Image Classification Using Biological Neural Networks

Alex Mallen

Abstract

While the application of artificial neural networks to image classification is a prominent and fast-growing area of research, we still know relatively little about how their biological predecessors work. In this paper, I used data from the Allen Institute for Brain Science to explore the extent to which we can map neural responses in the mouse visual cortex to specific visual stimuli. Specifically, I attempted to classify the stimulus presented based on the spatio-temporal neural response, and found significant but imperfect clustering of responses. Additionally, I explored the extent to which neural response is redundant using principle component analysis (PCA).

1 Introduction

A central problem in computational neuroscience is neural decoding: recovering the stimulus that produced a measured neural response. This problem is fundamentally asking how the brain represents sensory information. In this paper I used data from the Allen Institute for Brain Science to decode responses to two kinds of visual stimuli—static gratings and natural images—and attempted to classify which image was presented based on the spatio-temporal neural response. In a final experiment, I analyzed the redundancy of the neural response to a movie using PCA.

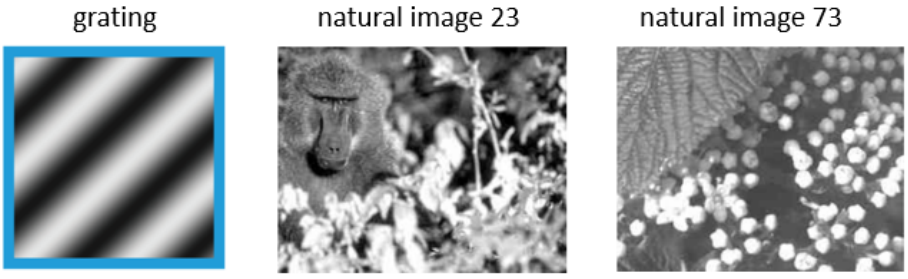

FIGURE 1: Example images presented to mouse [8].

1.1 Methods

Static gratings are sinusoidal patterns of stripes with various orientations, spatial frequencies, and phases, like the one shown in figure 1. In one experiment, which I will refer to as the controlled experiment, all gratings have a spatial frequency of 0.04 cycles per degree (cpd) and a phase of 0. The goal of this experiment is to classify the orientation of the presented grating as one of six possible angles (0, 30, 60, 90, 120, and 150 degrees) based on the spatio-temporal neural response of the mouse. There are 294 data points in this experiment, with roughly 50 instances of each grating. Classifying the orientation of these gratings will serve as a benchmark task.

In another experiment, the uncontrolled experiment, the gratings have many possible combinations of frequency and phase, but the phase and frequency are still uncorrelated with the orientation. Classifying the orientation in the uncontrolled case is more challenging, but there is more data. Every combination of orientation, frequency, and phase is presented about 50 times for a total of 5811 data points.

There are 118 unique natural images which are each presented exactly 50 times. Natural images were taken of various animals, plants, and structures [8], and are meant to give insight into real-world functioning of the visual cortex. Both natural images and static gratings are presented for 250 ms.

The data is from 60 neurons in the primary visual cortex (VISp) of a 118-day-old male mouse with Sst genotype, recorded in an Allen Brain Observatory session. I will use 3 classifiers in order to decode the stimulus: support vector machine (SVM), maximum likelihood, and covariance.

For the final experiment I used a natural movie to analyze the redundancy of neural responses. I chose to use a natural movie because it is 30 seconds, rather than 250 ms, which means that the responses are high dimensional and will not limit the rank of the data matrix.

2 Theory

As described in my previous paper [3], principle component analysis projects a 0-mean dataset onto an orthogonal basis such that the first n basis vectors (known as principle components) are the n vectors that explain the most variance of the dataset. This is a powerful tool for dimensionality reduction. The covariance matrix of a matrix X with features as rows is given by $X_c^T X_c$, where X_c is the mean-subtracted matrix. PCA uses the eigendecomposition of the covariance matrix to find a basis in which the covariance matrix is diagonal. This is equivalent to finding the singular value decomposition of X_c, given by

$$X_c = U\Sigma V^T, \tag{1}$$

where Σ is a diagonal matrix of singular values, whose squared entries indicate the variance of the data along the corresponding principle components in U, the columns of U form

an orthogonal basis for the entire dataset sorted by importance, and the columns of ΣV^T represent each of the data points in the principle component basis. PCA is the tool used to determine the redundancy of the neural responses to the natural movies; if the first few singular values are much larger than the rest, the response is redundant. PCA is also used as a preprocessing step for SVM classification.

SVMs are supervised machine learning algorithms that attempt to linearly separate the data. SVMs are only directly capable of two-class partitioning. The objective of an SVM given data in $\mathbb{R}^n$ of two distinct classes is to find a dividing $(n-1)$-dimensional hyperplane that minimizes the number of misclassifications by incurring a penalty for misclassifications, while maximizing the margin between the hyperplane and the closest points (called support vectors) [5]. I chose to use SVM classification because it was the most reliable in my previous experiments [2].

Covariance classification assumes that all members of a certain class will be highly correlated with the average member of that class (ie. that the convex hull would be a good decision boundary). It is a supervised clustering algorithm that takes the average data point $\bar{\mathbf{x}}_i$ of each class i and then computes the dot product between each $\bar{\mathbf{x}}_i$ and a new observation $\mathbf{x}$ and classifies $\mathbf{x}$ as the argmax product. If both vectors are 0-mean, then this computes the covariance between the two vectors, but this is not necessarily the case.

Another commonly used supervised classification technique is maximum likelihood. If we assume that every component x_{ij}, $j = 1, 2, \ldots, n$ of the response $\mathbf{x}_i \in \mathbb{R}^n$ to stimulus i is independent and normally distributed, then we can estimate the mean $\hat{\mu}_{ij}$ and variance $\hat{\sigma}_{ij}^2$ of the distributions using the sample mean and standard deviation from all observations of class i in the training data. Then we classify a new observation $\mathbf{x}_{\text{test}}$ as the class which has the highest probability of producing $\mathbf{x}_{\text{test}}$ given the assumed distribution $\mathcal{N}(\hat{\mu}_i, \hat{\sigma}_i^2)$.

One commonly used metric in computational neuroscience is mutual information I_m, which quantifies how much information one random variable conveys about another. I computed the mutual information between the stimulus and the predicted stimulus in order to gain an understanding of what factors contribute to mutual information.

For intuition, a mutual information between random variables $\mathcal{S}$ and $\mathcal{R}$ of $I_m = 3$ bits means that, on average, knowing $\mathcal{S}$ narrows down the "number of possibilities" (probabilistically) for $\mathcal{R}$ by a factor of $2^3 = 8$. More rigorously,

$$I_m(\mathcal{S}, \mathcal{R}) = H(\mathcal{R}) - \sum_{s \in \mathcal{S}} P(s) H(\mathcal{R}|s) \tag{2}$$

where $H(\mathcal{R})$ is the entropy of $\mathcal{R}$ defined as

$$H(\mathcal{R}) = -\sum_{r \in \mathcal{R}} P(r) \log_2(P(r)). \tag{3}$$

Mutual information is mutual in that this value is symmetric with respect to its inputs $\mathcal{S}$ and $\mathcal{R}$ [9]. In my calculation of of mutual information, I attempted to gain insight into how much information the neural response encodes about the stimulus presented. This is not

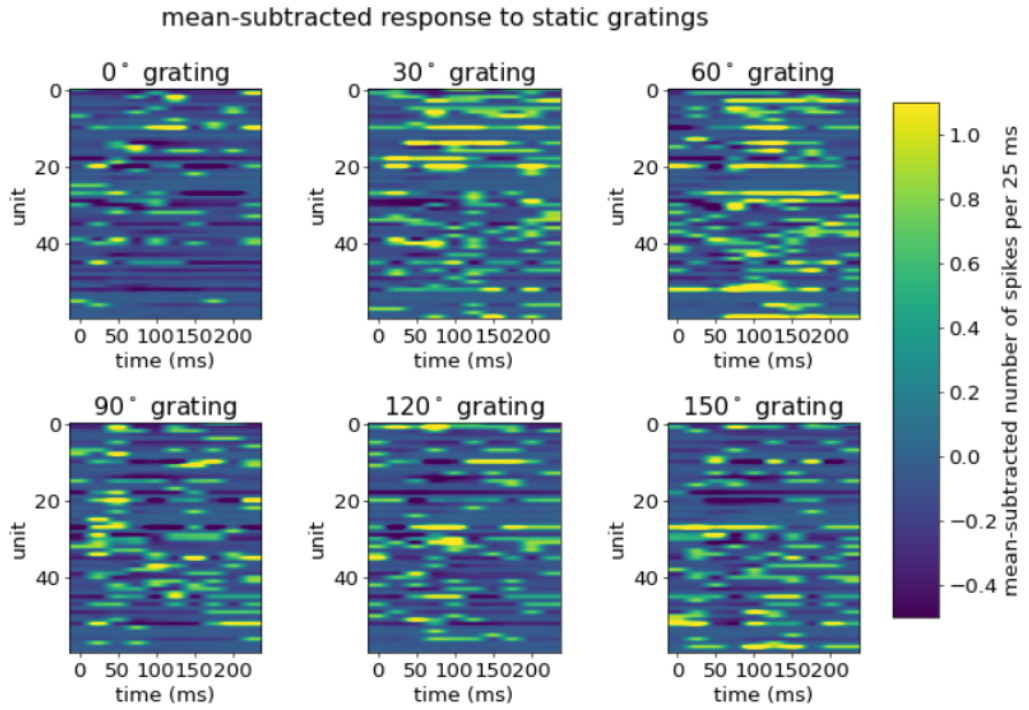

FIGURE 2: Example of the response of this mouse to the 6 different orientations of the grating in the controlled case, where the spatial frequency is 0.04 cpd and the phase is 0. Note: "unit" roughly means a neuron, but there is no guarantee that every measured unit corresponds to exactly one neuron.

the same as asking how much information the classifier-predicted stimulus encodes about the true stimulus because the classifiers are not perfect. Let $\mathcal{S}$ be the true stimulus, and $\hat{\mathcal{S}}$ be the predicted stimulus. The mutual information is therefore at least as large as the best mutual information $I_m(\hat{\mathcal{S}}_{\text{classifier}}, \mathcal{S})$ obtained by any classifier on the test set. I define a lower bound

$$I_m^* = \max_{\text{classifiers}} I_m(\hat{\mathcal{S}}_{\text{classifier}}, \mathcal{S}) \tag{4}$$

3 Implementation

I accessed the data through the `allensdk` Python library. The jupyter notebook with the source code can be found on github [1]. I should note that installing the `allensdk` was challenging, as it has many dependencies, and thanks to the help of my friend Nathan Yan, I found out that ultimately needed to side-by-side install Python 3.7 and specify a new kernel for jupyter. Accessing the data involves setting up a project cache so that I download only the required subset of the terabytes of available data, and only once. After filtering the sessions by genotype, session type, and brain regions measured, I selected one session to study in depth, and the IDs of the 60 units measured in VISp.

For each of the 3 experiments, I then obtained the stimulus table for the experiment by calling `session.get_stimulus_table([experiment_name])`. This contains the information about which stimulus was presented at what times, Finally, I obtained the measured neural response to each stimulus by calling `session.presentationwise_spike_counts`, which is described in appendix A. It returns a 3D array describing how many spikes were fired by each of the 60 units during each of 10 time bins, in response to each stimulus presented. An example response to each orientation of static grating is shown in figure 2. The average response was subtracted from each matrix and then it was reshaped into a vector in $\mathbb{R}^{600}$ which formed the columns of the matrix X_c. In order to ensure that no information from the test set was leaking into the training set, I partitioned X_c into a training and test set before performing PCA. I used a 70/30 training test split, and after experimenting with this ratio for the entire natural scenes dataset, I found 0.7 was the best fraction.

In order to visualize the structure of the dataset and to reduce the dimensions of my SVM classifier, I performed PCA on the training X_c matrix, following the same process as in my previous paper [2] (calling `np.linalg.svd` on X_c and projecting the data onto the rank-truncated U matrix). I used a 20-dimensional PCA space for my SVM classifier, so I trained a `sklearn.svm.SVC` classifier on the first 20 singular values times the first 20 rows of V^T. I then projected the testing X_c matrix onto the PCA space found in the training step by multiplying it on the left by U^T, and taking the first 20 rows, before classifying it. PCA was not used for covariance and maximum likelihood classification because it would not have a purpose in these cases, so I instead used X_c directly. I wrote `covar_classify` and `max_likelihood_classify` functions, which are described in appendix A. For all three classifiers, I wrote a wrapper function `<classifier>_evaluate` that trains, tests, calculates I_m^* using equation 4, and plots a confusion matrix for the given data.

For the natural image experiment, I tried classifying various numbers of unique images with, as expected, varying results. I therefore decided to perform a more complete experiment to that end. I looped over a logarithmically spaced list from 2 unique images to all 118 unique images, and I performed PCA and clustering for each number of images. I plotted the performance of each clustering algorithm versus the number of unique images, as well as a lower bound estimate of the mutual information I_m^*.

Coding Redundancy

My first step in analyzing the redundancy of the neural response was to load the spike counts of the neural response by calling `session.presentationwise_spike_times`, and computing a histogram of response times for each of the 20 repetitions of the 30 second video. The reason for using this process is described in Appendix A 5.0.2.

I averaged the response over all 20 trials to obtain each unit's typical response, and then subtracted the average response of all the units. I then called `np.linalg.svd` on this matrix to obtain a list of singular values, S. Next, I computed the variances along each principle component by squaring S, and then computed the overall variance of the dataset.

In a for-loop over the elements of S^2, I simultaneously computed the cumulative explained variance from the first r principle components and found the smallest rank that explained at least 90% of the variance. I performed this for various temporal resolutions of the histogram to measure the robustness of the calculated rank. I also performed this on a 120 second video, for which there were 10 repetitions instead of 20 as a further measure of robustness.

4 Results

As seen in figure 3, the projection of the neural responses onto the first 3 PCA modes in the controlled experiment show definite but imperfect clustering. Meanwhile, in the uncontrolled test, clustering is far less apparent. All three singular value spectra are similarly shaped: they have a few dominant principle components, but the remaining components are nontrivial. The last few singular values in the controlled case and in the natural images are 0 likely because the number data points and the number of features were of the same order of magnitude. The 6 randomly chosen natural images also show significant clustering in PCA space.

Table 1 shows the error rate of each classifier at 4 tasks. The first three tasks, where there were 6 classes to distinguish between, serve as a direct comparison between the difficulty of each task. Unsurprisingly, predicting the orientation of the static grating from neural responses was more difficult in the uncontrolled experiment, where the gratings had varying spatial frequencies and phases. However, it came as a surprise to me that distinguishing between 6 natural images was as easy as the controlled static gratings, despite the poorly-defined differences between natural images. This likely reflects that biological systems are adapted to distinguishing natural scenes, even though it is easier to rigorously describe the differences between static gratings. That SVMs and covariance classifiers could correctly predict which stimulus was presented up to 75% of the time is an impressive result given the level of obfuscation in the brain. This is 4.5 times better than a random classifier. As expected, classifying all 118 natural images had a lower success rate of around 20%. However, when compared to a random classifier, it still performed more than 20 times better.

Notice in figure 4 that image 73 and image 23 are often confused, meaning they induce a similar neural response. These two images are shown in figure 1. Despite having unrelated

CLASSIFIER	controlled grating		uncontrolled		6 natural images		118 natural images	
	train	test	train	test	train	test	train	test
SVM	0.15	0.33	0.41	0.60	0.12	0.25	0.5	0.79
max likelihood	0.10	0.49	0.47	0.57	0.09	0.46	0.17	0.86
covariance	0.22	0.30	0.58	0.60	0.24	0.36	0.61	0.78

TABLE 1: Error rate in decoding the visual stimulus for the SVM clustering algorithm, maximum likelihood decoding, and covariance decoding.

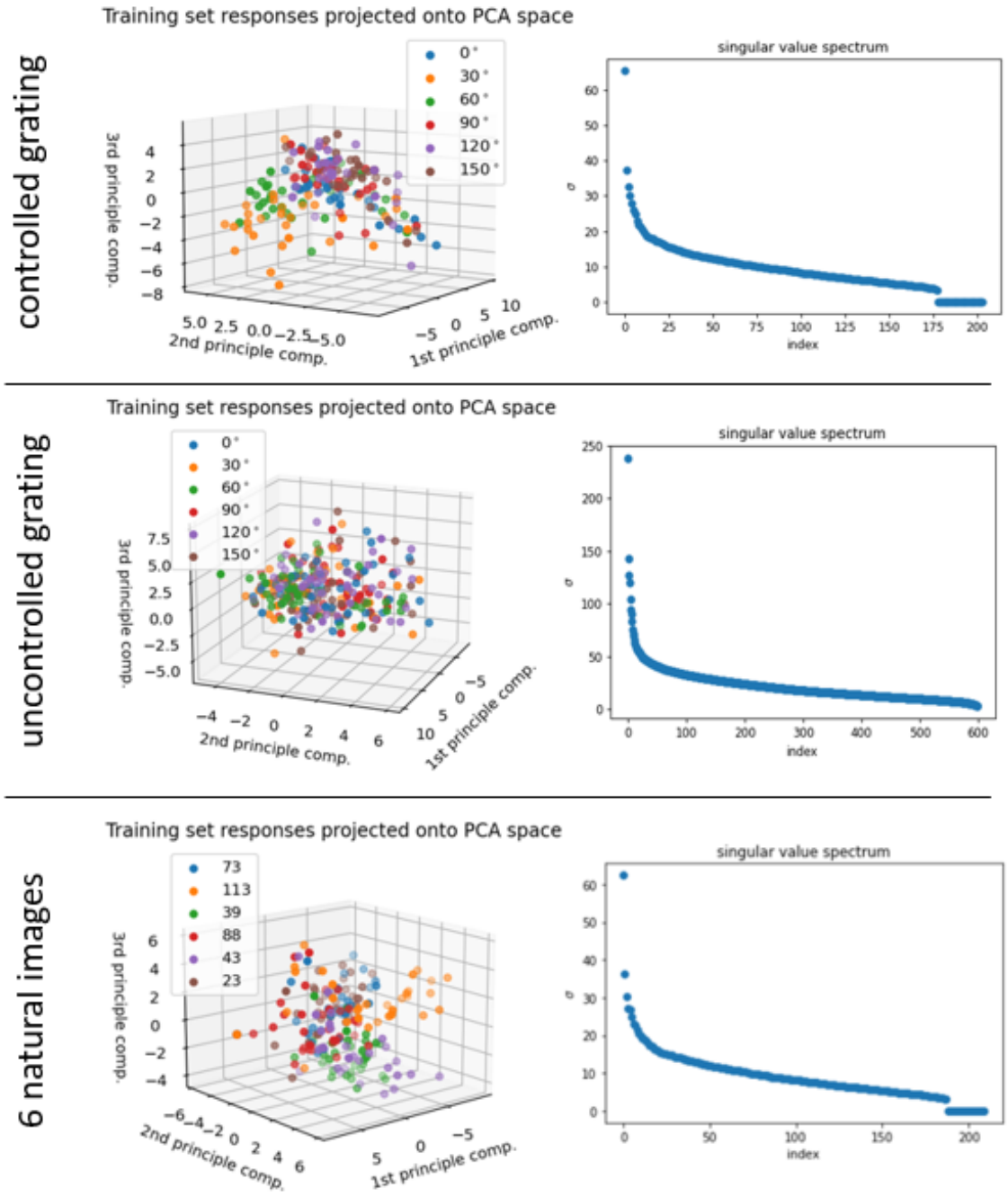

FIGURE 3: PCA results. The PCA results of all 118 natural images are similar to those of the 6 randomly sampled ones shown and are omitted.

subjects, they do have one feature in common: namely, a diagonal edge across the upper-left corner, partitioning dark from light. This indicates that the neurons we are measuring in this mouse's brain are processing the shapes and edges in the visual stimulus.

The results of classifying natural image datasets with a varying number of unique images is shown in figure 5. There is a consistent increasing relationship between the error rate and

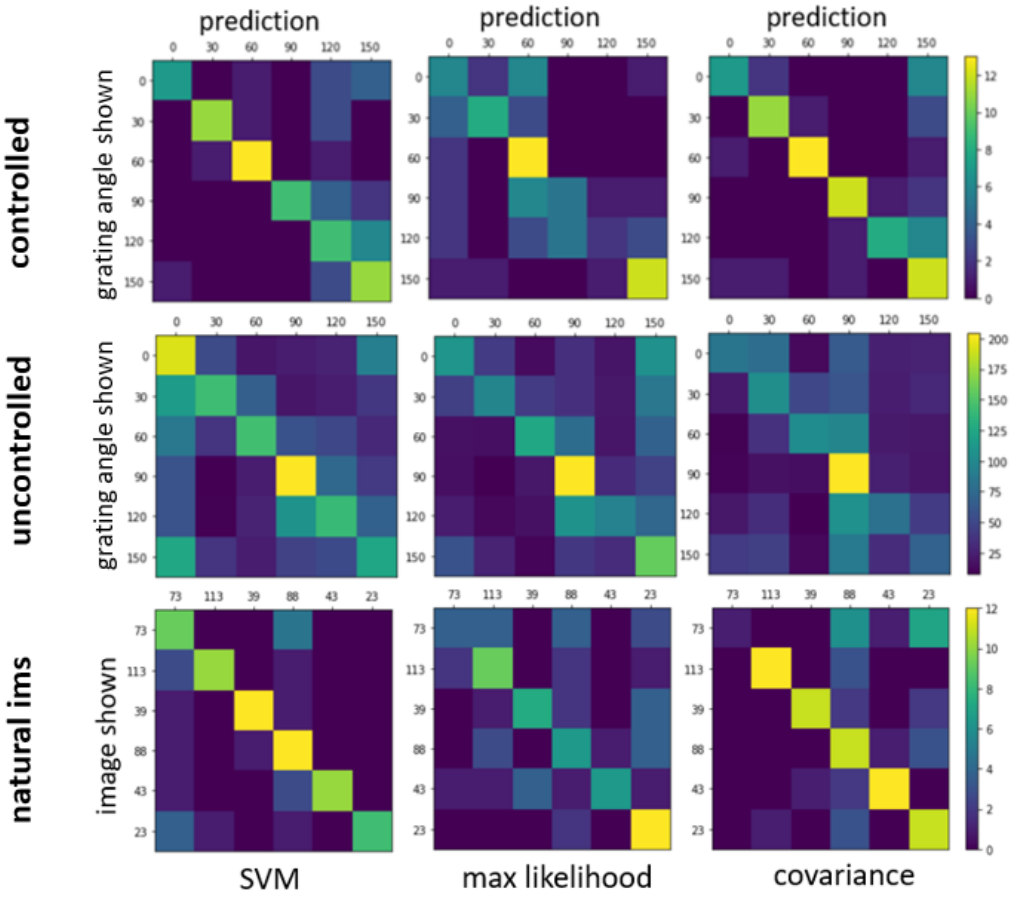

FIGURE 4: Confusion matrices for various experiments and classification algorithms. The brightness indicates the number of classifications.

the number of classes for all classifiers. The high variability near the beginning is likely due to the fact that certain natural images are easier to distinguish than others. Mutual information seems to increase logarithmically with the number of unique classes, which makes sense because it is the difference of two entropies, which scale with the logarithm of the number of possibilities.

The lower bound mutual information I_m^* in the controlled case was 1.38 bits, while it was 1.52 bits for natural scenes, supporting the idea that the neural response to natural images carries more information about the stimulus than the neural response to static gratings. Meanwhile for the uncontrolled case it was 0.413. Assuming that the mutual information is constant for all spatial frequencies and phases, the mutual information for the controlled and uncontrolled tests should be identical. The fact that the lower bound estimate was very different for the two cases indicates that our classifier was poor, and could not handle the complexity introduced by altering other variables of the grating.

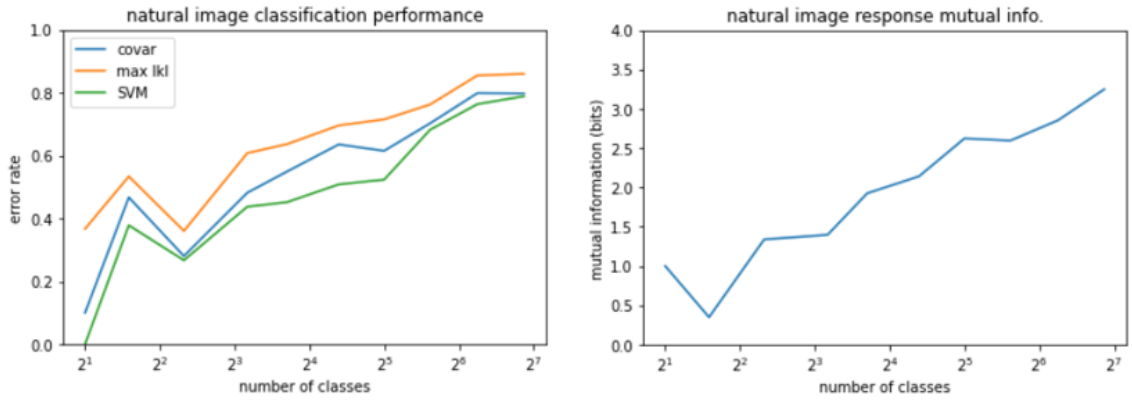

FIGURE 5: Error rate and mutual information as a function of the number of unique images provided to distinguish between.

Coding Redundancy

Figure 6 shows the results for how repetitive the neural responses were to the natural movie. As seen in the covariance matrix, few units' responses were significantly linearly associated

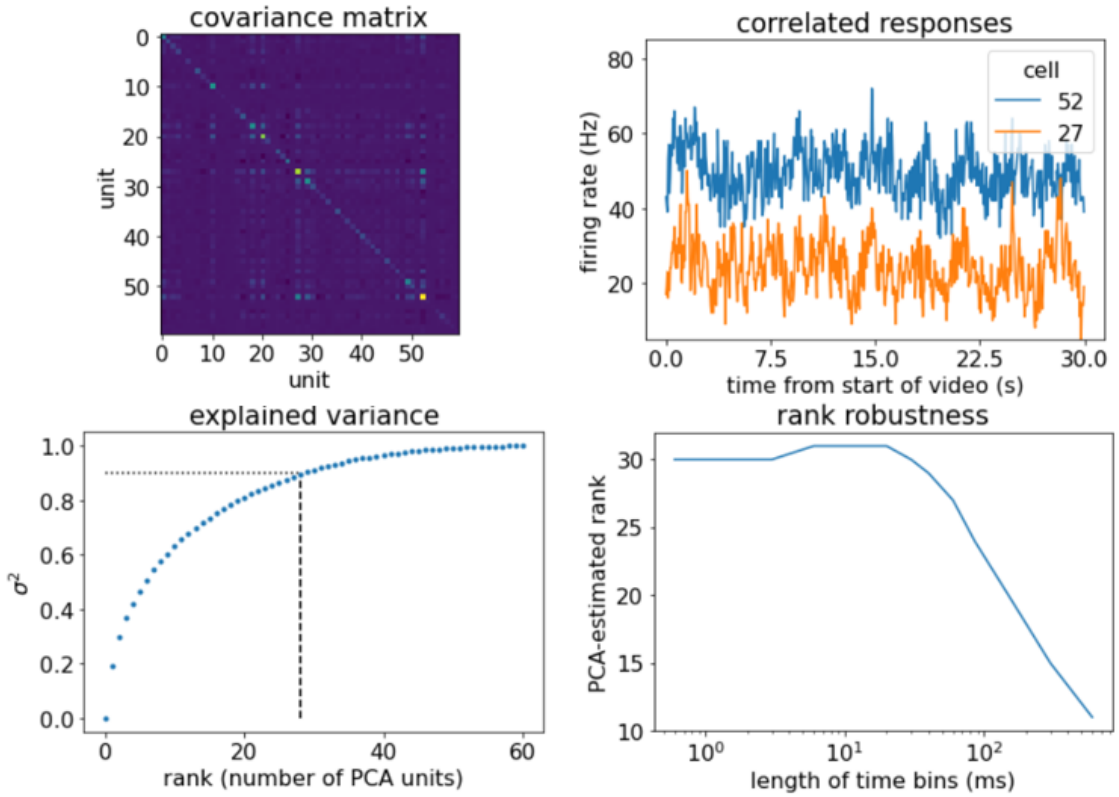

FIGURE 6: Figures illustrating the level of redundancy in the 60 neural responses to the natural movie when partitioned into 50 ms time bins. PCA suggests an underlying rank-28 structure in the neural responses. Lower Right: robustness of these results to the length of time bins.

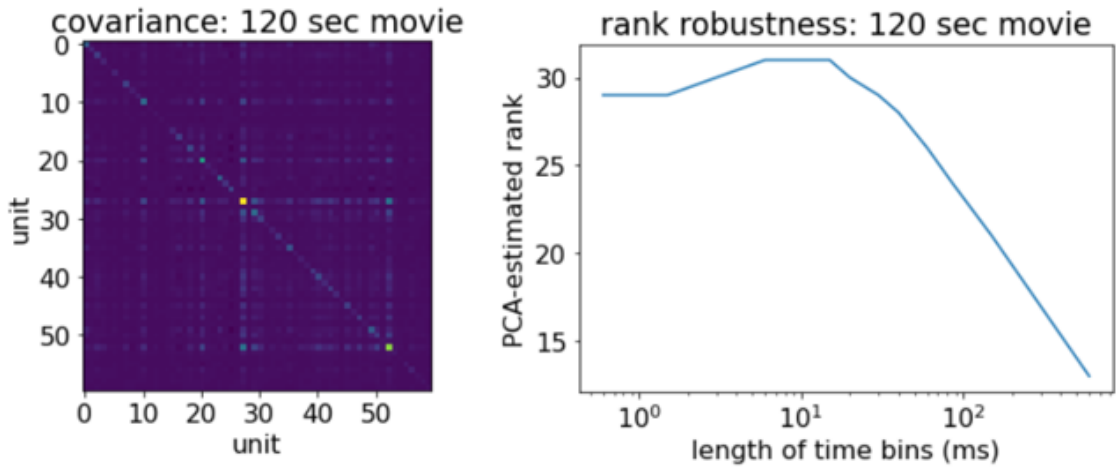

FIGURE 7: The covariance and PCA-estimated rank when testing on a 120 second movie, rather than 30 second, provide very similar results.

with each other. Units, 27 and 52 had the largest covariance, despite being located far apart physically. Their responses to the movie averaged over the 20 repetitions is shown in the upper right of 6. About 30 units account for 90% of the variance in the neural response to this movie. This rank estimate is robust to the length of the time bins once they are smaller than a threshold resolution of around 50 ms, as can be seen in the lower right of 6. Time bins of less than 1 ms produce encodings of the response where a 1 indicates that a spike occurred in that millisecond for a particular unit, while longer time bins often count multiple responses occurring in each time bin. Both produced similar rank estimates.

Moreover, figure 7 shows that when using a 120 second video rather than 30 seconds, the results are nearly identical: the rank was approximately 30 for all bin sizes of less than 50 ms, and the covariance matrix looked nearly identical. This indicates that this measurement of rank is robust, and that the reason why the bin size matters is because of resolution rather than the shape of the matrix. This rank estimate of 30 is half of the original size of this population of neurons, which suggests that covariance is significant, but there exists no small subset of neurons that explains a majority of the dynamics.

5 Conclusion

The Allen Institute for Brain Science databases are vast, and they are awaiting innumerable analyses. In this paper, I tackled a minute fraction of the data in a few small ways. I was able to successfully identify the stimulus that induced a spatio-temporal neural response in a single mouse using simple classification models. Simply seeing that there is structure to the responses in PCA space is an interesting finding.

When using the SVM classifier, I tried various kernels to improve the performance of classification by making a nonlinear decision boundary. I expected this to vastly improve

performance based on the fact that 9^{th} degree polynomial SVMs were able to achieve less than 1% error rate in classifying MNIST digits [10], and because neural responses are nonlinear, but linear classifiers worked the best in the end. I expect that this is because the small amount of data available (35 training points per class) was not enough information to build a nonlinear classifier without overfitting. Neural responses are complicated, highly nonlinear computations, with structure that is not entirely accounted for by the response space. For example, a high firing rate at one point in time is associated with a high firing rate at a neighboring point in time, but this domain knowledge is not built into the model.

Among many other potential follow-up investigations, it would be interesting to associate the response of specific units, or even neurons, with a stimulus. If this was performed on multiple mice, it may be possible to map the neurons of one mouse to those of another. While I expect this would be a challenging task, it could theoretically be used to communicate the visual thoughts from one mouse to another without the need for sensory input.

Appendix A

Important functions and their implementations

5.0.1 `sklearn.svm.SVC`

This is a SVM classifier class that can be trained using `SVC.fit(train_data, train_labels)`, and then used to predict the labels of new data using `test_predics = SVC.predict(test_data)`. There are two options for using an SVM for multi-class classification, given by the option `decision_function_shape`. If "ovo" (one-versus-one is passed, SVC creates a unique support vector machine for every pair of classes. The default is "ovr" (one-versus-rest), which creates one support vector machine for each class. See [6].

5.0.2 `EcephysSession.presentationwise_spike_counts`

Computes a histogram of spike events for given stimuli and units. This is a method of an allensdk session, and takes 3 arguments: a list of the unit IDs you are interested in, a list of stimulus presentation IDs you are interested in, and a list of bin edges used for the histogram for each stimulus presentation. A stimulus presentation means a specific frame (ie. one natural image presented for 0.25 s). Note that for natural movies, each stimulus presentation corresponds to one frame of the video and is only 30 ms, so this method cannot produce a histogram of the response over the course of the video. Instead, call `presentationwise_spike_times`, and call `np.histogram` on the spike times that occur within one presentation of the video.

5.0.3 `covar_classify`

Predicts the stimulus that produced given responses assuming the label i produces an average response of `avgs`$[i]$. For each input response, it loops over the average response to each of the labels, and computes the dot product between the input response and that label. It appends the label whose average associated response had the highest dot product with the given response to the prediction, and ultimately returns the prediction.

5.0.4 `max_likelihood_classify`

Predicts the stimulus that produced given responses assuming the label i produces an average response of `avgs`$[i]$, with a standard deviation of `stds`$[i]$. For each input response, it loops over the average response and standard deviation to each of the labels, and computes the log likelihood that a multivariate Gaussian with that label's mean and standard deviation would produce the response. It appends the label whose negative log likelihood of producing the observed response to the prediction, and ultimately returns the prediction.

Appendix B

Python code

```
#!/usr/bin/env python
# coding: utf-8

# In[1]:

import os
import shutil

import numpy as np
import pandas as pd
import matplotlib
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D
font = {'size': 16}
matplotlib.rc('font', **font)

from allensdk.brain_observatory.ecephys.ecephys_project_cache import
    EcephysProjectCache
# In[2]:
data_directory = './ecephys_cache' # must be updated to a valid directory
    in your filesystem
manifest_path = os.path.join(data_directory, "manifest.json")
# In[3]:
cache = EcephysProjectCache.from_warehouse(manifest=manifest_path)
# In[4]:
sessions = cache.get_session_table()
print('Total number of sessions: ' + str(len(sessions)))
# In[5]:
filtered_sessions = sessions[(sessions.full_genotype.str.find('Sst') > -1)
    & (sessions.session_type == 'brain_observatory_1.1') & (['VISp' in
    acronyms for acronyms in sessions.ecephys_structure_acronyms])]
print(len(filtered_sessions))
filtered_sessions.head()
# In[6]:
session_id = 715093703 # based on the above filter
session = cache.get_session_data(session_id)
# In[7]:
units = cache.get_units()
# In[9]:
unit_ids = units[(units.ecephys_structure_acronym=="VISp") &
    (units.specimen_id==filtered_sessions.specimen_id[session_id])].index
unit_ids.shape
```

```
# # Static Grating Classification
# ## Controlled orientation decoding

# In[27]:
stim_epochs = session.get_stimulus_epochs()
stim_epochs = stim_epochs[stim_epochs.stimulus_name=="static_gratings"]
stim_epochs
# In[28]:
stims = session.get_stimulus_table(['static_gratings'])
stims = stims[stims.orientation!='null']
# In[19]:
filtered_stims = stims[(stims.phase=="0.0") &
    (stims.spatial_frequency=="0.04")]
filtered_stims
# In[20]:
# In[21]:
num_bins = 10
stim_duration = filtered_stims.duration.min() # minimum so that none of
    the recording periods are overlapping
bin_edges = np.linspace(0, stim_duration, num_bins + 1)
counts = session.presentationwise_spike_counts(bin_edges=bin_edges,
                                               stimulus_presentation_ids=filtered_stims.index,
                                               unit_ids=unit_ids)
# In[22]:
oris = np.unique(filtered_stims.orientation).astype(int)
oris
# In[23]:
avg_resp = np.mean(counts, axis=0)
# In[26]:
fig, ax = plt.subplots(2, 3, figsize=(10, 8), dpi=60)
fig.suptitle("mean-subtracted response to static gratings")
for i, ori in enumerate(oris):
    plt.subplot(2, 3, i+1)
    idxs = filtered_stims[filtered_stims.orientation==ori].index
    plt.ylabel("unit")
    plt.xlabel("time (ms)")
#     im = plt.imshow((np.mean(counts.loc[idxs], axis=0).T - avg_resp),
    aspect=0.2, vmin=-0.5, vmax=1.1)
    im = plt.imshow(counts.loc[idxs[0]].T - avg_resp, aspect=0.2,
        vmin=-0.5, vmax=1.1)
    ax = plt.gca()
    ax.set_xticks(np.arange(0, num_bins, 2))
    ax.set_xticklabels((2000 * np.arange(num_bins // 2) * stim_duration /
        num_bins).astype(int))
    plt.title(f"{ori}$^\circ$ grating")
cbar_ax = fig.add_axes([1.0, 0.15, 0.05, 0.7])
```

```
fig.colorbar(mappable=im, label=f"mean-subtracted number of spikes per
    {int(1000 * stim_duration / num_bins)} ms", cax=cbar_ax)
plt.tight_layout()
plt.show()

# ## Data preprocessing and PCA
# In[522]:
np.random.seed(10)
# In[523]:
Xall = np.reshape(np.array(counts), (counts.shape[0], num_bins *
    len(unit_ids)))
labels = np.array(list(filtered_stims.orientation))
# train test split
train_frac = 0.7
train_idxs = np.array([], dtype=int)
test_idxs = np.array([], dtype=int)
for label in np.unique(labels):
    matches = np.nonzero(labels == label)[0]
    np.random.shuffle(matches)
    train_idxs = np.concatenate([train_idxs, matches[:int(train_frac *
        len(matches))]])
    test_idxs = np.concatenate([test_idxs, matches[int(train_frac *
        len(matches)):]])
# In[524]:
Xc_train = Xall[train_idxs] - np.mean(Xall[train_idxs], axis=0,
    keepdims=True)
Xc_test = Xall[test_idxs] - np.mean(Xall[test_idxs], axis=0, keepdims=True)
U, S, Vh = np.linalg.svd(Xc_train, full_matrices=False)
# In[525]:
proj_rank = 20
train = (np.diag(S[:proj_rank]) @ U.T[:proj_rank]).T
test = Xc_test @ Vh.T[:, :proj_rank]
# In[526]:
get_ipython().run_line_magic('matplotlib', 'notebook')
sample_size = 2000 # all
fig = plt.figure(dpi=120)
ax = fig.gca(projection='3d')
plt.title("Training set responses projected onto PCA space")
# for ori in oris[[1,3]]:
for ori in oris:
    ori_ims = train[np.nonzero(labels[train_idxs]==ori)[0],
        :3][:sample_size // len(oris)]
    ax.scatter(ori_ims[:, 0], ori_ims[:, 1], ori_ims[:, 2],
        label=f"{ori}$^\circ$")
ax.view_init(elev=15, azim=50)
plt.xlabel("1st principle comp.")
plt.ylabel("2nd principle comp.")
```

```
ax.set_zlabel("3rd principle comp.")
plt.legend()
plt.show()
# In[528]:
plt.figure()
plt.scatter(np.arange(len(S)), S)
plt.xlabel("index")
plt.ylabel("$\sigma$")
plt.title("singular value spectrum")
plt.show()
# In[531]:
len(S), S[177], S[178]
# In[532]:
def mutual_info(conf_mat):
    response_freqs = np.sum(conf_mat, axis=0) / np.sum(conf_mat)
    resp_entropy = -np.sum(response_freqs * np.log2(response_freqs +
        0.00001)) # bits
    print(resp_entropy)
    stim_entropy = 0
    for i in range(conf_mat.shape[0]):
        response_freqs = conf_mat[i] / np.sum(conf_mat[i])
        stim_entropy += -np.sum(response_freqs * np.log2(response_freqs +
            0.00001))
    stim_entropy /= conf_mat.shape[0]
    return resp_entropy - stim_entropy

# # covariance classification
# In[533]:
avgs = []
stds = []
train_avg = np.mean(Xc_train, axis=0)
for ori in oris:
    idxs = np.nonzero(labels[train_idxs]==ori)
    avgs.append(np.mean(Xc_train[idxs] - train_avg, axis=0))
    stds.append(np.std(Xc_train[idxs] - train_avg, axis=0))


# In[534]:
def covar_classify(Xc, avgs, oris):
    pre = []
    for response in Xc:
        covars = []
        for avg in avgs:
            covars.append(np.mean(avg.T @ response))
        ori_idx = np.argmax(covars)
        pre.append(oris[ori_idx])
    return pre
```

```
# In[586]:
def covar_evaluate(Xc_train, Xc_test, labels, train_idxs, test_idxs, avgs,
    oris, return_minfo=False):
    fit = covar_classify(Xc_train, avgs, oris)
    train_err = np.sum(fit != labels[train_idxs]) / len(fit)

    pre = covar_classify(Xc_test, avgs, oris)
    err = np.sum(pre != labels[test_idxs]) / len(pre)

    conf = metrics.confusion_matrix(labels[test_idxs], pre, labels=oris)

    minfo = mutual_info(conf)
    print("mutual information:", minfo)

    get_ipython().run_line_magic('matplotlib', 'inline')
    plt.figure()
    plt.imshow(conf)
    ax = plt.gca()
    ax.xaxis.tick_top()
    ax.set_xticks(np.arange(len(oris)))
    ax.set_yticks(np.arange(len(oris)))
    ax.set_xticklabels(oris)
    ax.set_yticklabels(oris)
    plt.colorbar()
    plt.show()

    if return_minfo:
        return train_err, err, minfo
    return train_err, err
# In[536]:
train_err, err = covar_evaluate(Xc_train, Xc_test, labels, train_idxs,
    test_idxs, avgs, oris)
print("train error:", train_err, " test error:", err)

# ## max likelihood classification
# In[537]:
def max_likelihood_classify(Xc, avgs, stds, oris):
    pre = []
    for response in Xc:
        nlls = []
        for avg, std in zip(avgs, stds):
            nlls.append(np.mean((avg - response)**2 / (2*(std + 0.01)**2) +
                np.log(std + 0.01)))
        ori_idx = np.argmin(nlls)
        pre.append(oris[ori_idx])

    return pre
```

```
# In[588]:
def max_likelihood_evaluate(Xc_train, Xc_test, labels, train_idxs,
    test_idxs, avgs, stds, oris, return_minfo=False):
    fit = max_likelihood_classify(Xc_train, avgs, stds, oris)
    train_err = np.sum(fit != labels[train_idxs]) / len(fit)

    pre = max_likelihood_classify(Xc_test, avgs, stds, oris)
    err = np.sum(pre != labels[test_idxs]) / len(pre)

    conf = metrics.confusion_matrix(labels[test_idxs], pre, labels=oris)

    minfo = mutual_info(conf)
    print("mutual information:", minfo)

    get_ipython().run_line_magic('matplotlib', 'inline')
    plt.figure()
    plt.imshow(conf)
    ax = plt.gca()
    ax.xaxis.tick_top()
    ax.set_xticks(np.arange(len(oris)))
    ax.set_yticks(np.arange(len(oris)))
    ax.set_xticklabels(oris)
    ax.set_yticklabels(oris)
    plt.colorbar()
    plt.show()

    if return_minfo:
        return train_err, err, minfo
    return train_err, err
# In[539]:
train_err, err = max_likelihood_evaluate(Xc_train, Xc_test, labels,
    train_idxs, test_idxs, avgs, stds, oris)
print("train error:", train_err, " test error:", err)

# ## SVM
# - compare with the STA method
# - might try normalizing it so no unit is too dominant
# In[540]:
from sklearn import svm
from sklearn import metrics
# In[587]:
def svm_evaluate(train, test, labels, train_idxs, test_idxs, oris,
    return_minfo=False):
    clf = svm.SVC(decision_function_shape="ovo")
    clf.fit(train, labels[train_idxs])

    fit = clf.predict(train)
```

```
    train_err = np.sum(fit != labels[train_idxs]) / len(fit)

    pre = clf.predict(test)
    err = np.sum(pre != labels[test_idxs]) / len(pre)

    conf = metrics.confusion_matrix(labels[test_idxs], pre, labels=oris)

    minfo = mutual_info(conf)
    print("mutual information:", minfo)

    get_ipython().run_line_magic('matplotlib', 'inline')
    plt.figure()
    plt.imshow(conf)
    ax = plt.gca()
    ax.xaxis.tick_top()
    ax.set_xticks(np.arange(len(oris)))
    ax.set_yticks(np.arange(len(oris)))
    ax.set_xticklabels(oris)
    ax.set_yticklabels(oris)
    plt.colorbar()
    plt.show()

    if return_minfo:
        return train_err, err, minfo
    return train_err, err
# In[542]:
train_err, err = svm_evaluate(train, test, labels, train_idxs, test_idxs,
    oris)
print("train error:", train_err, " test error:", err)

# # Classification of all 6000 gratings
# ## Get data PCA projections
# In[553]:
num_bins = 10
stim_duration = stims.duration.min() # minimum so that none of the
    recording periods are overlapping
bin_edges = np.linspace(0, stim_duration, num_bins + 1)
counts = session.presentationwise_spike_counts(bin_edges=bin_edges,
                                               stimulus_presentation_ids=stims.index,
                                               unit_ids=unit_ids)
# In[554]:
Xall = np.reshape(np.array(counts), (counts.shape[0], num_bins *
    len(unit_ids)))
labels = np.array(list(stims.orientation))

# train test split
train_frac = 0.7
```

```
train_idxs = np.array([], dtype=int)
test_idxs = np.array([], dtype=int)
for label in np.unique(labels):
    matches = np.nonzero(labels == label)[0]
    np.random.shuffle(matches)
    train_idxs = np.concatenate([train_idxs, matches[:int(train_frac *
        len(matches))]])
    test_idxs = np.concatenate([test_idxs, matches[int(train_frac *
        len(matches)):]])
# In[555]:
Xc_train = Xall[train_idxs] - np.mean(Xall[train_idxs], axis=0,
    keepdims=True)
Xc_test = Xall[test_idxs] - np.mean(Xall[test_idxs], axis=0, keepdims=True)
U, S, Vh = np.linalg.svd(Xc_train, full_matrices=False)
# In[556]:
proj_rank = 20
train = (np.diag(S[:proj_rank]) @ U.T[:proj_rank]).T
test = Xc_test @ Vh.T[:, :proj_rank]
# In[557]:
get_ipython().run_line_magic('matplotlib', 'notebook')
sample_size = 300
fig = plt.figure(dpi=120)
ax = fig.gca(projection='3d')
plt.title("Training set responses projected onto PCA space")
for ori in oris:
    ori_ims = train[np.nonzero(labels[train_idxs]==ori)[0],
        :3][:sample_size // len(oris)]
    ax.scatter(ori_ims[:, 0], ori_ims[:, 1], ori_ims[:, 2],
        label=f"{ori}$^\circ$")
ax.view_init(elev=15, azim=50)
plt.xlabel("1st principle comp.")
plt.ylabel("2nd principle comp.")
ax.set_zlabel("3rd principle comp.")
plt.legend()
plt.show()
# In[558]:
plt.figure()
plt.scatter(np.arange(len(S)), S)
plt.xlabel("index")
plt.ylabel("$\sigma$")
plt.title("singular value spectrum")
plt.show()

# # covariance classification
# In[560]:
avgs = []
stds = []
```

```
train_avg = np.mean(Xc_train, axis=0)
for ori in oris:
    idxs = np.nonzero(labels[train_idxs]==ori)
    avgs.append(np.mean(Xc_train[idxs] - train_avg, axis=0))
    stds.append(np.std(Xc_train[idxs] - train_avg, axis=0))

# In[561]:
train_err, err = covar_evaluate(Xc_train, Xc_test, labels, train_idxs,
    test_idxs, avgs, oris)
print("train error:", train_err, " test error:", err)

# ## max likelihood classification
# In[562]:
train_err, err = max_likelihood_evaluate(Xc_train, Xc_test, labels,
    train_idxs, test_idxs, avgs, stds, oris)
print("train error:", train_err, " test error:", err)

# ## SVM
# In[563]:
train_err, err = svm_evaluate(train, test, labels, train_idxs, test_idxs,
    oris)
print("train error:", train_err, " test error:", err)

# # Natural Scene Image Classification
# ## get data and PCA projections
# In[566]:
stim_epochs = session.get_stimulus_epochs()
stim_epochs = stim_epochs[stim_epochs.stimulus_name=="natural_scenes"]
stim_epochs
# In[567]:
stims = session.get_stimulus_table(["natural_scenes"])
stims
# In[568]:
label_set = np.random.choice(np.arange(118), 6, replace=False)
filtered_stims = stims[list(frame in label_set for frame in stims.frame)]
filtered_stims.shape
# In[569]:
label_set
# Each image was shown 50 times
# In[570]:
num_bins = 10
stim_duration = filtered_stims.duration.min() # minimum so that none of
    the recording periods are overlapping
bin_edges = np.linspace(0, stim_duration, num_bins + 1)
counts = session.presentationwise_spike_counts(bin_edges=bin_edges,
                                               stimulus_presentation_ids=filtered_stims.index,
                                               unit_ids=unit_ids)
```

```
# In[571]:
Xall = np.reshape(np.array(counts), (counts.shape[0], num_bins *
    len(unit_ids)))
labels = np.array(list(filtered_stims.frame))
# train test split
train_frac = 0.7
train_idxs = np.array([], dtype=int)
test_idxs = np.array([], dtype=int)
for label in label_set:
    matches = np.nonzero(labels == label)[0]
    np.random.shuffle(matches)
    train_idxs = np.concatenate([train_idxs, matches[:int(train_frac *
        len(matches))]])
    test_idxs = np.concatenate([test_idxs, matches[int(train_frac *
        len(matches)):]])
# In[572]:
Xc_train = Xall[train_idxs] - np.mean(Xall[train_idxs], axis=0,
    keepdims=True)
Xc_test = Xall[test_idxs] - np.mean(Xall[test_idxs], axis=0, keepdims=True)
U, S, Vh = np.linalg.svd(Xc_train, full_matrices=False)
# In[573]:
proj_rank = 20
train = (np.diag(S[:proj_rank]) @ U.T[:proj_rank]).T
test = Xc_test @ Vh.T[:, :proj_rank]
# In[574]:
get_ipython().run_line_magic('matplotlib', 'notebook')
sample_size = 2000
fig = plt.figure(dpi=120)
ax = fig.gca(projection='3d')
plt.title("Training set responses projected onto PCA space")
# for ori in oris[[1,3]]:
for lab in label_set:
    lab_resp = train[np.nonzero(labels[train_idxs]==lab)[0],
        :3][:sample_size // len(label_set)]
    ax.scatter(lab_resp[:, 0], lab_resp[:, 1], lab_resp[:, 2],
        label=f"{lab}")
ax.view_init(elev=15, azim=50)
plt.xlabel("1st principle comp.")
plt.ylabel("2nd principle comp.")
ax.set_zlabel("3rd principle comp.")
plt.legend()
plt.show()
# In[576]:
get_ipython().run_line_magic('matplotlib', 'inline')
plt.figure()
plt.scatter(np.arange(len(S)), S)
plt.xlabel("index")
```

```
plt.ylabel("$\sigma$")
plt.title("singular value spectrum")
plt.show()
# In[577]:
S[183], S[184]

# ## covariance classification
# In[578]:
avgs = []
stds = []
train_avg = np.mean(Xc_train, axis=0)
for lab in label_set:
    idxs = np.nonzero(labels[train_idxs]==lab)
    avgs.append(np.mean(Xc_train[idxs] - train_avg, axis=0))
    stds.append(np.std(Xc_train[idxs] - train_avg, axis=0))
# In[579]:
train_err, err = covar_evaluate(Xc_train, Xc_test, labels, train_idxs,
    test_idxs, avgs, label_set)
print("train error:", train_err, " test error:", err)

# ## max likelihood classification
# In[580]:
train_err, err = max_likelihood_evaluate(Xc_train, Xc_test, labels,
    train_idxs, test_idxs, avgs, stds, label_set)
print("train error:", train_err, " test error:", err)

# ## SVM classification
# In[581]:
train_err, err = svm_evaluate(train, test, labels, train_idxs, test_idxs,
    label_set)
print("train error:", train_err, " test error:", err)

# # Classification accuracy versus number of images
# In[582]:
nums_included = np.logspace(0.4, 2.07, 10, base=10).astype(int)
nums_included
# In[589]:
errs_cov = []
errs_ml = []
errs_svm = []
minfos = []

for num_included in nums_included:
    print("NUM INCLUDED ------- ", num_included)
    label_set = np.random.choice(np.arange(118), num_included,
        replace=False)
```

```
filtered_stims = stims[list(frame in label_set for frame in
    stims.frame)]

stim_duration = filtered_stims.duration.min() # minimum so that none of
    the recording periods are overlapping
bin_edges = np.linspace(0, stim_duration, num_bins + 1)
counts = session.presentationwise_spike_counts(bin_edges=bin_edges,
                                               stimulus_presentation_ids=filtered_stims.index,
                                               unit_ids=unit_ids)

Xall = np.reshape(np.array(counts), (counts.shape[0], num_bins *
    len(unit_ids)))
labels = np.array(list(filtered_stims.frame))

# train test split
train_frac = 0.7
train_idxs = np.array([], dtype=int)
test_idxs = np.array([], dtype=int)
for label in label_set:
    matches = np.nonzero(labels == label)[0]
    np.random.shuffle(matches)
    train_idxs = np.concatenate([train_idxs, matches[:int(train_frac *
        len(matches))]])
    test_idxs = np.concatenate([test_idxs, matches[int(train_frac *
        len(matches)):]])

Xc_train = Xall[train_idxs] - np.mean(Xall[train_idxs], axis=0,
    keepdims=True)
Xc_test = Xall[test_idxs] - np.mean(Xall[test_idxs], axis=0,
    keepdims=True)
U, S, Vh = np.linalg.svd(Xc_train, full_matrices=False)

proj_rank = 20
train = (np.diag(S[:proj_rank]) @ U.T[:proj_rank]).T
test = Xc_test @ Vh.T[:, :proj_rank]

avgs = []
stds = []
train_avg = np.mean(Xc_train, axis=0)
for lab in label_set:
    idxs = np.nonzero(labels[train_idxs]==lab)
    avgs.append(np.mean(Xc_train[idxs] - train_avg, axis=0))
    stds.append(np.std(Xc_train[idxs] - train_avg, axis=0))

print("COVARIANCE")
train_err, err, minfo_cov = covar_evaluate(Xc_train, Xc_test, labels,
    train_idxs, test_idxs, avgs, label_set, return_minfo=True)
```

```
    errs_cov.append(err)
    print("train error:", train_err, " test error:", err)

    print("MAX LIKELIHOOD")
    train_err, err, minfo_ml = max_likelihood_evaluate(Xc_train, Xc_test,
        labels, train_idxs, test_idxs, avgs, stds, label_set,
        return_minfo=True)
    errs_ml.append(err)
    print("train error:", train_err, " test error:", err)

    print("SVM")
    train_err, err, minfo_svm = svm_evaluate(train, test, labels,
        train_idxs, test_idxs, label_set, return_minfo=True)
    errs_svm.append(err)
    print("train error:", train_err, " test error:", err)

    minfos.append(max(minfo_cov, minfo_ml, minfo_svm))
# In[596]:
plt.figure()
plt.plot(nums_included, errs_cov, label="covar")
plt.plot(nums_included, errs_ml, label="max lkl")
plt.plot(nums_included, errs_svm, label="SVM")
plt.ylim([0, 1])
plt.ylabel("error rate")
plt.xlabel("number of classes")
plt.title("natural image classification performance")
plt.semilogx(base=2)
plt.legend()
plt.show()

# In[595]:
plt.figure()
plt.plot(nums_included, minfos)
plt.ylim([0, 4])
plt.ylabel("mutual information (bits)")
plt.xlabel("number of classes")
plt.title("natural image response mutual info.")
plt.semilogx(base=2)
plt.show()


# # Analysis of redundancy
# pca on the overall average response to natural movies

stim_epochs = session.get_stimulus_epochs()
stim_epochs = stim_epochs[stim_epochs.stimulus_name=="natural_movie_three"]
stim = session.get_stimulus_table(['natural_movie_three'])
```

```
stim_pres_ids = stim.index

# In[224]:
times =
    session.presentationwise_spike_times(stimulus_presentation_ids=stim_pres_ids,
    # is this right? todo
                                         unit_ids=unit_ids)
# In[239]:
num_bins = 3000
# stim_length = 30 # seconds for natural movie one
stim_length = 120 # seconds for natural movie three
stim_range = np.array([0, stim_length])

trials = np.zeros((20, num_bins, len(unit_ids))) # stimulus was presented
    20 times
bin_edges = np.linspace(0, stim_length, num_bins + 1)
bin_centers = (bin_edges[1:] + bin_edges[:-1]) / 2
bin_width = bin_edges[1] - bin_edges[0]

for j, unit_id in enumerate(unit_ids):
    # 1st block where nat 1 is shown, it is shown 10 times in each block
    presentations1 = stim_epochs.iloc[0].start_time + np.linspace(0,
        stim_epochs.iloc[0].duration, 10)
    # 2nd block
    presentations2 = stim_epochs.iloc[1].start_time + np.linspace(0,
        stim_epochs.iloc[1].duration, 10)
    presentations = np.concatenate([presentations1, presentations2])
    for i, start in enumerate(presentations):
        times_ij = times[(times.unit_id==unit_id) & (start < times.index) &
            (times.index < (start + stim_length))].index
        unit_counts, _ = np.histogram(times_ij, num_bins, range=start +
            stim_range)
        trials[i, :, j] = unit_counts

# In[240]:
avg_resp = trials.mean(axis=0)
avg_resp_c = avg_resp - avg_resp.mean(axis=0, keepdims=True)
# In[241]:
U, S, Vh = np.linalg.svd(avg_resp_c, full_matrices=False)
# In[252]:
cov = avg_resp_c.T @ avg_resp_c / (avg_resp_c.shape[1] - 1)
plt.imshow(cov)
plt.title("covariance: 120 sec movie")
plt.ylabel("unit")
plt.xlabel("unit")
plt.xticks(range(0, 60 ,10))
plt.show()
```

```
# In[244]:
plt.plot(avg_resp[:, [27, 52]] * num_bins / stim_length)
ticks = np.linspace(0, num_bins, 5).astype(int)
plt.xticks(ticks)
ax = plt.gca()
ax.set_xticklabels(np.round(bin_edges[ticks], 3))
plt.ylabel("firing rate (Hz)")
plt.xlabel("time from start of video (s)")
plt.title("correlated responses")
plt.legend(title="cell", labels=[52, 27], loc="upper right")
plt.ylim([5, 85])
# In[245]:
variances = S**2 / (avg_resp_c.shape[1] - 1)
cumul_var = sum(variances)
cumul_vars = [0]
rank_hat = None
for i, var in enumerate(variances):
    cumul_vars.append(var + cumul_vars[-1])
    if rank_hat is None and cumul_vars[-1] / cumul_var > 0.9:
        rank_hat = i
# In[246]:
plt.plot(cumul_vars / cumul_vars[-1], ".")
plt.xlabel("rank (number of PCA units)")
plt.hlines(0.9, 0, rank_hat, linestyle=":", color="k")
plt.vlines(rank_hat, 0, 0.9, linestyle="--", color="k")
plt.ylabel("$\sigma^2$")
plt.title("explained variance")
plt.show()
# In[248]:
avg_resp[:, 27].mean()
# ## robustness of PCA ran
# In[236]:
nums_bins = np.array([50, 100, 200, 350, 500, 750, 1000, 1500, 2000, 5000,
    10000, 20000, 50000])
rank_hats = []
# In[237]:
for num_bins in nums_bins:
    print("NUM BINS", num_bins)
    stim_length = 30 # seconds for natural movie one
    stim_range = np.array([0, stim_length])

    trials = np.zeros((20, num_bins, len(unit_ids))) # stimulus was
        presented 20 times
    bin_edges = np.linspace(0, stim_length, num_bins + 1)
    bin_centers = (bin_edges[1:] + bin_edges[:-1]) / 2
    bin_width = bin_edges[1] - bin_edges[0]
```

```
    for j, unit_id in enumerate(unit_ids):
        # 1st block where nat 1 is shown, it is shown 10 times in each block
        presentations1 = stim_epochs.iloc[0].start_time + np.linspace(0,
            stim_epochs.iloc[0].duration, 10)
        # 2nd block
        presentations2 = stim_epochs.iloc[1].start_time + np.linspace(0,
            stim_epochs.iloc[1].duration, 10)
        presentations = np.concatenate([presentations1, presentations2])
        for i, start in enumerate(presentations):
            times_ij = times[(times.unit_id==unit_id) & (start <
                times.index) & (times.index < (start + stim_length))].index
            unit_counts, _ = np.histogram(times_ij, num_bins, range=start +
                stim_range)
            trials[i, :, j] = unit_counts

    avg_resp = trials.mean(axis=0)
    avg_resp_c = avg_resp - avg_resp.mean(axis=0, keepdims=True)

    U, S, Vh = np.linalg.svd(avg_resp_c, full_matrices=False)

    variances = S**2 / (avg_resp_c.shape[1] - 1)
    cumul_var = sum(variances)
    cumul_vars = [0]
    rank_hat = None
    for i, var in enumerate(variances):
        cumul_vars.append(var + cumul_vars[-1])
        if rank_hat is None and cumul_vars[-1] / cumul_var > 0.9:
            rank_hat = i
    rank_hats.append(rank_hat)
# In[250]:
plt.plot(1000 * 30 / nums_bins, rank_hats)
plt.xlabel("length of time bins (ms)")
plt.title("rank robustness: 120 sec movie")
plt.semilogx()
plt.ylabel("PCA-estimated rank")

# the correlated units are located far apart
# In[188]:
unit52 = units[units.index == unit_ids[52]]
# In[193]:
unit27 = units[units.index == unit_ids[27]]
# In[194]:
print(unit27['anterior_posterior_ccf_coordinate'])
print(unit27['dorsal_ventral_ccf_coordinate'])
unit27['left_right_ccf_coordinate']
# In[189]:
print(unit52['anterior_posterior_ccf_coordinate'])
```

```
print(unit52['dorsal_ventral_ccf_coordinate'])
unit52['left_right_ccf_coordinate']
```

References

[1] Mallen, Alex Troy. GitHub source code.
https://github.com/AlexTMallen/AMATH-582/tree/master/allenSDK

[2] Mallen, Alex Troy. SVMs, LDA, and Decision Trees in PCA Space to Recognize MNIST Digits.
https://github.com/AlexTMallen/AMATH-582/tree/master/HW4

[3] Mallen, Alex Troy. Dimensionality Reduction of Oscillator Dynamics.
https://github.com/AlexTMallen/AMATH-582/tree/master/HW3

[4] Stefano Recanatesi, Serena Bradde, Vijay Balasubramanian, Nicholas A Steinmetz, and Eric Shea-Brown, A scale-dependent measure of system dimensionality, bioRxiv 2020.12.19.423618;
https://doi.org/10.1101/2020.12.19.423618

[5] Kutz, Jose Nathan. Data Analysis: Clustering and Classification (Lec. 3, part 1)
https://www.youtube.com/watch?v=YWTzFPZgYCA

[6] Scikit learn SVM.
https://scikit-learn.org/stable/modules/svm.html

[7] Scikit-learn: Machine Learning in Python. Pedregosa, F. and Varoquaux, G. and Gramfort, A. and Michel, V. and Thirion, B. and Grisel, O. and Blondel, M. and Prettenhofer, P. and Weiss, R. and Dubourg, V. and Vanderplas, J. and Passos, A. and Cournapeau, D. and Brucher, M. and Perrot, M. and Duchesnay, E. Journal of Machine Learning Research, 12, 2825–2830, 2011

[8] Natural Images Documentation.
http://observatory.brain-map.org/visualcoding/stimulus/natural_scenes

[9] Dayan, Peter, and L. F. Abbott. Theoretical Neuroscience. *MIT Press*. Chapter I4: Information Theory.

[10] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner. "Gradient-based learning applied to document recognition." Proceedings of the IEEE, 86(11):2278-2324, November 1998.