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

Convolutional Neural Network on MNIST

https://github.com/keras-team/keras/blob/master/examples/mnist_cnn.py

[15]:
```python
'''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]:
```python
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_t
# print(x_tr
# print(x_te
```

[17]:
```python
# convert cl
y_train = ke
y_test = ker

model = Sequ
model.add(Co


model.add(Co
model.add(Ma
model.add(Dr
model.add(Fl
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])
```

```
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
```

Test accuracy: 0.81

Convolutional Neural Network on MNIST

How to improve performance?

https://github.com/keras-team/keras/blob/master/examples/mnist_cnn.py

[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_t
# print(x_tr
# print(x_te
```

[17]:
```
# convert cl
y_train = ke
y_test = ker

model = Sequ
model.add(Co


model.add(Co
model.add(Ma
model.add(Dr
model.add(Fl
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])
```

```
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
```

Test accuracy: 0.81

https://github.com/keras-team/keras/blob/master/examples/mnist_cnn.py

[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_t
# print(x_tr
# print(x_te
```

[17]:
```
# convert cl
y_train = ke
y_test = ker

model = Sequ
model.add(Co


model.add(Co
model.add(Ma
model.add(Dr
model.add(Fl
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])
```

```
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
```

Test accuracy: 0.81

Convolutional Neural Network on MNIST

How to improve performance?

A. Different architecture

https://github.com/keras-team/keras/blob/master/examples/mnist_cnn.py

[15]:
```python
'''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]:
```python
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_t
# print(x_tr
# print(x_te
```

[17]:
```python
# convert cl
y_train = ke
y_test = ker

model = Sequ
model.add(Co


model.add(Co
model.add(Ma
model.add(Dr
model.add(Fl
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])
```

```
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
```

Test accuracy: 0.81

Convolutional Neural Network on MNIST

How to improve performance?

A. Different architecture

B. Tweak hyperparameter

https://github.com/keras-team/keras/blob/master/examples/mnist_cnn.py

```
[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_t
# print(x_tr
# print(x_te
```

```
[17]: # convert cl
y_train = ke
y_test = ker

model = Sequ
model.add(Co


model.add(Co
model.add(Ma
model.add(Dr
model.add(Fl
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])
```

```
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
```

Test accuracy: 0.81

Convolutional Neural Network on MNIST

How to improve performance?

A. Different architecture

B. Tweak hyperparameter

C. Adjust loss function

https://github.com/keras-team/keras/blob/master/examples/mnist_cnn.py

```
[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_t
# print(x_tr
# print(x_te
```

```
[17]: # convert cl
y_train = ke
y_test = ker

model = Sequ
model.add(Co


model.add(Co
model.add(Ma
model.add(Dr
model.add(Fl
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

https://github.com/keras-team/keras/blob/master/examples/mnist_cnn.py

```
[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_t
# print(x_tr
# print(x_te
```

```
[17]: # convert cl
y_train = ke
y_test = ker

model = Sequ
model.add(Co


model.add(Co
model.add(Ma
model.add(Dr
model.add(Fl
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])
```

```
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
```

Test accuracy: 0.81

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!

https://github.com/keras-team/keras/blob/master/examples/mnist_cnn.py

```
[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_t
# print(x_tr
# print(x_te
```

```
[17]: # convert cl
y_train = ke
y_test = ker

model = Sequ
model.add(Co


model.add(Co
model.add(Ma
model.add(Dr
model.add(Fl
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
```

Add 10x data

https://github.com/keras-team/keras/blob/master/examples/mnist_cnn.py

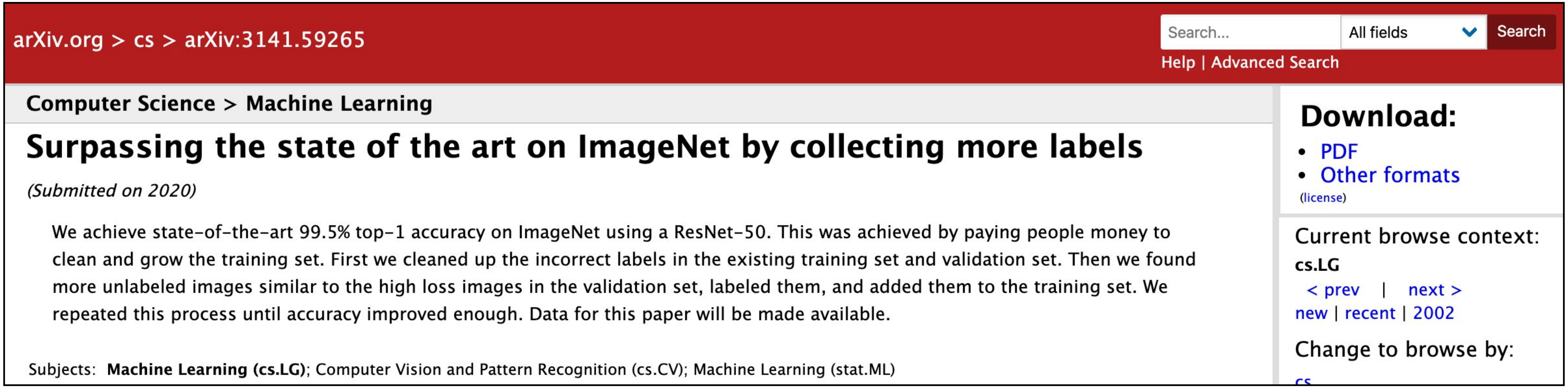
```
[12]: '''Trains a simple convnet on the MNIST dataset.
Gets to 99.25% test accuracy after 12 epochs
(there is still a lot of margin for parameter tuning).
16 seconds per epoch on a GRID K520 GPU.
'''

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_trai
# print(x_train.
# print(x_test.s
```

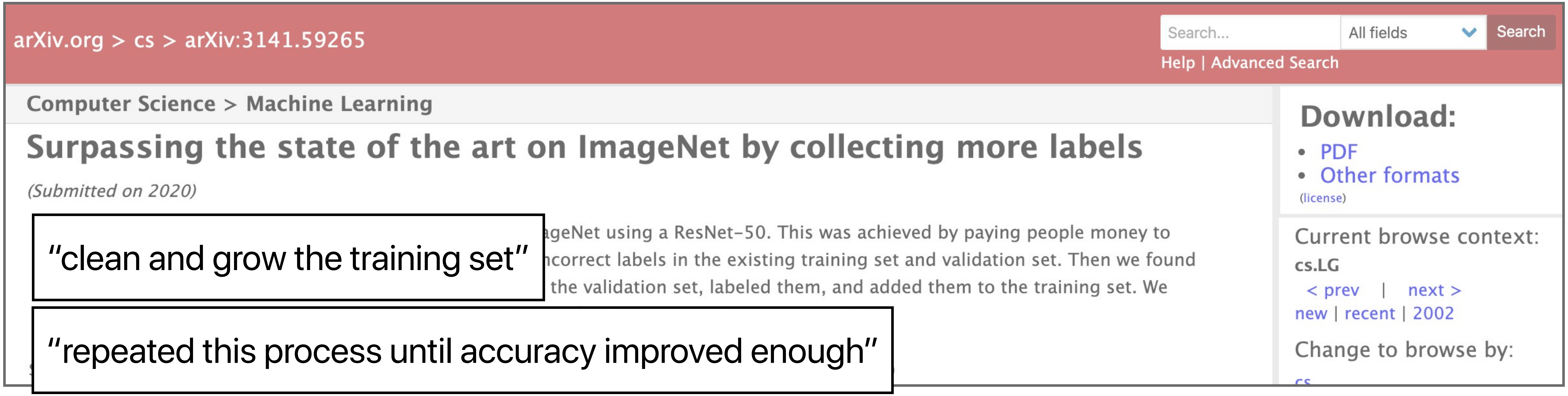
```
[14]: # convert class
y_train = keras.
y_test = keras.u

model = Sequenti
model.add(Conv2D


model.add(Conv2D
model.add(MaxPoo
model.add(Dropou
model.add(Flatte
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

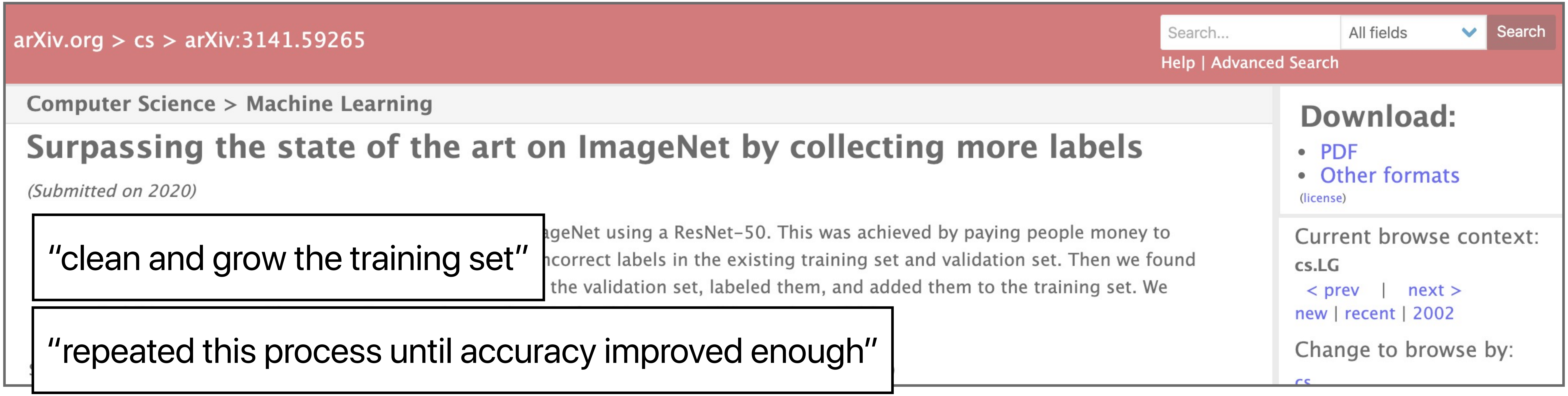
```
Train on 60000 samples, validate on 10000 samples
Epoch 1/10
60000/60000 [==============================] - 97s 2ms/step - loss: 0.2591 - accuracy: 0.9206 - val_loss: 0.0602 - val_accuracy: 0.9798
Epoch 2/10
60000/60000 [==============================] - 98s 2ms/step - loss: 0.0918 - accuracy: 0.9728 - 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
```



https://twitter.com/karpathy/status/1231378194948706306



https://twitter.com/karpathy/status/1231378194948706306



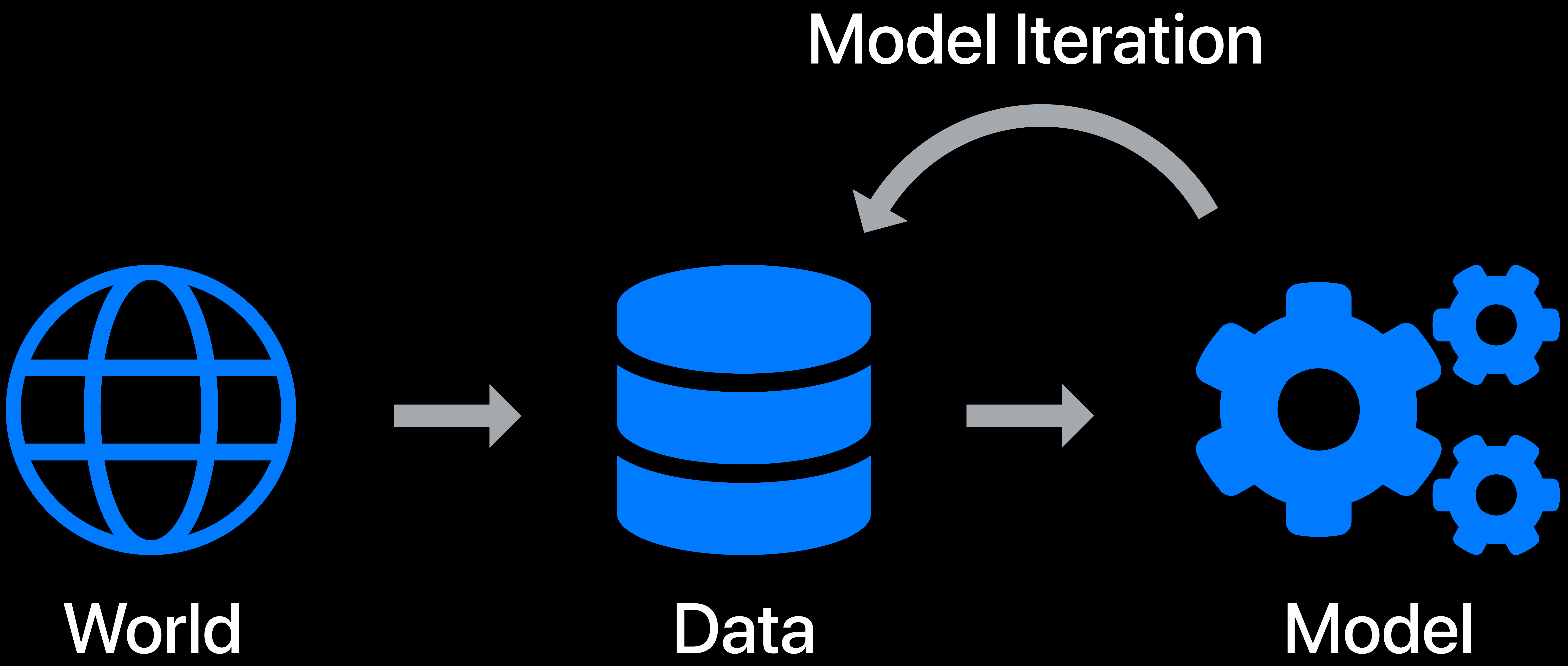
"clean and grow the training set"

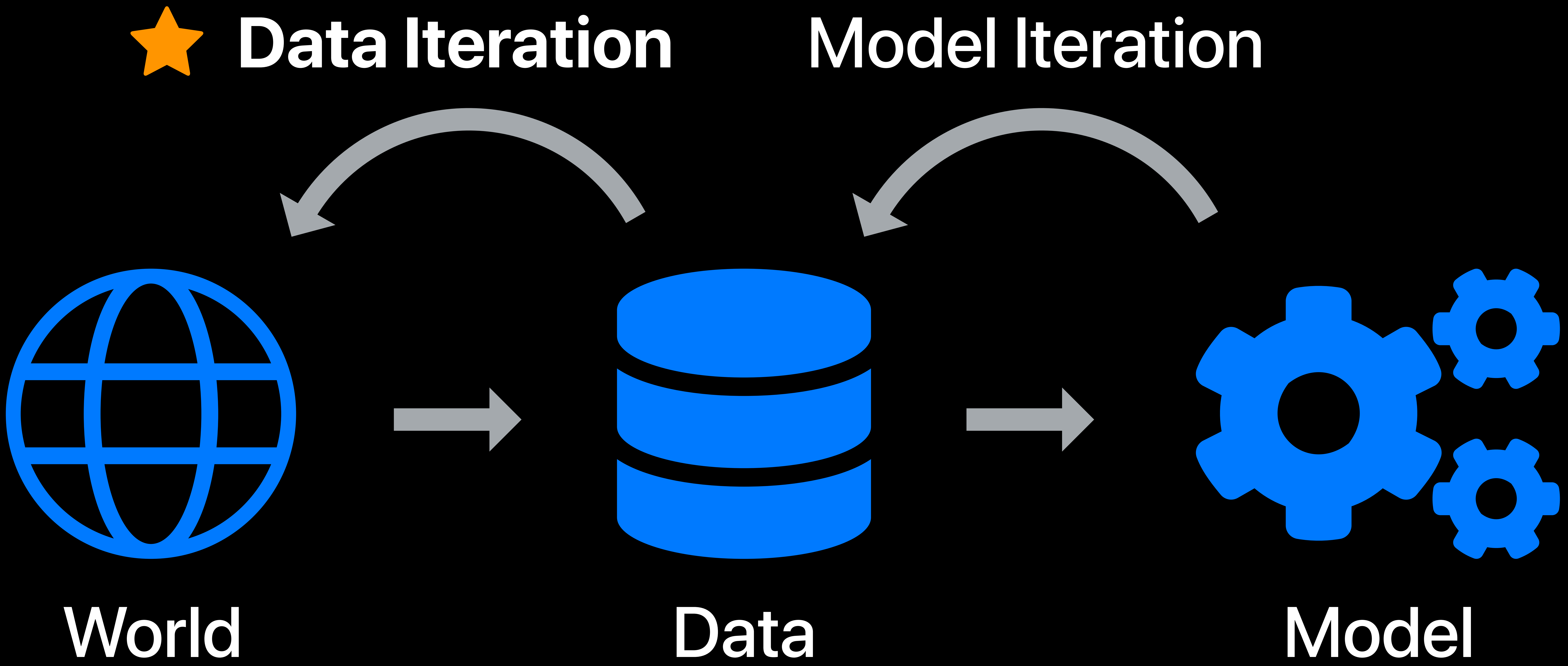
"repeated this process until accuracy improved enough"

This is a **data iteration**!

https://twitter.com/karpathy/status/1231378194948706306


World
Data
Model

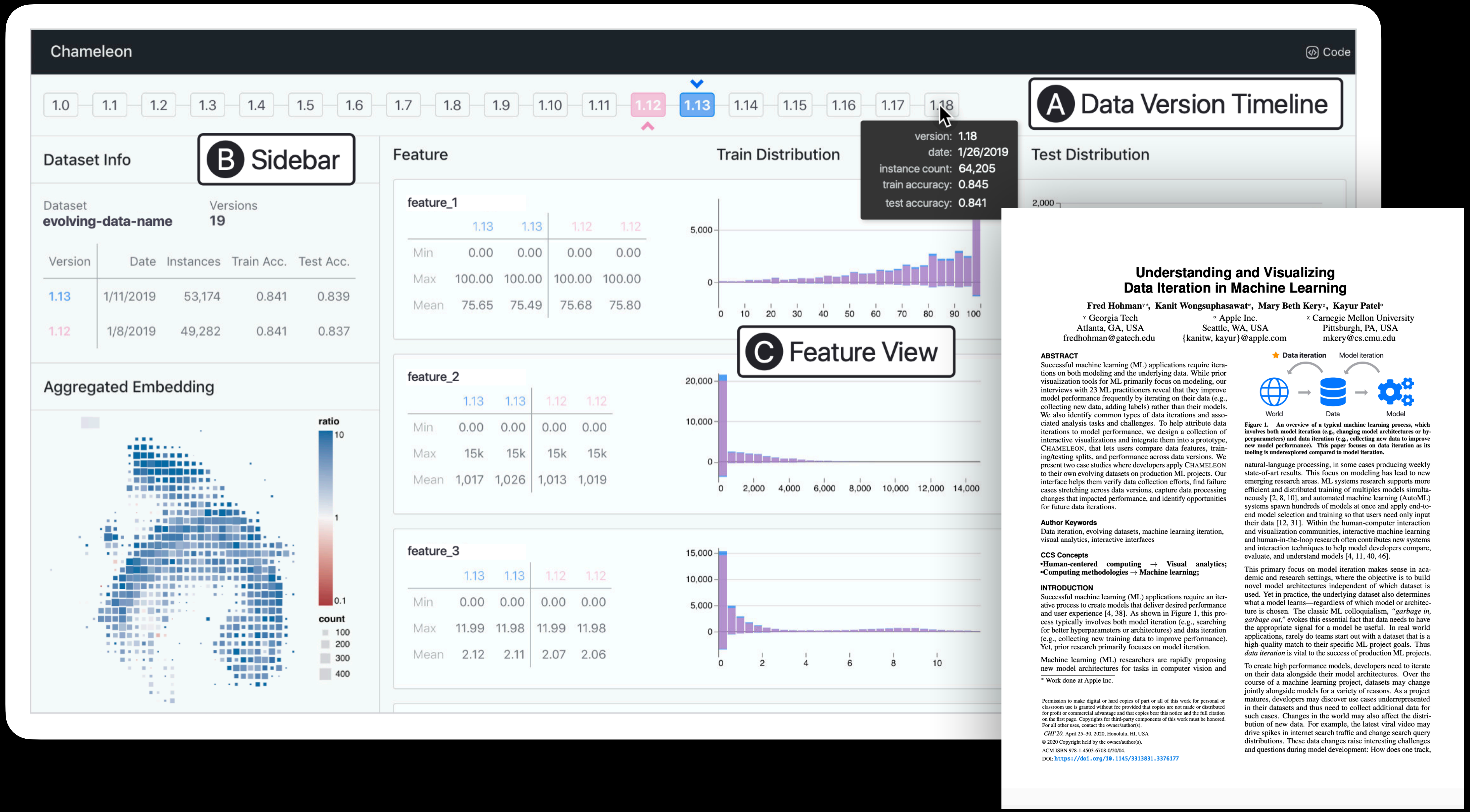

Model Iteration
World
Data
Model


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



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
Senors	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 boostraps 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

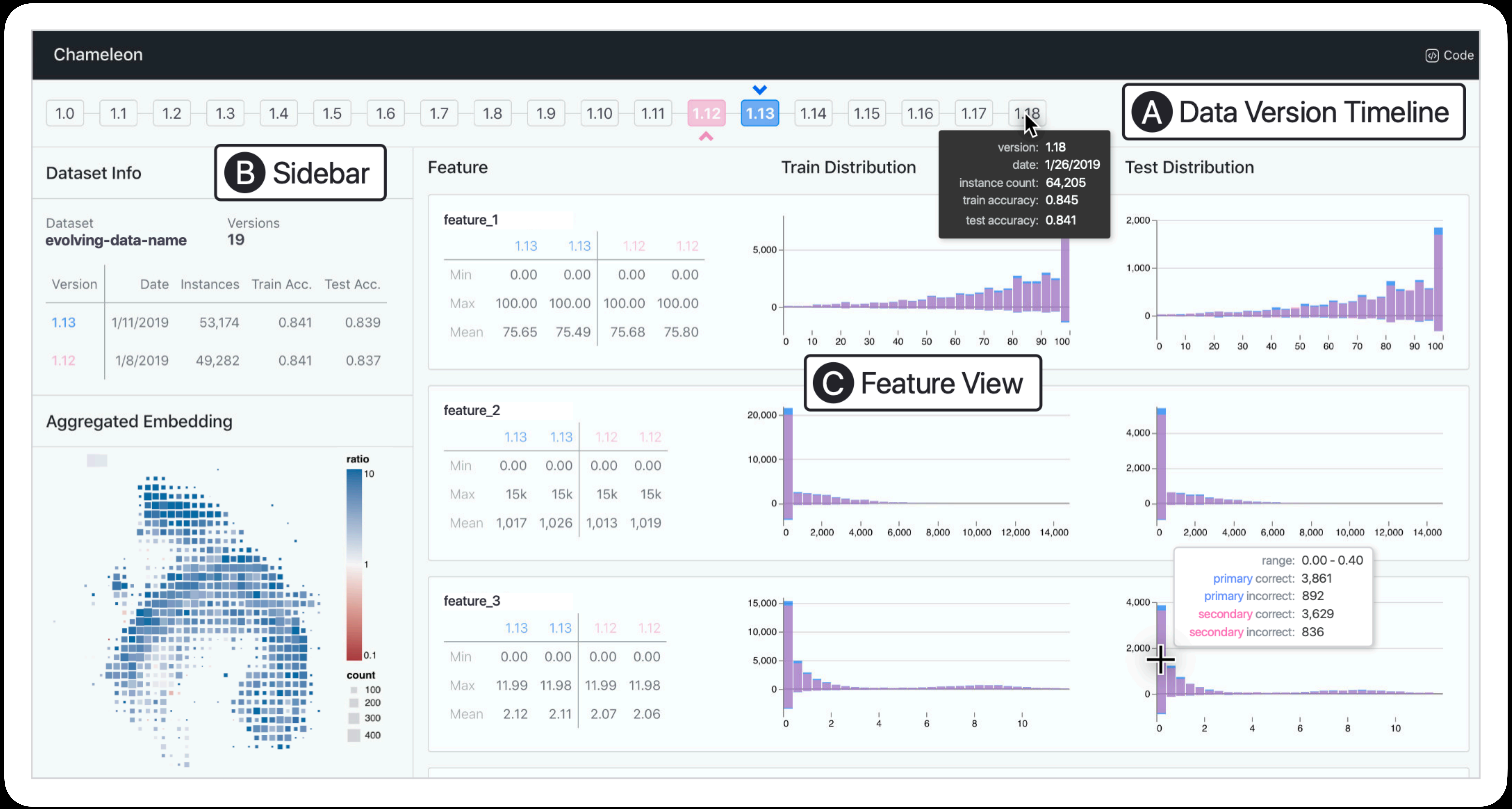
- When to “unfreeze” data versions
- When to stop collecting data

- 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

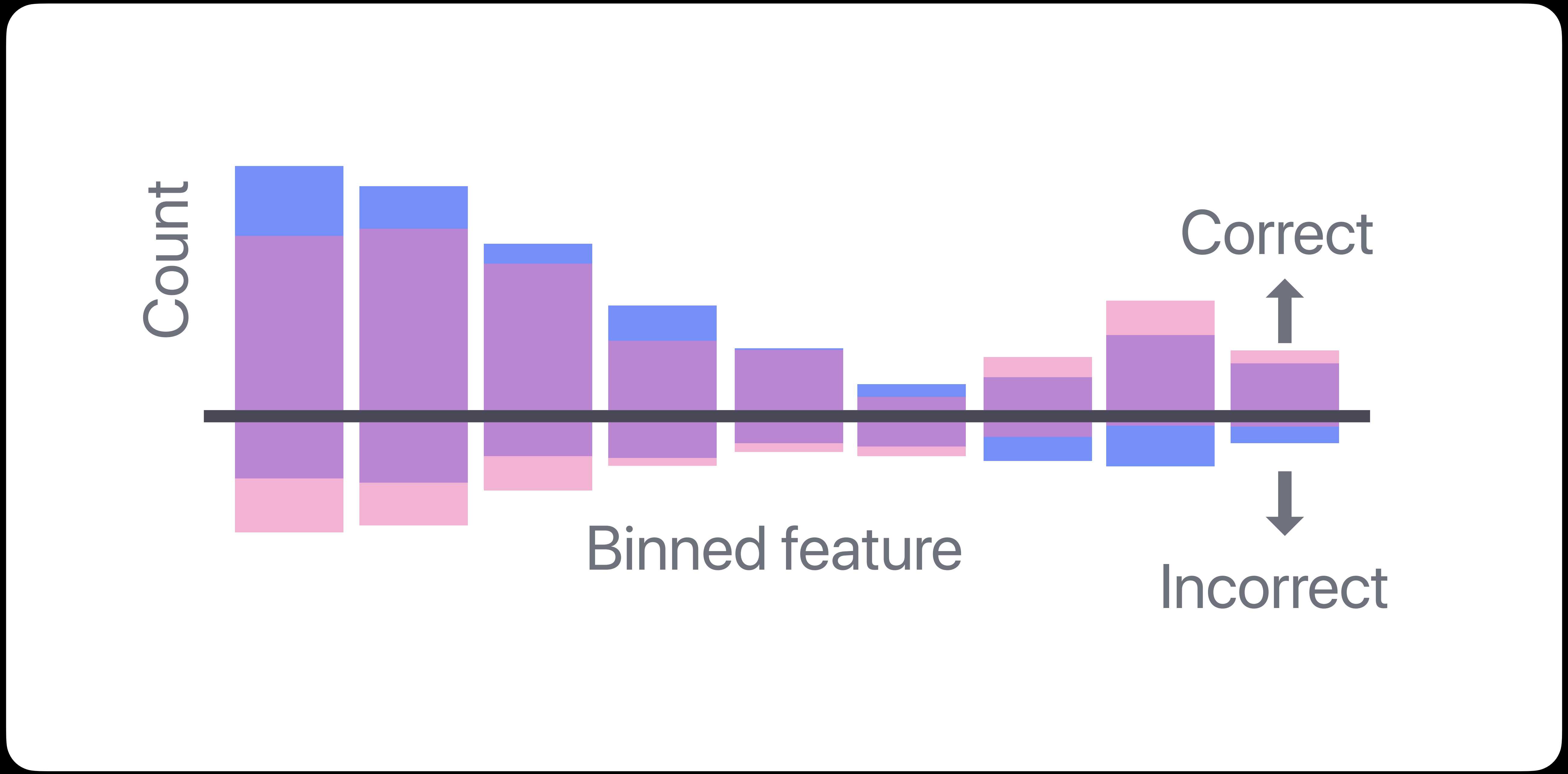


Chameleon

Understanding and Visualization Data Iteration

Compare feature distributions by:

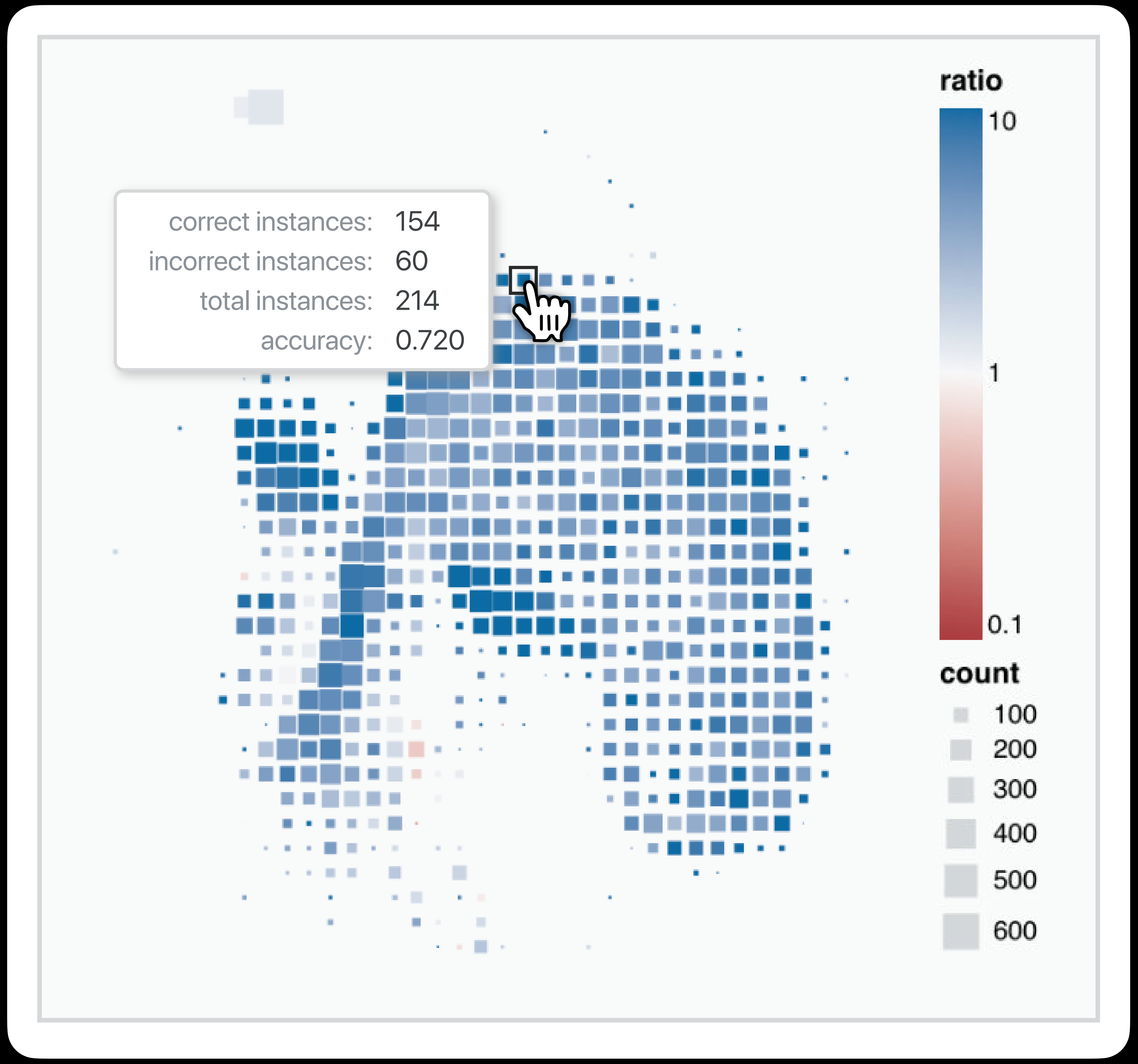
- Training and testing splits
- Performance (e.g., correct v. incorrect predictions)
- Data versions



"Overlaid diverging histogram" per feature

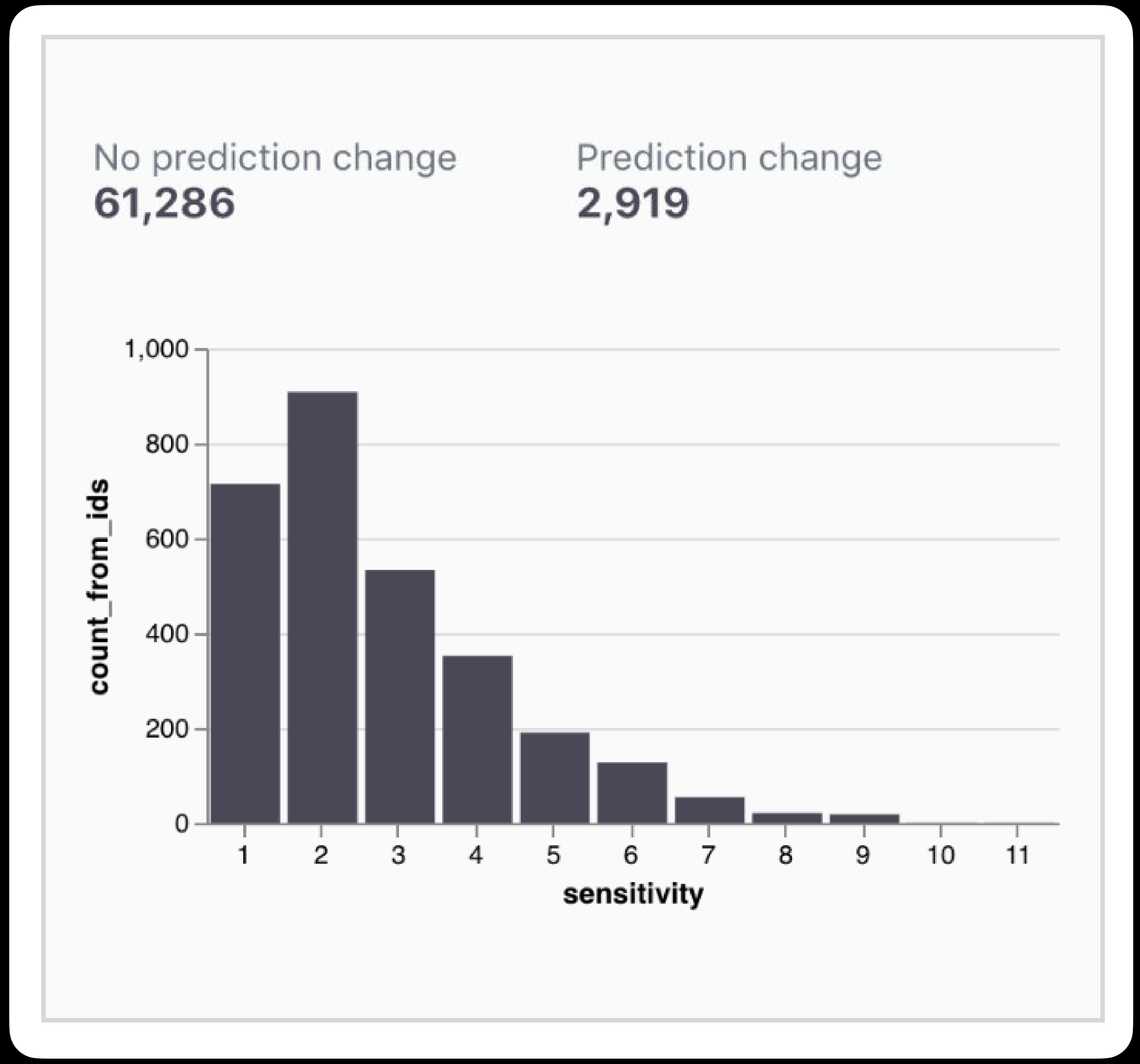
CHAMELEON Visualizations

Aggregated Embedding



Prediction Change Matrix

Sensitivity Histogram



Chameleon

Code

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

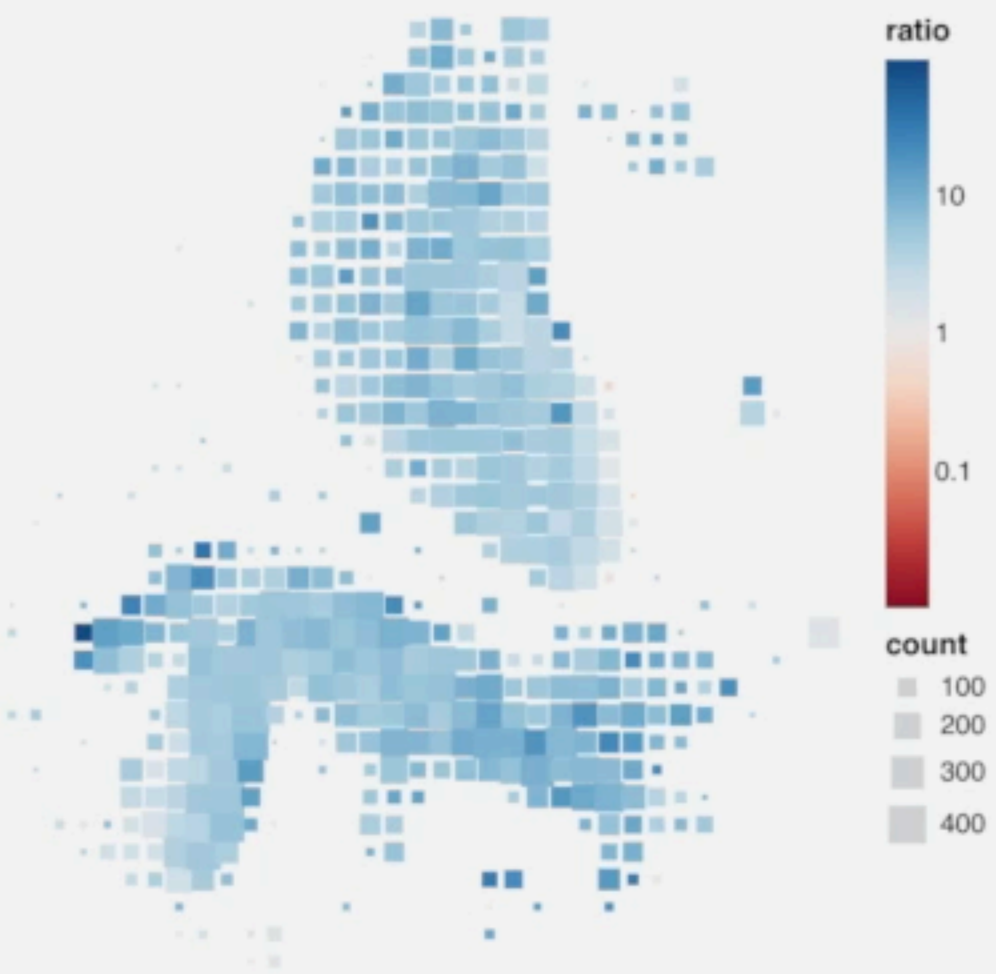
Dataset Info

Dataset
Redacted data name

Versions
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

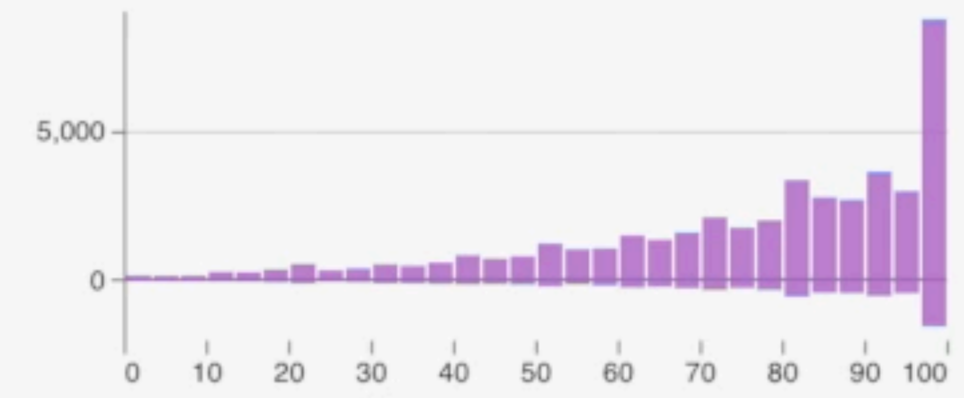


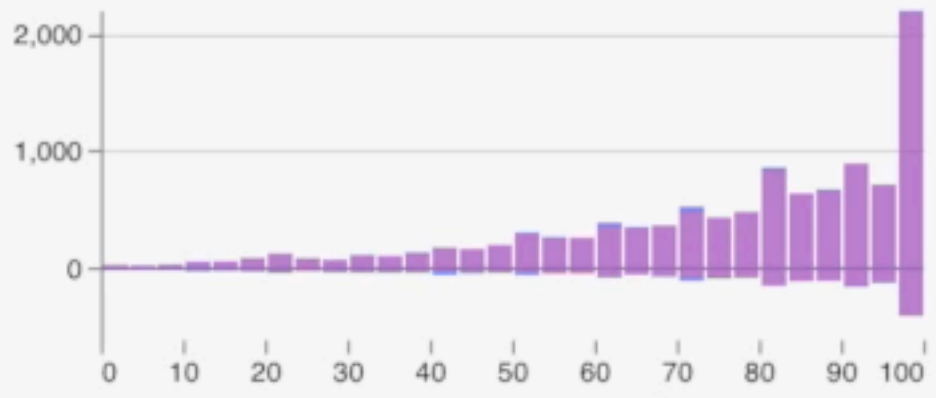
Prediction Change

Feature | Train Distribution | Test Distribution

Redacted feature name

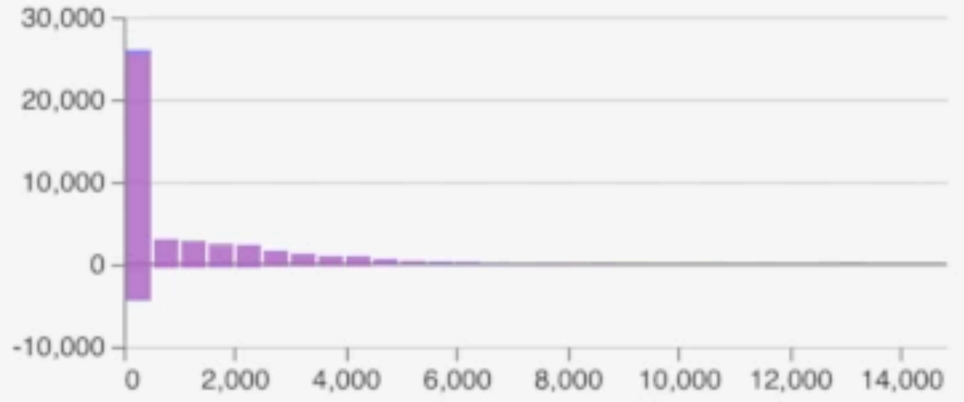
	1.18	1.18	1.17	1.17
Min	0.00	0.00	0.00	0.00
Max	100.00	100.00	100.00	100.00
Mean	75.44	75.47	75.50	75.49

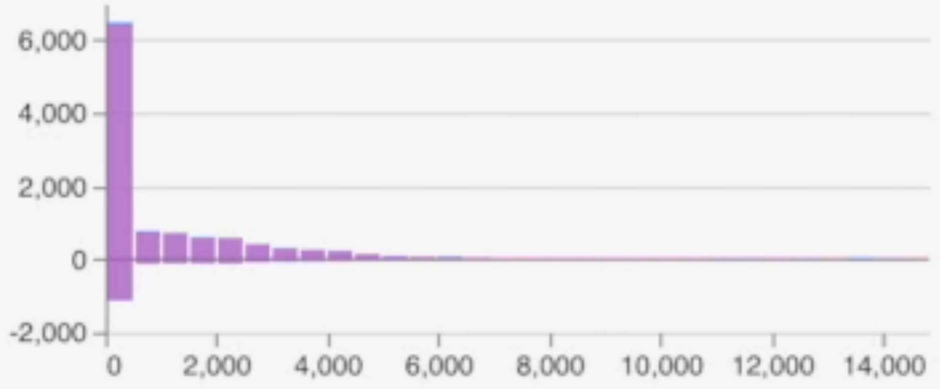




Redacted feature name

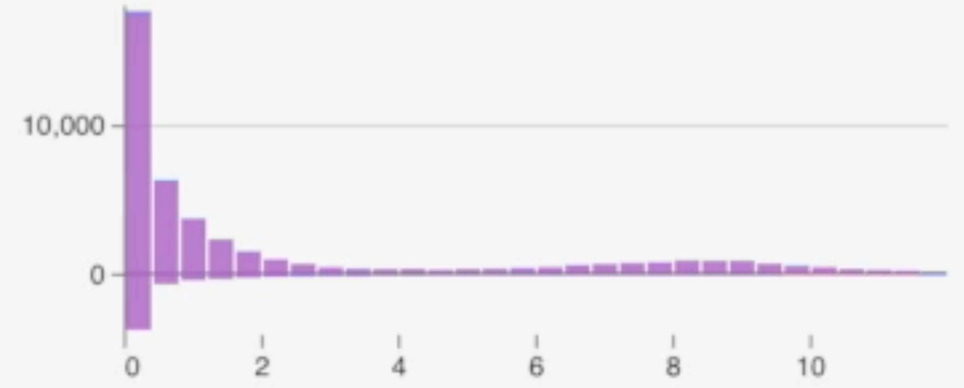
	1.18	1.18	1.17	1.17
Min	0.00	0.00	0.00	0.00
Max	15k	15k	15k	15k
Mean	1,026	1,031	1,024	1,027





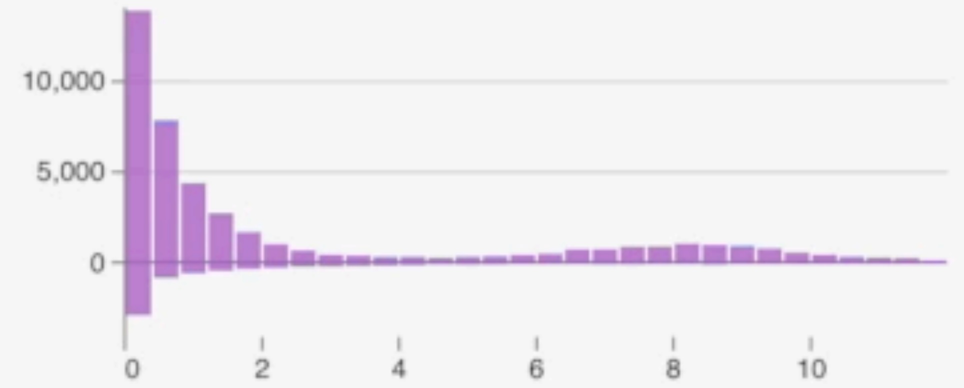
Redacted feature name

	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



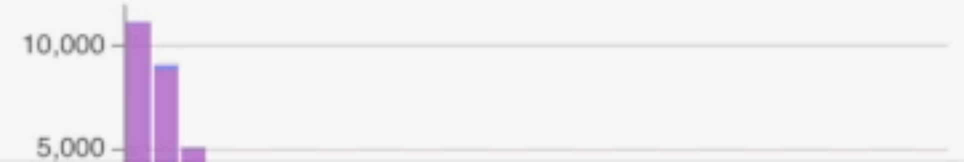
Redacted feature name

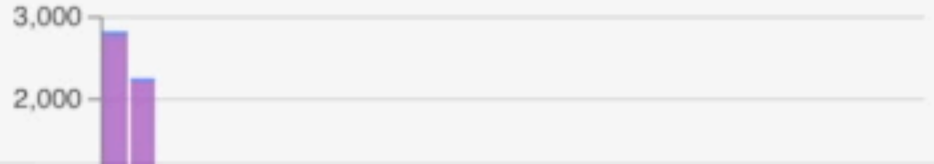
	1.18	1.18	1.17	1.17
Min	0.00	0.00	0.00	0.00
Max	11.98	11.89	11.98	11.98
Mean	2.46	2.44	2.45	2.46

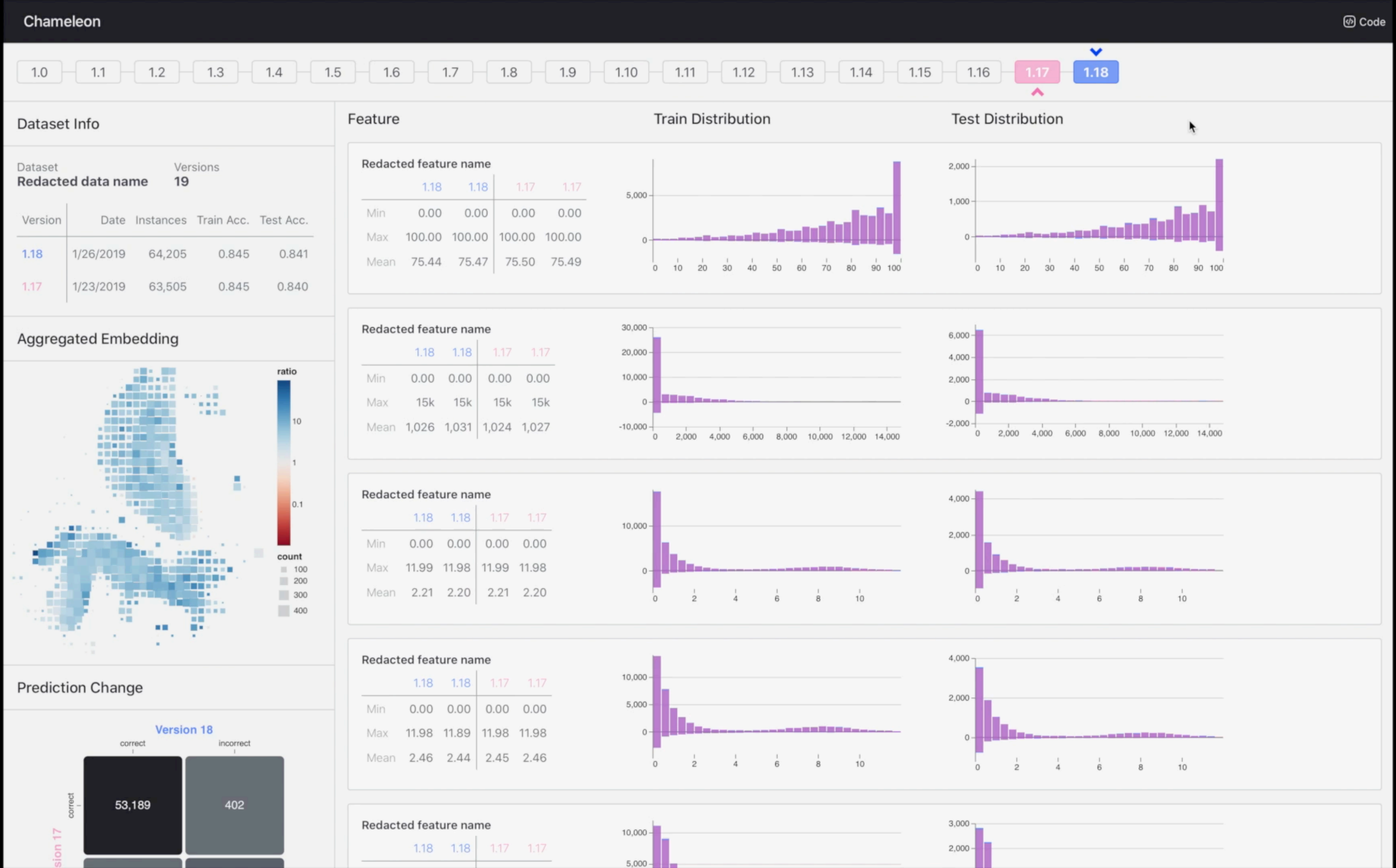


Redacted feature name

	1.18	1.18	1.17	1.17





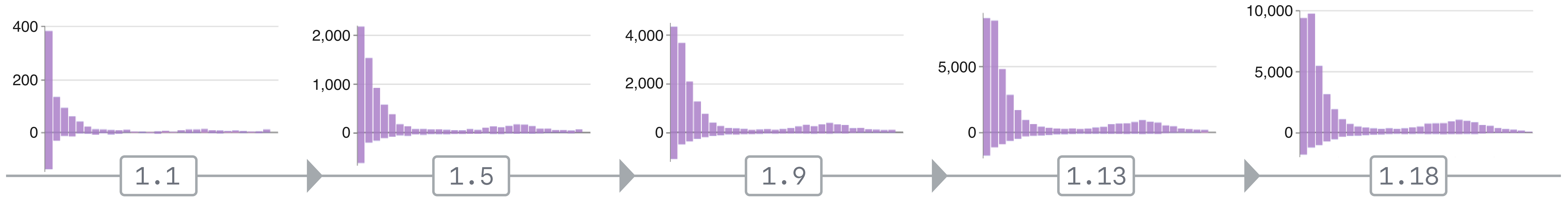


Chameleon
Code
1.0 1.1 1.2 1.3 1.4 1.5 1.6 1.7 1.8 1.9 1.10 1.11 1.12 1.13 1.14 1.15 1.16 1.17 1.18
Dataset Info
Dataset
Redacted data name
Versions
19
Version | Date | Instances | Train Acc. | Test Acc.
1.18 | 1/26/2019 | 64,205 | 0.845 | 0.841
1.17 | 1/23/2019 | 63,505 | 0.845 | 0.840
Aggregated Embedding
ratio
10
1
0.1
count
100
200
300
400
Prediction Change
Version 18
correct
incorrect
correct
53,189
402
sion 17
Feature
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 100.00 100.00 100.00 100.00
Mean 75.44 75.47 75.50 75.49
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

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

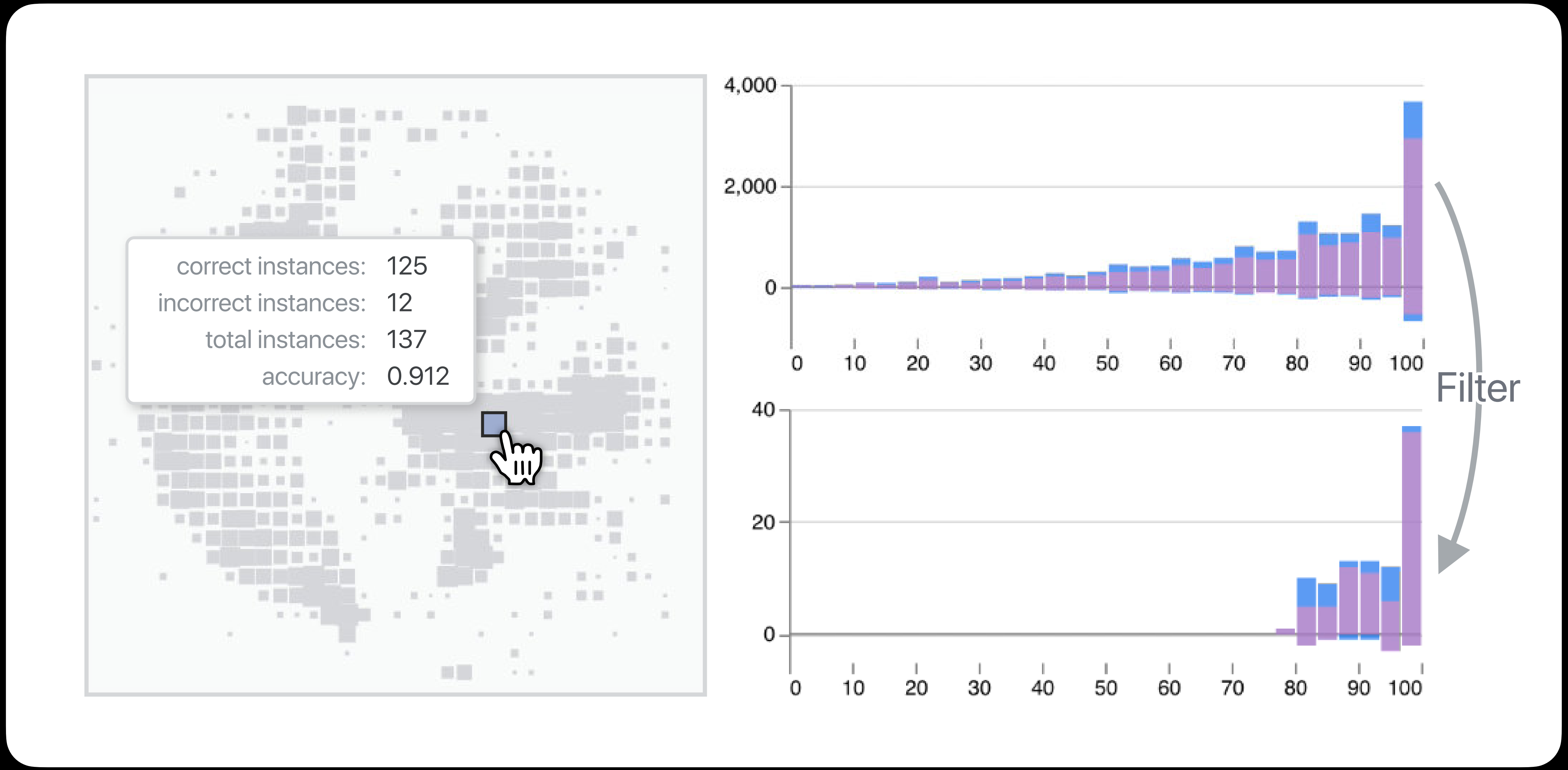


A feature's long-tailed, multi-modal distribution shape solidifies over collection time: 1,442 → 64,205 instances

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