8546 - accuracy: 0.7250 - val_loss: 0.7551 - val_accuracy: 0.7800
Epoch 5/10
600/600 [=====] - 1s 2ms/step - loss: 0.6299 - accuracy: 0.8000 - val_loss: 0.5108 - val_accuracy: 0.8100
Epoch 6/10
600/600 [=====] - 1s 2ms/step - loss: 0.5073 - accuracy: 0.8367 - val_loss: 0.4397 - val_accuracy: 0.8700
Epoch 7/10
600/600 [=====] - 1s 2ms/step - loss: 0.4291 - accuracy: 0.8700 - val_loss: 0.4042 - val_accuracy: 0.8700
Epoch 8/10
600/600 [=====] - 1s 2ms/step - loss: 0.3493 - accuracy: 0.8783 - val_loss: 0.3459 - val_accuracy: 0.8500
Epoch 9/10
600/600 [=====] - 1s 2ms/step - loss: 0.3295 - accuracy: 0.8950 - val_loss: 0.4374 - val_accuracy: 0.8500
Epoch 10/10
600/600 [=====] - 1s 2ms/step - loss: 0.2969 - accuracy: 0.9083 - val_loss: 0.4780 - val_accuracy: 0.8100
Test loss: 0.4779697322845459
Test accuracy: 0.8100000023841858
```

Convolutional Neural Network on MNIST

How to improve performance?

A. Different architecture

B. Tweak hyperparameter

C. Adjust loss function

D. Get an ML PhD

```
[15]: '''Trains a simple convnet on the MNIST dataset.
Gets to 99.25% test accuracy after 12 epochs
(there is still a lot of margin for parameter tuning).
16 seconds per epoch on a GRID K520 GPU.
'''

from __future__ import print_function
import keras
from keras.datasets import mnist
from keras.models import Sequential
from keras.layers import Dense, Dropout, Flatten
from keras.layers import Conv2D, MaxPooling2D
from keras import backend as K

batch_size = 128
num_classes = 10
epochs = 10

# input image dimensions
img_rows, img_cols = 28, 28

# the data, split between train and test sets
(x_train, y_train), (x_test, y_test) = mnist.load_data()

if K.image_data_format() == 'channels_first':
    x_train = x_train.reshape(x_train.shape[0], 1, img_rows, img_cols)
    x_test = x_test.reshape(x_test.shape[0], 1, img_rows, img_cols)
    input_shape = (1, img_rows, img_cols)
else:
    x_train = x_train.reshape(x_train.shape[0], img_rows, img_cols, 1)
    x_test = x_test.reshape(x_test.shape[0], img_rows, img_cols, 1)
    input_shape = (img_rows, img_cols, 1)
```

```
[16]: x_train = x_train.astype('float32')
x_test = x_test.astype('float32')
x_train /= 255
x_test /= 255
x_train = x_train[0:600]
x_test = x_test[0:100]
y_train = y_train[0:600]
y_test = y_test[0:100]
# print('x_train shape:', x_train.shape)
# print('x_test shape:', x_test.shape)
# print('y_train shape:', y_train.shape)
# print('y_test shape:', y_test.shape)
```

```
[17]: # convert class vectors to binary class matrices
y_train = keras.utils.to_categorical(y_train, num_classes)
y_test = keras.utils.to_categorical(y_test, num_classes)

model = Sequential()
model.add(Conv2D(32, kernel_size=(3, 3), activation='relu', input_shape=input_shape))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Conv2D(64, kernel_size=(3, 3), activation='relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Flatten())
model.add(Dense(128, activation='relu'))
model.add(Dropout(0.5))
model.add(Dense(num_classes, activation='softmax'))

model.compile(loss=keras.losses.categorical_crossentropy,
              optimizer=keras.optimizers.Adadelta(),
              metrics=['accuracy'])

model.fit(x_train, y_train,
        batch_size=batch_size,
        epochs=epochs,
        verbose=1,
        validation_data=(x_test, y_test))
score = model.evaluate(x_test, y_test, verbose=0)
print('Test loss:', score[0])
print('Test accuracy:', score[1])
```

Test accuracy: 0.81

```
Train on 600 samples, validate on 100 samples
Epoch 1/10
600/600 [=====] - 1s 2ms/step - loss: 2.0781 - accuracy: 0.2733 - val_loss: 1.6467 - val_accuracy: 0.4900
Epoch 2/10
600/600 [=====] - 1s 2ms/step - loss: 1.6297 - accuracy: 0.4983 - val_loss: 1.4862 - val_accuracy: 0.4600
Epoch 3/10
600/600 [=====] - 1s 2ms/step - loss: 1.2831 - accuracy: 0.5967 - val_loss: 0.9324 - val_accuracy: 0.6700
Epoch 4/10
600/600 [=====] - 1s 2ms/step - loss: 0.8546 - accuracy: 0.7250 - val_loss: 0.7551 - val_accuracy: 0.7800
Epoch 5/10
600/600 [=====] - 1s 2ms/step - loss: 0.6299 - accuracy: 0.8000 - val_loss: 0.5108 - val_accuracy: 0.8100
Epoch 6/10
600/600 [=====] - 1s 2ms/step - loss: 0.5073 - accuracy: 0.8367 - val_loss: 0.4397 - val_accuracy: 0.8700
Epoch 7/10
600/600 [=====] - 1s 2ms/step - loss: 0.4291 - accuracy: 0.8700 - val_loss: 0.4042 - val_accuracy: 0.8700
Epoch 8/10
600/600 [=====] - 1s 2ms/step - loss: 0.3493 - accuracy: 0.8783 - val_loss: 0.3459 - val_accuracy: 0.8500
Epoch 9/10
600/600 [=====] - 1s 2ms/step - loss: 0.3295 - accuracy: 0.8950 - val_loss: 0.4374 - val_accuracy: 0.8500
Epoch 10/10
600/600 [=====] - 1s 2ms/step - loss: 0.2969 - accuracy: 0.9083 - val_loss: 0.4780 - val_accuracy: 0.8100
Test loss: 0.4779697322845459
Test accuracy: 0.8100000023841858
```

Convolutional Neural Network on MNIST

How to improve performance?

A. Different architecture

B. Tweak hyperparameter

C. Adjust loss function

D. Get an ML PhD

Add more data!

```
[15]: '''Trains a simple convnet on the MNIST dataset.
Gets to 99.25% test accuracy after 12 epochs
(there is still a lot of margin for parameter tuning).
16 seconds per epoch on a GRID K520 GPU.
'''

from __future__ import print_function
import keras
from keras.datasets import mnist
from keras.models import Sequential
from keras.layers import Dense, Dropout, Flatten
from keras.layers import Conv2D, MaxPooling2D
from keras import backend as K

batch_size = 128
num_classes = 10
epochs = 10

# input image dimensions
img_rows, img_cols = 28, 28

# the data, split between train and test sets
(x_train, y_train), (x_test, y_test) = mnist.load_data()

if K.image_data_format() == 'channels_first':
    x_train = x_train.reshape(x_train.shape[0], 1, img_rows, img_cols)
    x_test = x_test.reshape(x_test.shape[0], 1, img_rows, img_cols)
    input_shape = (1, img_rows, img_cols)
else:
    x_train = x_train.reshape(x_train.shape[0], img_rows, img_cols, 1)
    x_test = x_test.reshape(x_test.shape[0], img_rows, img_cols, 1)
    input_shape = (img_rows, img_cols, 1)
```

```
[16]: x_train = x_train.astype('float32')
x_test = x_test.astype('float32')
x_train /= 255
x_test /= 255
x_train = x_train[0:600]
x_test = x_test[0:100]
y_train = y_train[0:600]
y_test = y_test[0:100]
# print('x_train shape:', x_train.shape)
# print('x_test shape:', x_test.shape)
# print('y_train shape:', y_train.shape)
# print('y_test shape:', y_test.shape)
```

```
[17]: # convert class labels to binary vectors
y_train = keras.utils.to_categorical(y_train, num_classes)
y_test = keras.utils.to_categorical(y_test, num_classes)

model = Sequential()
model.add(Conv2D(32, kernel_size=(3, 3), activation='relu', input_shape=input_shape))
model.add(Conv2D(64, kernel_size=(3, 3), activation='relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.5))
model.add(Flatten())
model.add(Dense(128, activation='relu'))
model.add(Dropout(0.5))
model.add(Dense(num_classes, activation='softmax'))

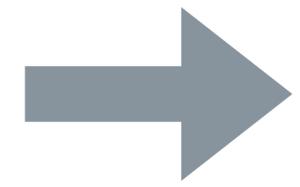
model.compile(loss=keras.losses.categorical_crossentropy,
              optimizer=keras.optimizers.Adadelta(),
              metrics=['accuracy'])

model.fit(x_train, y_train,
        batch_size=batch_size,
        epochs=epochs,
        verbose=1,
        validation_data=(x_test, y_test))
score = model.evaluate(x_test, y_test, verbose=0)
print('Test loss:', score[0])
print('Test accuracy:', score[1])
```

Test accuracy: 0.81

```
Train on 600 samples, validate on 100 samples
Epoch 1/10
600/600 [=====] - 1s 2ms/step - loss: 2.0781 - accuracy: 0.2733 - val_loss: 1.6467 - val_accuracy: 0.4900
Epoch 2/10
600/600 [=====] - 1s 2ms/step - loss: 1.6297 - accuracy: 0.4983 - val_loss: 1.4862 - val_accuracy: 0.4600
Epoch 3/10
600/600 [=====] - 1s 2ms/step - loss: 1.2831 - accuracy: 0.5967 - val_loss: 0.9324 - val_accuracy: 0.6700
Epoch 4/10
600/600 [=====] - 1s 2ms/step - loss: 0.8546 - accuracy: 0.7250 - val_loss: 0.7551 - val_accuracy: 0.7800
Epoch 5/10
600/600 [=====] - 1s 2ms/step - loss: 0.6299 - accuracy: 0.8000 - val_loss: 0.5108 - val_accuracy: 0.8100
Epoch 6/10
600/600 [=====] - 1s 2ms/step - loss: 0.5073 - accuracy: 0.8367 - val_loss: 0.4397 - val_accuracy: 0.8700
Epoch 7/10
600/600 [=====] - 1s 2ms/step - loss: 0.4291 - accuracy: 0.8700 - val_loss: 0.4042 - val_accuracy: 0.8700
Epoch 8/10
600/600 [=====] - 1s 2ms/step - loss: 0.3493 - accuracy: 0.8783 - val_loss: 0.3459 - val_accuracy: 0.8500
Epoch 9/10
600/600 [=====] - 1s 2ms/step - loss: 0.3295 - accuracy: 0.8950 - val_loss: 0.4374 - val_accuracy: 0.8500
Epoch 10/10
600/600 [=====] - 1s 2ms/step - loss: 0.2969 - accuracy: 0.9083 - val_loss: 0.4780 - val_accuracy: 0.8100
Test loss: 0.4779697322845459
Test accuracy: 0.810000023841858
```

Add 10x data



```
[12]: '''Trains a simple convnet on the MNIST dataset.
Gets to 99.25% test accuracy after 12 epochs
(there is still a lot of margin for parameter tuning).
16 seconds per epoch on a GRID K520 GPU.
'''

from __future__ import print_function
import keras
from keras.datasets import mnist
from keras.models import Sequential
from keras.layers import Dense, Dropout, Flatten
from keras.layers import Conv2D, MaxPooling2D
from keras import backend as K

batch_size = 128
num_classes = 10
epochs = 10

# input image dimensions
img_rows, img_cols = 28, 28

# the data, split between train and test sets
(x_train, y_train), (x_test, y_test) = mnist.load_data()

if K.image_data_format() == 'channels_first':
    x_train = x_train.reshape(x_train.shape[0], 1, img_rows, img_cols)
    x_test = x_test.reshape(x_test.shape[0], 1, img_rows, img_cols)
    input_shape = (1, img_rows, img_cols)
else:
    x_train = x_train.reshape(x_train.shape[0], img_rows, img_cols, 1)
    x_test = x_test.reshape(x_test.shape[0], img_rows, img_cols, 1)
    input_shape = (img_rows, img_cols, 1)
```

```
[13]: x_train = x_train.astype('float32')
x_test = x_test.astype('float32')
x_train /= 255
x_test /= 255
# x_train = x_train[0:600]
# x_test = x_test[0:100]
# y_train = y_train[0:600]
# y_test = y_test[0:100]
# print('x_train shape:', x_train.shape)
# print('x_test shape:', x_test.shape)
# print('y_train shape:', y_train.shape)
# print('y_test shape:', y_test.shape)
```

```
[14]: # convert class labels to binary vectors
y_train = keras.utils.to_categorical(y_train, num_classes)
y_test = keras.utils.to_categorical(y_test, num_classes)

model = Sequential()
model.add(Conv2D(32, kernel_size=(3, 3), activation='relu', input_shape=input_shape))
model.add(Conv2D(64, kernel_size=(3, 3), activation='relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.5))
model.add(Flatten())
model.add(Dense(128, activation='relu'))
model.add(Dropout(0.5))
model.add(Dense(num_classes, activation='softmax'))

model.compile(loss=keras.losses.categorical_crossentropy,
              optimizer=keras.optimizers.Adadelta(),
              metrics=['accuracy'])

model.fit(x_train, y_train,
        batch_size=batch_size,
        epochs=epochs,
        verbose=1,
        validation_data=(x_test, y_test))
score = model.evaluate(x_test, y_test, verbose=0)
print('Test loss:', score[0])
print('Test accuracy:', score[1])
```

Test accuracy: 0.99

```
Train on 60000 samples, validate on 10000 samples
Epoch 1/10
60000/60000 [=====] - 97s 2ms/step - loss: 0.2591 - accuracy: 0.9206 - val_loss: 0.0602 - val_accuracy: 0.9798
Epoch 2/10
60000/60000 [=====] - 98s 2ms/step - loss: 0.0918 - accuracy: 0.9728 - val_loss: 0.0395 - val_accuracy: 0.9870
Epoch 3/10
60000/60000 [=====] - 96s 2ms/step - loss: 0.0660 - accuracy: 0.9801 - val_loss: 0.0394 - val_accuracy: 0.9868
Epoch 4/10
60000/60000 [=====] - 94s 2ms/step - loss: 0.0566 - accuracy: 0.9830 - val_loss: 0.0316 - val_accuracy: 0.9894
Epoch 5/10
60000/60000 [=====] - 107s 2ms/step - loss: 0.0475 - accuracy: 0.9853 - val_loss: 0.0382 - val_accuracy: 0.9875
Epoch 6/10
60000/60000 [=====] - 113s 2ms/step - loss: 0.0426 - accuracy: 0.9872 - val_loss: 0.0286 - val_accuracy: 0.9909
Epoch 7/10
60000/60000 [=====] - 109s 2ms/step - loss: 0.0376 - accuracy: 0.9887 - val_loss: 0.0294 - val_accuracy: 0.9898
Epoch 8/10
60000/60000 [=====] - 92s 2ms/step - loss: 0.0350 - accuracy: 0.9893 - val_loss: 0.0307 - val_accuracy: 0.9903
Epoch 9/10
60000/60000 [=====] - 99s 2ms/step - loss: 0.0319 - accuracy: 0.9901 - val_loss: 0.0266 - val_accuracy: 0.9908
Epoch 10/10
60000/60000 [=====] - 96s 2ms/step - loss: 0.0293 - accuracy: 0.9908 - val_loss: 0.0264 - val_accuracy: 0.9919
Test loss: 0.02642502591495936
Test accuracy: 0.9919000267982483
```





Surpassing the state of the art on ImageNet by collecting more labels

(Submitted on 2020)

We achieve state-of-the-art 99.5% top-1 accuracy on ImageNet using a ResNet-50. This was achieved by paying people money to clean and grow the training set. First we cleaned up the incorrect labels in the existing training set and validation set. Then we found more unlabeled images similar to the high loss images in the validation set, labeled them, and added them to the training set. We repeated this process until accuracy improved enough. Data for this paper will be made available.

Subjects: **Machine Learning (cs.LG)**; Computer Vision and Pattern Recognition (cs.CV); Machine Learning (stat.ML)

Download:

- [PDF](#)
- [Other formats](#)
(license)

Current browse context:

cs.LG

[< prev](#) | [next >](#)
[new](#) | [recent](#) | [2002](#)

Change to browse by:

cs

Surpassing the state of the art on ImageNet by collecting more labels

(Submitted on 2020)

“clean and grow the training set”

ImageNet using a ResNet-50. This was achieved by paying people money to correct labels in the existing training set and validation set. Then we found the validation set, labeled them, and added them to the training set. We

“repeated this process until accuracy improved enough”

Download:

- [PDF](#)
- [Other formats](#)
(license)

Current browse context:

cs.LG

[< prev](#) | [next >](#)
[new](#) | [recent](#) | [2002](#)

Change to browse by:

cs

Surpassing the state of the art on ImageNet by collecting more labels

(Submitted on 2020)

"clean and grow the training set"

ImageNet using a ResNet-50. This was achieved by paying people money to correct labels in the existing training set and validation set. Then we found the validation set, labeled them, and added them to the training set. We

"repeated this process until accuracy improved enough"

Download:

- [PDF](#)
- [Other formats](#)
(license)

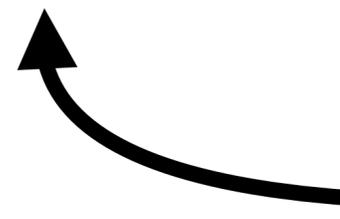
Current browse context:

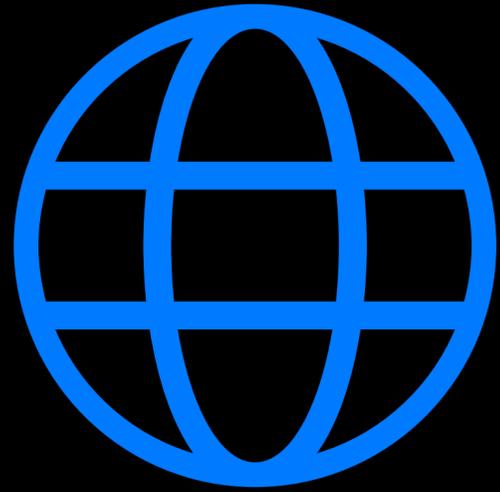
cs.LG

[< prev](#) | [next >](#)
[new](#) | [recent](#) | [2002](#)

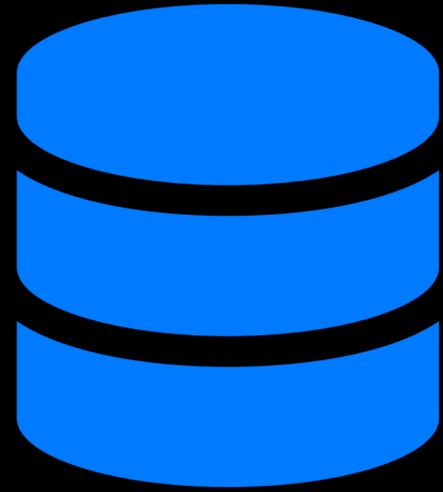
Change to browse by:

[cs](#)

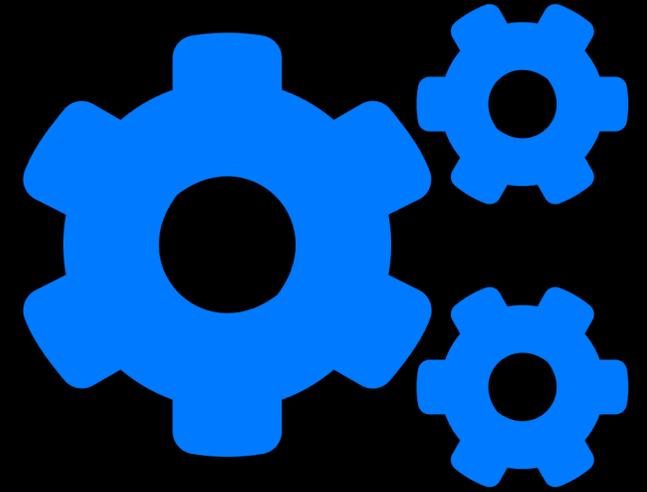
 This is a **data iteration!**



World



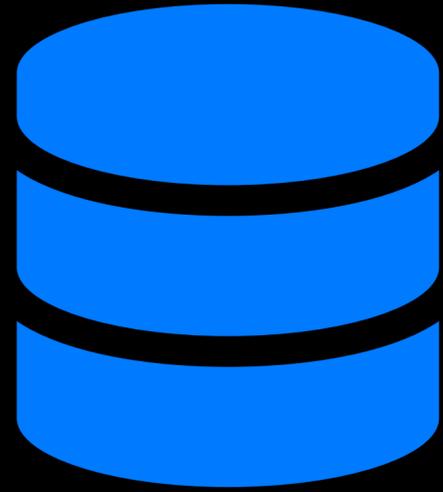
Data



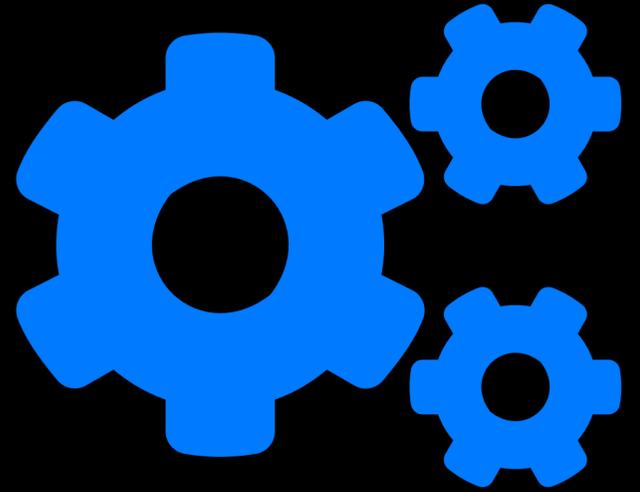
Model



World

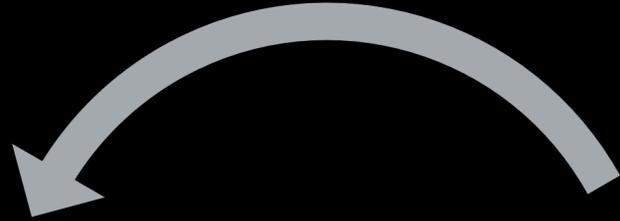


Data



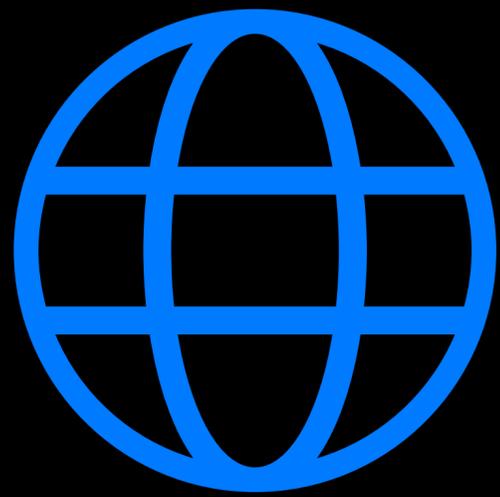
Model

Model Iteration

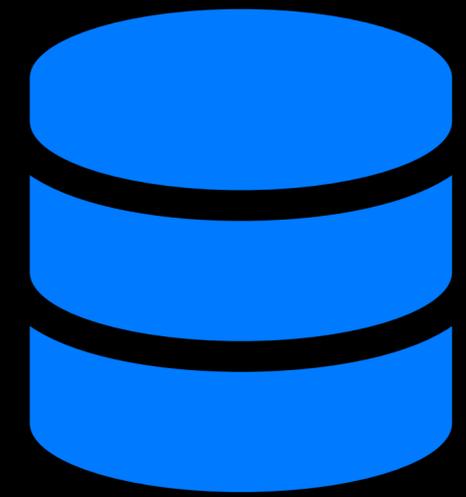


★ **Data Iteration**

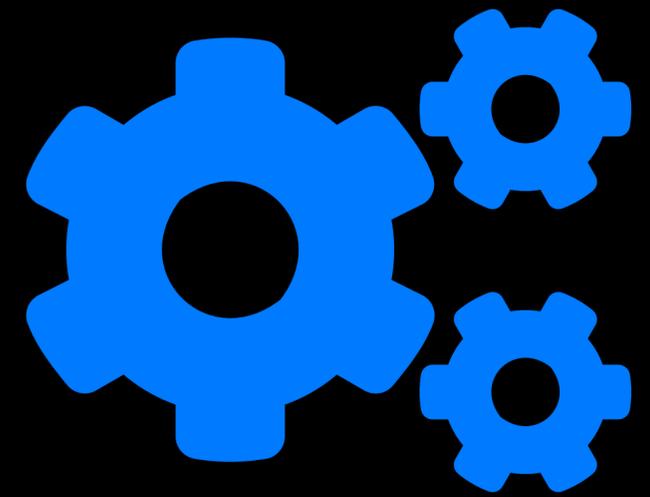
Model Iteration



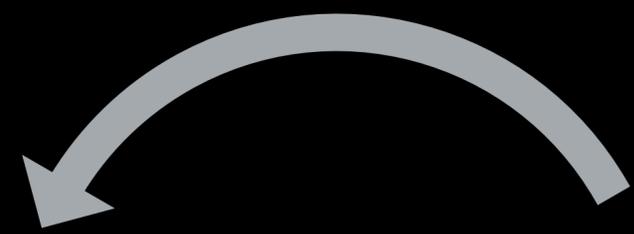
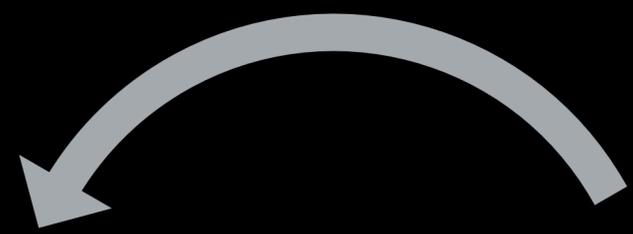
World



Data



Model



Understanding and Visualization Data Iteration

Contributions

- Identify common data iterations and challenges through practitioner interviews at Apple
- **CHAMELEON**: Interactive visualization for data iteration
- Case studies on real datasets

The screenshot displays the Chameleon interface, which is designed for visualizing data iteration. At the top, a horizontal timeline shows versions from 1.0 to 1.18, with version 1.18 selected. A sidebar (labeled 'B') provides dataset information for 'evolving-data-name', showing 19 versions and a table of version, date, instances, train accuracy, and test accuracy. The main area is divided into three sections: 'Feature' (labeled 'C'), 'Train Distribution', and 'Test Distribution'. The 'Feature' section shows three features (feature_1, feature_2, feature_3) with their respective statistics (Min, Max, Mean) for versions 1.13 and 1.12. The 'Train Distribution' and 'Test Distribution' sections show histograms of the data distributions. An 'Aggregated Embedding' section at the bottom shows a heatmap of the data. A 'Data Version Timeline' (labeled 'A') is also visible on the right side of the interface.

Dataset Info

Version	Date	Instances	Train Acc.	Test Acc.
1.13	1/11/2019	53,174	0.841	0.839
1.12	1/8/2019	49,282	0.841	0.837

Feature

Feature	1.13	1.13	1.12	1.12
Min	0.00	0.00	0.00	0.00
Max	100.00	100.00	100.00	100.00
Mean	75.65	75.49	75.68	75.80

Train Distribution

Test Distribution

Aggregated Embedding

Data Version Timeline

Feature View

Understanding and Visualizing Data Iteration in Machine Learning

Fred Hohman¹, Kanit Wongsuphasawat², Mary Beth Kery³, Kayur Patel⁴
¹ Georgia Tech, Atlanta, GA, USA, fhohman@gatech.edu
² Apple Inc., Seattle, WA, USA, {kanitw, kayur}@apple.com
³ Apple Inc., Seattle, WA, USA, mkery@cs.cmu.edu
⁴ Carnegie Mellon University, Pittsburgh, PA, USA, mkery@cs.cmu.edu

ABSTRACT
Successful machine learning (ML) applications require iterations on both modeling and the underlying data. While prior visualization tools for ML primarily focus on modeling, our interviews with 23 ML practitioners reveal that they improve model performance frequently by iterating on their data (e.g., collecting new data, adding labels) rather than their models. We also identify common types of data iterations and associated analysis tasks and challenges. To help attribute data iterations to model performance, we design a collection of interactive visualizations and integrate them into a prototype, CHAMELEON, that lets users compare data features, training/testing splits, and performance across data versions. We present two case studies where developers apply CHAMELEON to their own evolving datasets on production ML projects. Our interface helps them verify data collection efforts, find failure cases stretching across data versions, capture data processing changes that impacted performance, and identify opportunities for future data iterations.

Author Keywords
Data iteration, evolving datasets, machine learning iteration, visual analytics, interactive interfaces

CCS Concepts
•Human-centered computing → Visual analytics;
•Computing methodologies → Machine learning;

INTRODUCTION
Successful machine learning (ML) applications require an iterative process to create models that deliver desired performance and user experience [4, 38]. As shown in Figure 1, this process typically involves both model iteration (e.g., searching for better hyperparameters or architectures) and data iteration (e.g., collecting new training data to improve performance). Yet, prior research primarily focuses on model iteration. Machine learning (ML) researchers are rapidly proposing new model architectures for tasks in computer vision and

Figure 1. An overview of a typical machine learning process, which involves both model iteration (e.g., changing model architectures or hyperparameters) and data iteration (e.g., collecting new data to improve model performance). This paper focuses on data iteration as its tooling is underrepresented compared to model iteration.

This primary focus on model iteration makes sense in academic and research settings, where the objective is to build novel model architectures independent of which dataset is used. Yet in practice, the underlying dataset also determines what a model learns—regardless of which model or architecture is chosen. The classic ML colloquialism, “garbage in, garbage out,” evokes this essential fact that data needs to have the appropriate signal for a model to be useful. In real world applications, rarely do teams start out with a dataset that is a high-quality match to their specific ML project goals. Thus data iteration is vital to the success of production ML projects. To create high performance models, developers need to iterate on their data alongside their model architectures. Over the course of a machine learning project, datasets may change jointly alongside models for a variety of reasons. As a project matures, developers may discover use cases underrepresented in their datasets and thus need to collect additional data for such cases. Changes in the world may also affect the distribution of new data. For example, the latest viral video may drive spikes in internet search traffic and change search query distributions. These data changes raise interesting challenges and questions during model development: How does one track,

Permission to make digital or hard copies of part or all of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for third-party components of this work must be honored. For all other uses, contact the owner(s).

CHI '20, April 25–30, 2020, Honolulu, HI, USA
© 2020 copyright held by the author(s).

ACM ISBN 978-1-4503-6708-0/20/04.
DOI: <https://doi.org/10.1145/3311831.3376177>

Interviews to Understand Data Iteration Practice

Participant Information

- Semi-structured interviews with ML researchers, engineers, and managers at Apple
- 23 practitioners across 13 teams

Domain	Specialization	# of people
Computer vision	Large-scale classification, object detection, video analysis, visual search	8
Natural language processing	Text classification, question answering, language understanding	8
Applied ML + Systems	Platform and infrastructure, crowdsourcing, annotation, deployment	5
Sensors	Activity recognition	1

“Most of the time, we improve performance more by adding additional data or cleaning data rather than changing the model [code].”

— Applied ML practitioner in computer vision

Interviews to Understand Data Iteration Practice

Findings Summary

Why do Data Iteration?

- Data improves performance
- Data bootstraps modeling
- The world changes, so must your data

Data Iteration Frequency

- Models: monthly → daily
- Data: monthly → per minute

Entangled Iterations

- Separate model and data iterations to ensure fair comparisons

Interviews to Understand Data Iteration Practice

Common Data Iterations

+ Add sampled instances

Gather more data randomly sampled from population

+ Add specific instances

Gather more data intentionally for specific label or feature range

+ Add synthetic instances

Gather more data by creating synthetic data or augmenting existing data

+ Add labels

Add and enrich instance annotations

- Remove instances

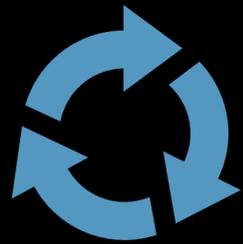
Remove noisy and erroneous outliers

~ Modify features, labels

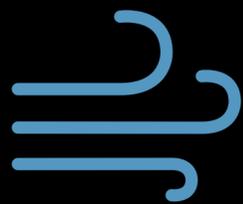
Clean, edit, or fix data

Interviews to Understand Data Iteration Practice

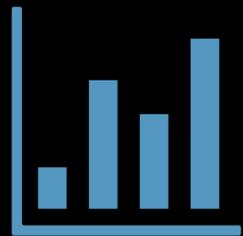
Challenges of Data Iteration



- Tracking experimental and iteration history



- When to “unfreeze” data versions
- When to stop collecting data

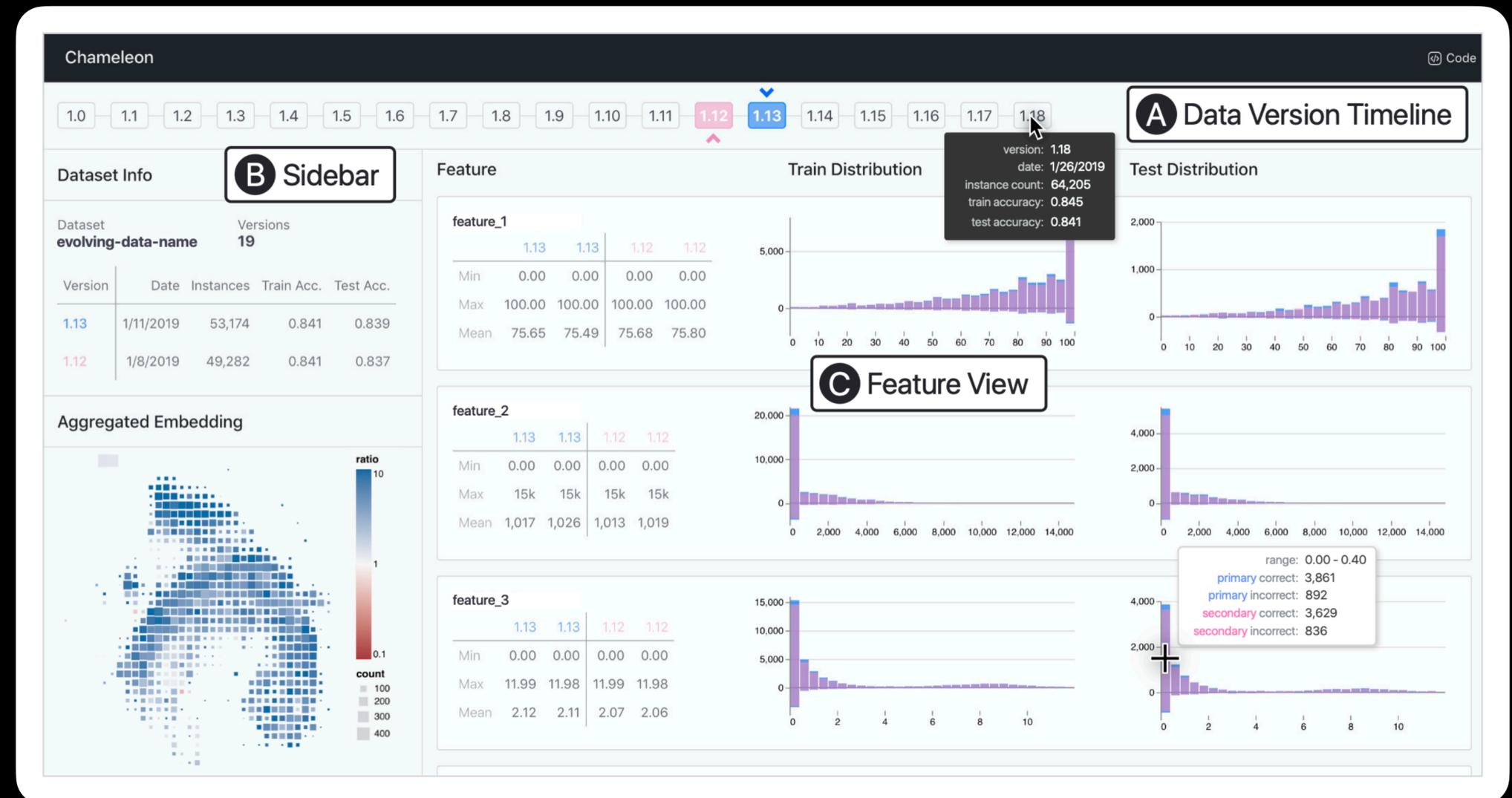


- Manual failure case analysis
- Building data blacklists

CHAMELEON

Understanding and Visualization Data Iteration

- Retroactively track and explore data iterations and metrics over versions
- Attribute model metric change to data iterations
- Understand model sensitivity over data versions

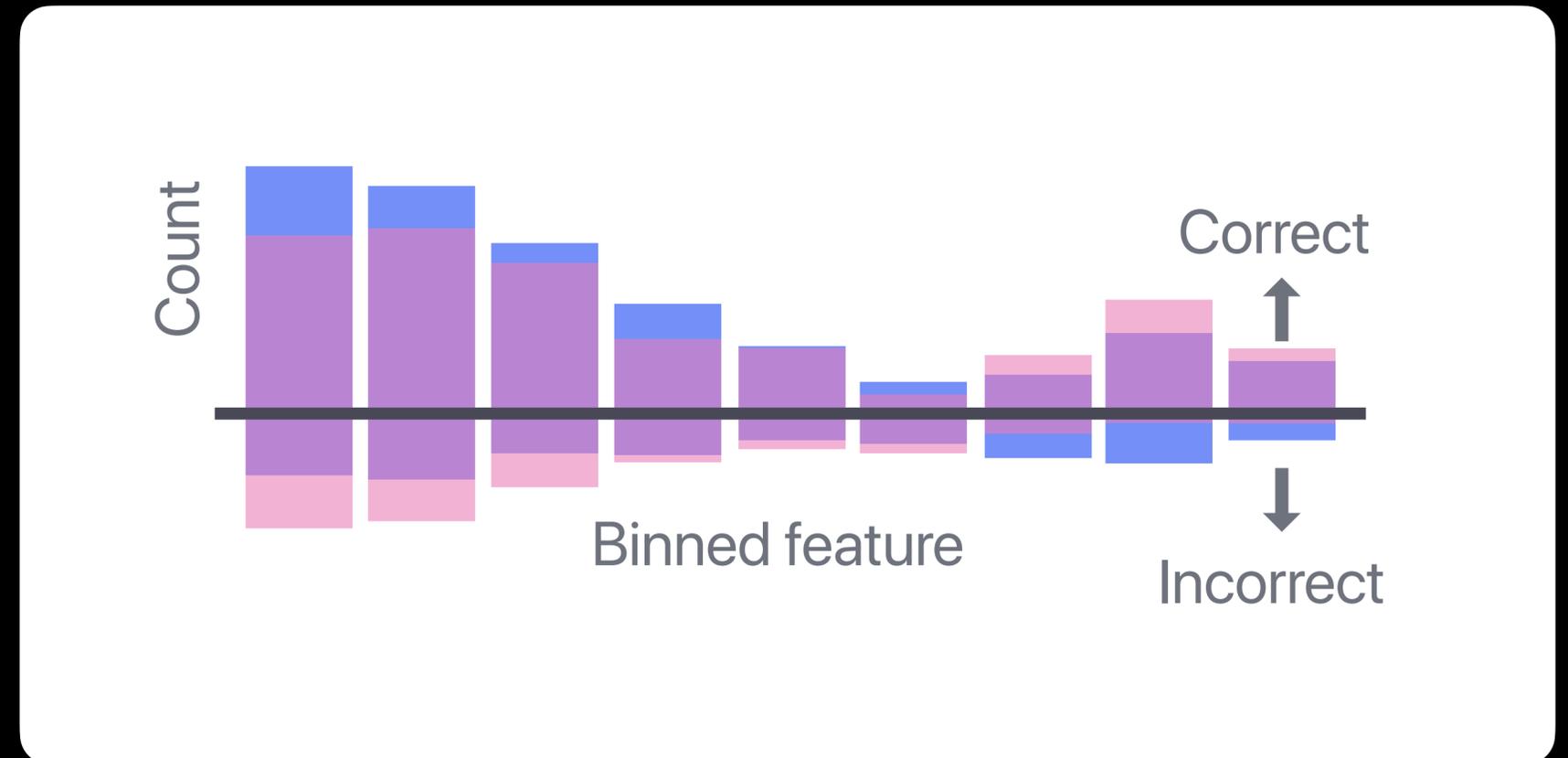


CHAMELEON

Understanding and Visualization Data Iteration

Compare feature distributions by:

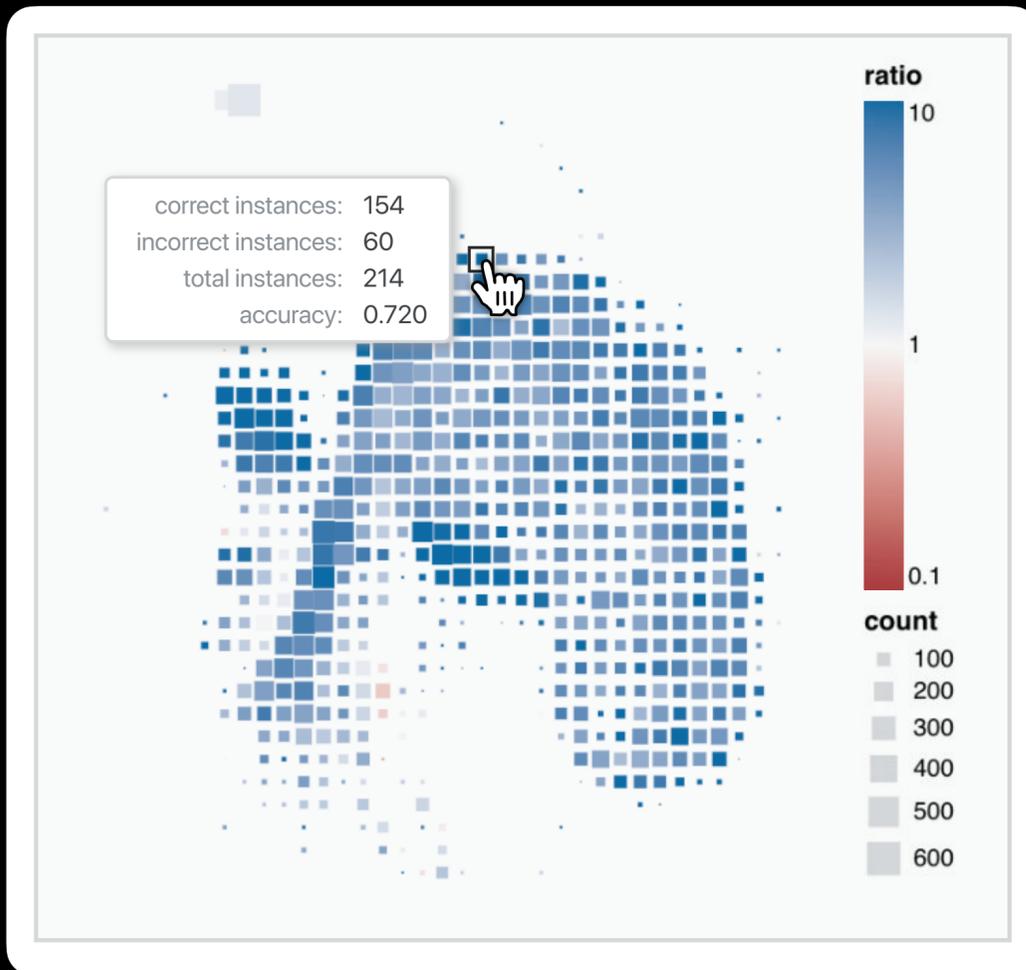
- Training and testing splits
- Performance (e.g., correct v. incorrect predictions)
- Data versions



"Overlaid diverging histogram" per feature

CHAMELEON Visualizations

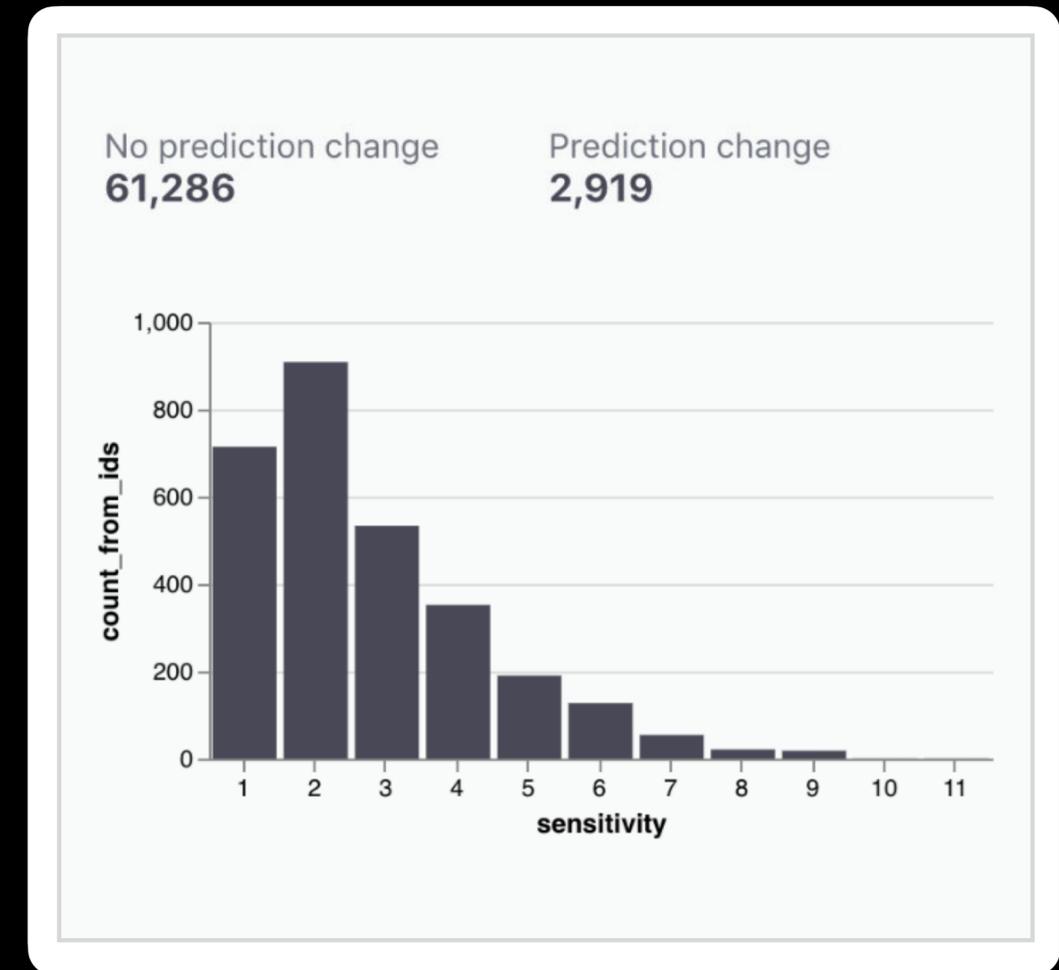
Aggregated Embedding



Prediction Change Matrix



Sensitivity Histogram

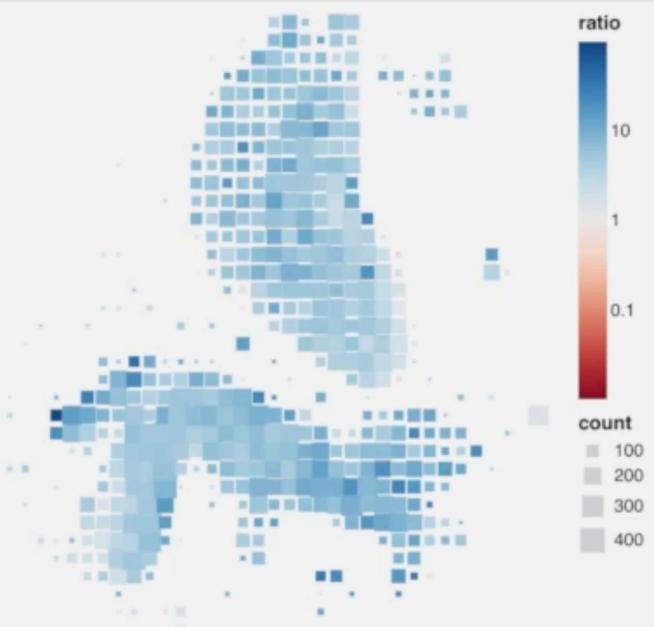


- 1.0
- 1.1
- 1.2
- 1.3
- 1.4
- 1.5
- 1.6
- 1.7
- 1.8
- 1.9
- 1.10
- 1.11
- 1.12
- 1.13
- 1.14
- 1.15
- 1.16
- 1.17
- 1.18

Dataset Info

Dataset		Versions		
Redacted data name		19		
Version	Date	Instances	Train Acc.	Test Acc.
1.18	1/26/2019	64,205	0.845	0.841
1.17	1/23/2019	63,505	0.845	0.840

Aggregated Embedding



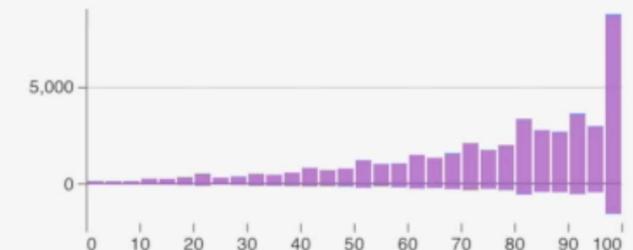
Prediction Change

		Version 18	
		correct	incorrect
Version 17	correct	53,189	402
	incorrect		

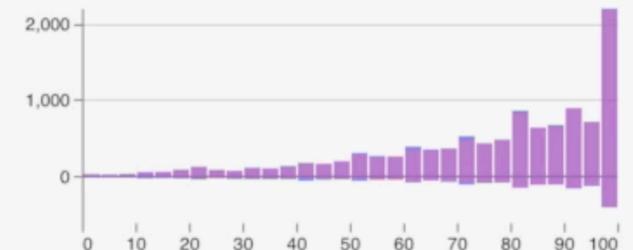
Feature

Redacted feature name	1.18	1.18	1.17	1.17
Min	0.00	0.00	0.00	0.00
Max	100.00	100.00	100.00	100.00
Mean	75.44	75.47	75.50	75.49

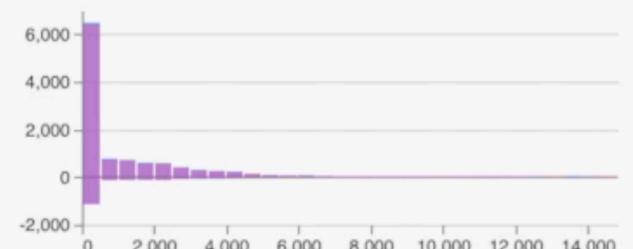
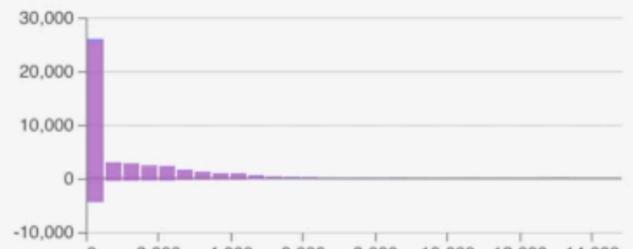
Train Distribution



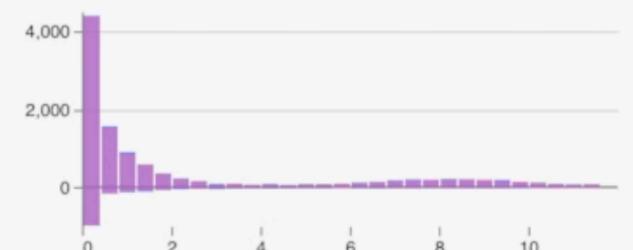
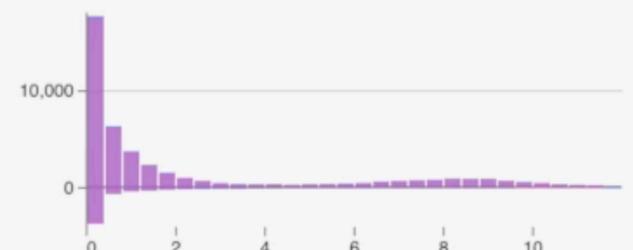
Test Distribution



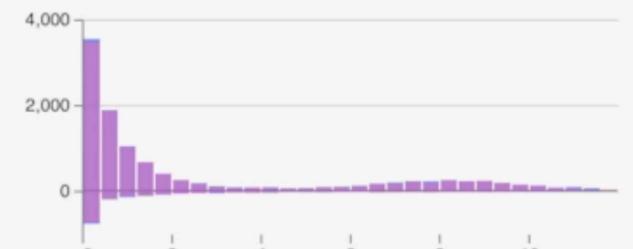
Redacted feature name	1.18	1.18	1.17	1.17
Min	0.00	0.00	0.00	0.00
Max	15k	15k	15k	15k
Mean	1,026	1,031	1,024	1,027



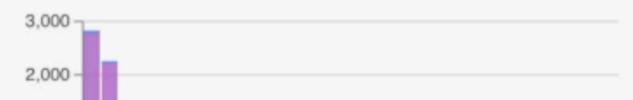
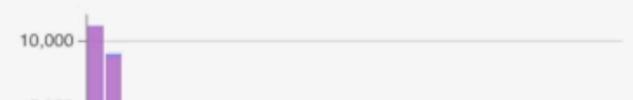
Redacted feature name	1.18	1.18	1.17	1.17
Min	0.00	0.00	0.00	0.00
Max	11.99	11.98	11.99	11.98
Mean	2.21	2.20	2.21	2.20



Redacted feature name	1.18	1.18	1.17	1.17
Min	0.00	0.00	0.00	0.00
Max	11.98	11.89	11.98	11.98
Mean	2.46	2.44	2.45	2.46



Redacted feature name	1.18	1.18	1.17	1.17
Min	0.00	0.00	0.00	0.00
Max	11.98	11.89	11.98	11.98
Mean	2.46	2.44	2.45	2.46

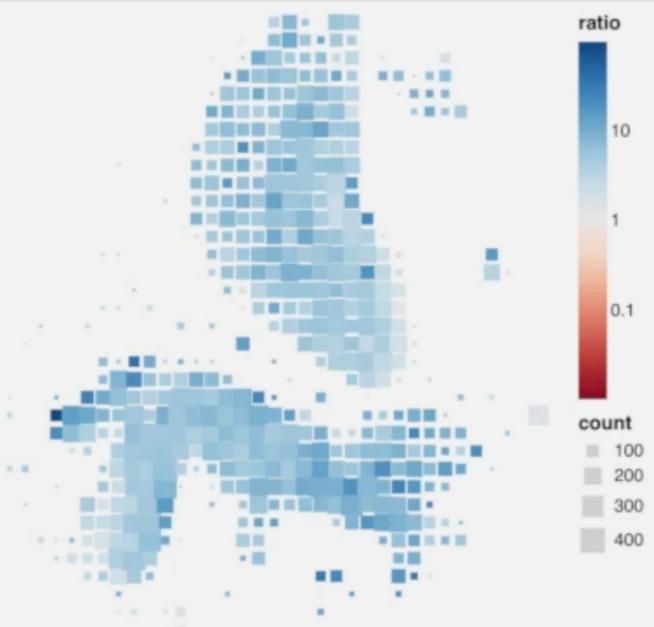


- 1.0
- 1.1
- 1.2
- 1.3
- 1.4
- 1.5
- 1.6
- 1.7
- 1.8
- 1.9
- 1.10
- 1.11
- 1.12
- 1.13
- 1.14
- 1.15
- 1.16
- 1.17
- 1.18

Dataset Info

Dataset		Versions		
Redacted data name		19		
Version	Date	Instances	Train Acc.	Test Acc.
1.18	1/26/2019	64,205	0.845	0.841
1.17	1/23/2019	63,505	0.845	0.840

Aggregated Embedding



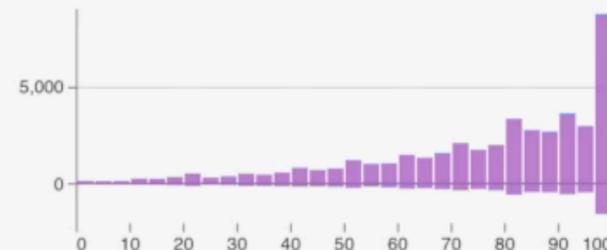
Prediction Change

		Version 18	
		correct	incorrect
Version 17	correct	53,189	402
	incorrect	402	53,189

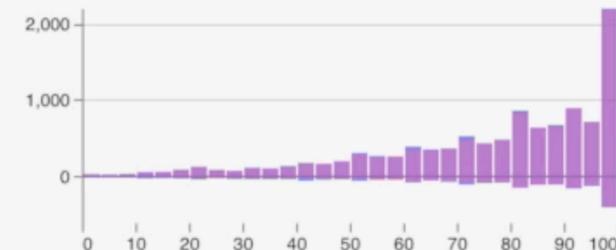
Feature

Redacted feature name	1.18	1.18	1.17	1.17
Min	0.00	0.00	0.00	0.00
Max	100.00	100.00	100.00	100.00
Mean	75.44	75.47	75.50	75.49

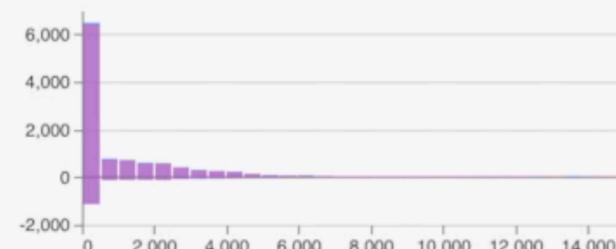
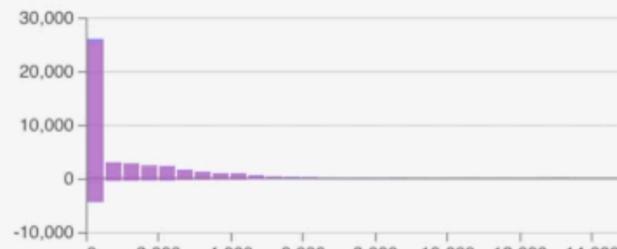
Train Distribution



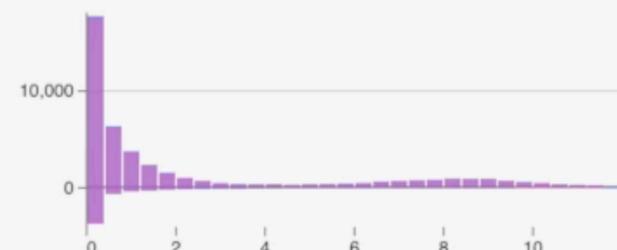
Test Distribution



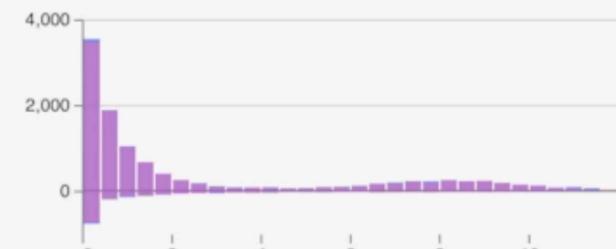
Redacted feature name	1.18	1.18	1.17	1.17
Min	0.00	0.00	0.00	0.00
Max	15k	15k	15k	15k
Mean	1,026	1,031	1,024	1,027



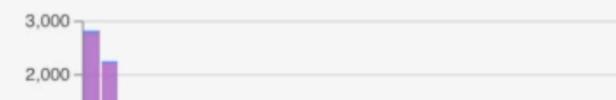
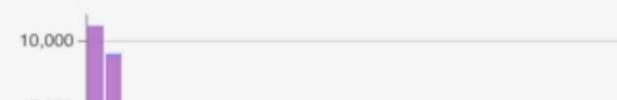
Redacted feature name	1.18	1.18	1.17	1.17
Min	0.00	0.00	0.00	0.00
Max	11.99	11.98	11.99	11.98
Mean	2.21	2.20	2.21	2.20



Redacted feature name	1.18	1.18	1.17	1.17
Min	0.00	0.00	0.00	0.00
Max	11.98	11.89	11.98	11.98
Mean	2.46	2.44	2.45	2.46



Redacted feature name	1.18	1.18	1.17	1.17
Min	0.00	0.00	0.00	0.00
Max	11.98	11.89	11.98	11.98
Mean	2.46	2.44	2.45	2.46

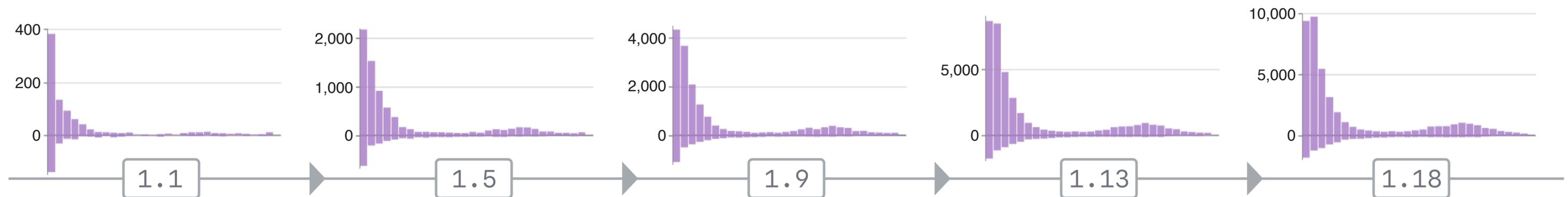


Case Study I

Sensor Prediction

- Visualization challenges prior data collection beliefs
- Finding failure cases
- Interface utility

- 64,502 instances
- Collected over 2 months
- 20 features



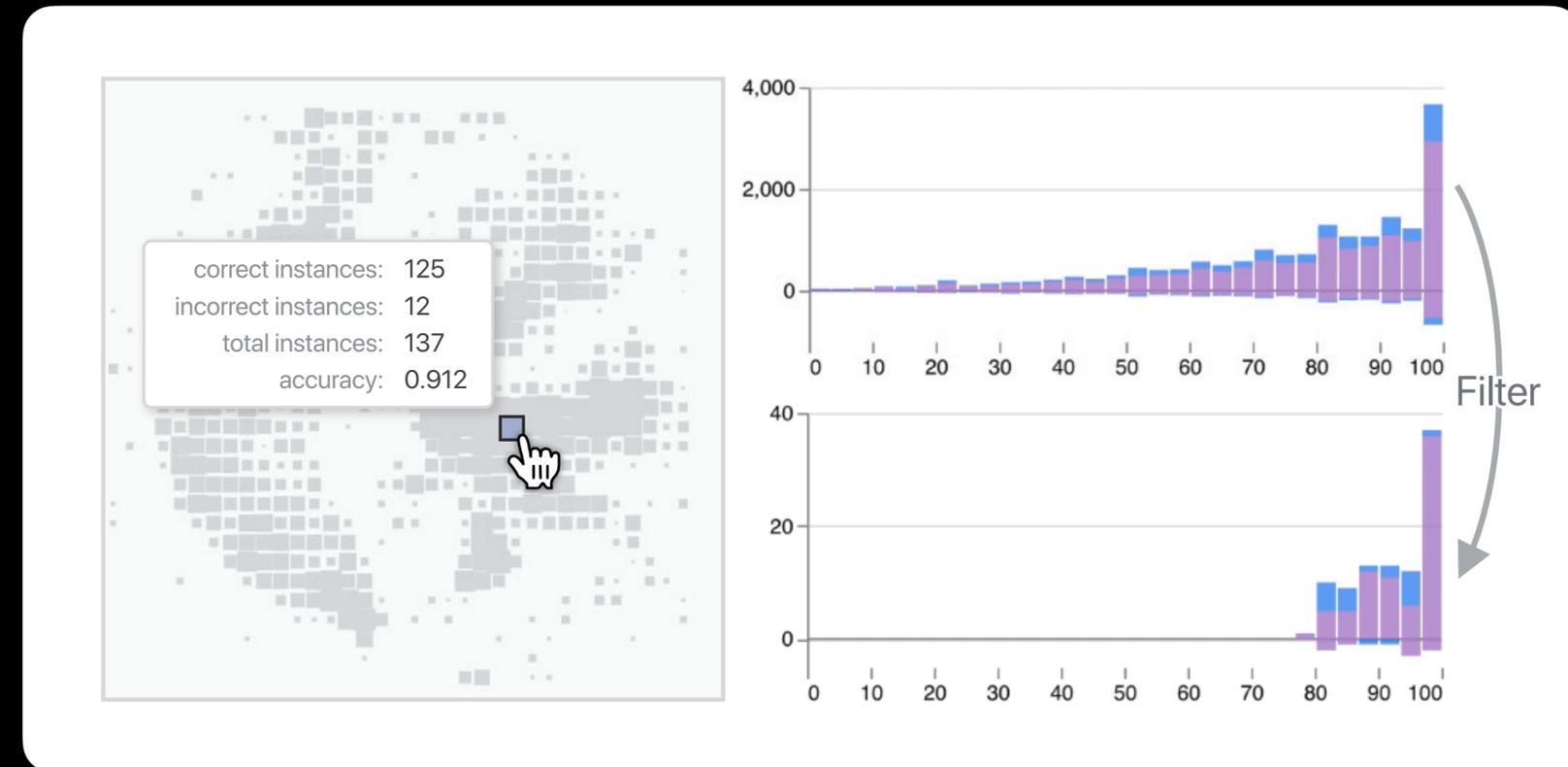
A feature's long-tailed, multi-modal distribution shape solidifies over collection time: 1,442 → 64,205 instances

Case Study II

Learning from Logs

- Inspecting performance across features
- Capturing data processing changes
- Encouraging instance-level analysis

- 48,000 instances
- Collected over 6 months
- 34 features



Filtering across features quickly finds data subsets to compare against global distributions

Opportunities for Future ML Iteration Tools

Opportunities for Future ML Iteration Tools

- Interfaces for both data and model iteration

Opportunities for Future ML Iteration Tools

- Interfaces for both data and model iteration
- Data iteration tooling to help experimental handoff

Opportunities for Future ML Iteration Tools

- Interfaces for both data and model iteration
- Data iteration tooling to help experimental handoff
- Data as a shared connection across user expertise

Opportunities for Future ML Iteration Tools

- Interfaces for both data and model iteration
- Data iteration tooling to help experimental handoff
- Data as a shared connection across user expertise
- Visualizing probabilistic labels from data programming

Opportunities for Future ML Iteration Tools

- Interfaces for both data and model iteration
- Data iteration tooling to help experimental handoff
- Data as a shared connection across user expertise
- Visualizing probabilistic labels from data programming
- Visualizations for other data types



Understanding and Visualizing Data Iteration in Machine Learning

Fred Hohman, Georgia Tech, @fredhohman

Kanit Wongsuphasawat, Apple, @kanitw

Mary Beth Kery, Carnegie Mellon University, @mbkery

Kayur Patel, Apple, @foil

fredhohman.com/papers/chameleon