We list some possible prediction methods in rough order of perceived utility.

Logistic regression

Neural networks

Should be good predictors for such a big data set if the right architecture can be found. Could train them on underutilised BSU HPC GPU cluster.

Convolutional

Speed ups available through CUDNN if I can install it on the HPC. Previous published networks applied to sequence have used convolutional networks.

Recurrent

Could be tricker to train than convolutional networks but I'm guessing more powerful as well.

Sequence embedding

What kind of embedding to use to represent the sequence?

Possible idea is to use this sort of thing:

```
# input: 2D tensor of integer indices of characters (eg. 1-57).
# input tensor has shape (samples, maxlen)
model = Sequential()
# embed into dense 3D float tensor (samples, maxlen, 256)
model.add(Embedding(max_features, 256))
# reshape into 4D tensor (samples, 1, maxlen, 256)
model.add(Reshape(1, maxlen, 256))
# VGG-like convolution stack
model.add(Convolution2D(32, 3, 3, 3, border_mode='full'))
model.add(Activation('relu'))
model.add(Convolution2D(32, 32, 3, 3))
model.add(Activation('relu'))
model.add(MaxPooling2D(poolsize=(2, 2)))
model.add(Dropout(0.25))
model.add(Flatten())
# then finish up with Dense layers
```

Also from same issue (https://github.com/fchollet/keras/issues/233):

```
ngram_filters = [3, 4, 5, 6, 7, 8]
conv_filters = []

for n_gram in ngram_filters:
    conv_filters.append(Sequential())
    conv_filters[-1].add(Convolution2D(nb_feature_maps, 1, n_gram, wv_sz))
    conv_filters[-1].add(MaxPooling2D(poolsize=(nb_tokens - n_gram + 1, 1)))
    conv_filters[-1].add(Flatten())

model = Sequential()
model.add(Merge(conv_filters, mode='concat'))
model.add(Dropout(0.5))
model.add(Dense(nb_feature_maps * len(ngram_filters), 1))
model.add(Activation('sigmoid'))
```