

Neural Networks – Code

Line Kruse - 201608877

Study Group 6

```
1
2 #####
3 # LOAD DATA AND PREPARE TRAINING AND TEST SETS
4 #####
5 from __future__ import print_function
6 import keras
7 from keras.datasets import mnist
8 from keras.models import Sequential
9 from keras.layers import Dense, Dropout, Flatten
10 from keras.layers import Conv2D, MaxPooling2D
11 from keras import backend as K
12 import matplotlib.pyplot as plt
13
14 batch_size = 128
15 num_classes = 10
16 epochs = 5 # Original code has 12 epochs
17
18 # input image dimensions
19 img_rows, img_cols = 28, 28
20
21 # the data, split between train and test sets
22 (x_train, y_train), (x_test, y_test) = mnist.load_data()
23 print(x_train.shape)
24
25 if K.image_data_format() == 'channels_first':
26     x_train = x_train.reshape(x_train.shape[0], 1, img_rows, img_cols)
27     x_test = x_test.reshape(x_test.shape[0], 1, img_rows, img_cols)
28     input_shape = (1, img_rows, img_cols)
29 else:
30     x_train = x_train.reshape(x_train.shape[0], img_rows, img_cols, 1)
31     x_test = x_test.reshape(x_test.shape[0], img_rows, img_cols, 1)
32     input_shape = (img_rows, img_cols, 1)
33
34 x_train = x_train.astype('float32')
35 x_test = x_test.astype('float32')
36 x_train /= 255
37 x_test /= 255
38
39 #Take 5000 out of 60000 subset of training data to improve speed
40 x_train = x_train[0:5000, :, :, :]
41 y_train = y_train[0:5000]
42 print(y_train[0:10])
43
44
45 #Take 1000 out of 10000 subset of test data to improve speed
46 x_test = x_test[0:1000, :, :, :]
47 y_test = y_test[0:1000]
48 print(y_test[0:10])
49
50 print('x_train shape:', x_train.shape)
51 print(x_train.shape[0], 'train samples')
52 print(x_test.shape[0], 'test samples')
53
```

```

54 # convert class label vectors to binary class matrices
55 y_train = keras.utils.to_categorical(y_train, num_classes)
56 y_test = keras.utils.to_categorical(y_test, num_classes)
57 print(y_train[0])
58
59
60
61 #####
62 #####
63 #CREATE MODEL 1 - FULLY CONNECTED FEEDFORWARD NETWORK
64 #Loss function: categorical_crossentropy
65 #Optimizer: SGD
66
67 x_train_flat = x_train.reshape((x_train.shape[0], -1))
68 x_test_flat = x_test.reshape((x_test.shape[0], -1))
69
70 model = Sequential()
71 model.add(Dense(32, input_shape=(x_train_flat.shape[-1],),
72 activation='relu'))
73 model.add(Dense(64, activation='relu'))
74 model.add(Dense(32, activation='relu'))
75 model.add(Dense(num_classes, activation='softmax'))
76
77 model.compile(loss=keras.losses.categorical_crossentropy,
78 optimizer=keras.optimizers.Adadelta(),
79 metrics=['accuracy'])
80 model.summary()
81
82 #TRAIN AND EVALUATE MODEL 1
83 model.fit(x_train_flat, y_train,
84 batch_size=batch_size,
85 epochs=epochs,
86 verbose=1,
87 validation_data=(x_test_flat, y_test))
88 score = model.evaluate(x_test_flat, y_test, verbose=0)
89 print('Test loss:', score[0])
90 print('Test accuracy:', score[1])
91
92 #####
93 #####
94 #CREATE MODEL 2 - CONVOLUTIONAL NETWORK
95 #Loss function: categorical_crossentropy
96 #Optimizer: SGD
97
98 model = Sequential()
99 model.add(Conv2D(32, kernel_size=(3, 3),
100 activation='relu',
101 input_shape=input_shape))
102 model.add(Conv2D(64, (3, 3), activation='relu'))
103 model.add(MaxPooling2D(pool_size=(2, 2)))
104 #model.add(Dropout(0.25)) #To prevent overfitting
105 model.add(Flatten()) #To input to fully connected layer
106 model.add(Dense(32, activation='relu')) #Fully connected layer
107 #model.add(Dropout(0.5)) #To avoid overfitting

```

```

105 #model.add(Dropout(0.5)) #to avoid overfitting
106 model.add(Dense(num_classes, activation='softmax')) #Output layer
107
108 model.compile(loss=keras.losses.categorical_crossentropy,
109               optimizer=keras.optimizers.Adadelta(),
110               metrics=['accuracy'])
111 model.summary()
112
113 #TRAIN AND EVALUATE MODEL 2
114 model.fit(x_train_flat, y_train,
115           batch_size=batch_size,
116           epochs=epochs,
117           verbose=1,
118           validation_data=(x_test_flat, y_test))
119 score = model.evaluate(x_test_flat, y_test, verbose=0)
120 print('Test loss:', score[0])
121 print('Test accuracy:', score[1])
122
123
124
125 #####
126 #####
127 #COMPARE OPTIMIZERS
128
129 #MODEL 2 - Fully connected feedforward neural network:
130 #Loss function: categorical_crossentropy
131 #Optimizer: Adadelta
132
133 x_train_flat = x_train.reshape((x_train.shape[0], -1))
134 x_test_flat = x_test.reshape((x_test.shape[0], -1))
135
136 model = Sequential()
137 model.add(Dense(32, input_shape=(x_train_flat.shape[-1],),
138                               activation='relu'))
139 model.add(Dense(64, activation='relu'))
140 model.add(Dense(32, activation='relu'))
141 model.add(Dense(num_classes, activation='softmax'))
142
143 model.compile(loss=keras.losses.categorical_crossentropy,
144               optimizer=keras.optimizers.Adadelta(),
145               metrics=['accuracy'])
146 model.summary()
147
148 #Train and evaluate model
149 model.fit(x_train_flat, y_train,
150           batch_size=batch_size,
151           epochs=epochs,
152           verbose=1,
153           validation_data=(x_test_flat, y_test))
154 score = model.evaluate(x_test_flat, y_test, verbose=0)
155 print('Test loss:', score[0])
156 print('Test accuracy:', score[1])
157
158 #####

```

```
150 #####
159 #####
160 #MODEL 4 - Convolutional neural network:
161 #Loss function: categorical_crossentropy
162 #Optimizer: Adadelta
163 model = Sequential()
164 model.add(Conv2D(32, kernel_size=(3, 3),
165                 activation='relu',
166                 input_shape=input_shape))
167 model.add(Conv2D(64, (3, 3), activation='relu'))
168 model.add(MaxPooling2D(pool_size=(2, 2)))
169 #model.add(Dropout(0.25)) #To prevent overfitting
170 model.add(Flatten()) #To input to fully connected layer
171 model.add(Dense(32, activation='relu')) #Fully connected layer
172 #model.add(Dropout(0.5)) #To avoid overfitting
173 model.add(Dense(num_classes, activation='softmax')) #Output layer
174
175 model.compile(loss=keras.losses.categorical_crossentropy,
176               optimizer=keras.optimizers.Adadelta(),
177               metrics=['accuracy'])
178 model.summary()
179
180 #TRAIN AND EVALUATE MODEL 3
181 model.fit(x_train_flat, y_train,
182         batch_size=batch_size,
183         epochs=epochs,
184         verbose=1,
185         validation_data=(x_test_flat, y_test))
186 score = model.evaluate(x_test_flat, y_test, verbose=0)
187 print('Test loss:', score[0])
188 print('Test accuracy:', score[1])
189
190
191
192
193
194
195
```

MEG Analysis – Code

Study Group 6

```

1  """
2  MEG analysis - Classification algorithms
3
4  @author: Martin, Christoffer, Line and Simon (Study Group 6)
5  """
6
7  # Import libraries
8  import numpy as np
9  import pandas as pd
10 from sklearn.naive_bayes import GaussianNB
11 from sklearn.model_selection import cross_val_score
12 from sklearn.svm import SVC
13 from sklearn.neighbors import KNeighborsClassifier
14 import matplotlib.pyplot as plt
15 from sklearn.pipeline import make_pipeline
16 from sklearn.preprocessing import StandardScaler
17 import statsmodels.stats.api as sms
18
19 # Import data and labels
20 labels = np.load('pos_neg_img_labels.npy')
21 data = np.load('pos_neg_img_trials.npy')
22
23 # Loop through each sample and perform cross-validation (linear SVM)
24 n_samples = data.shape[2] # Get number of samples
25 index=np.arange(0)
26 Classification = pd.DataFrame(columns = ["Sample", "Score", "SD", "CI_low",
27 "CI_high"], index = index) # Create empty dataframe
28
29 for sample_index in range(n_samples):
30     temp = data[:, :, sample_index] # Use all datapoints for the given sample index
31     clf = SVC(kernel="linear", C=1) # Linear suport vector classifier
32     clf = make_pipeline(StandardScaler(copy=False), clf) #Standardize data
33     cv_score = cross_val_score(clf, temp, labels, cv=10) #perform 10-fold cross-
34 validation
35     mean_cv = np.mean(cv_score) # Mean
36     sd_cv = np.std(cv_score) # Standard Deviation
37     CI_low, CI_high=sms.DescrStatsW(cv_score).tconfint_mean() # 95 % Confidence
38 intervals
39     Classification = Classification.append({"Sample": sample_index, "Score":
40 mean_cv, "SD": sd_cv, "CI_low": CI_low, "CI_high": CI_high}, ignore_index = True) #
41 Append to dataframe
42
43 # Create a preliminary plot
44 plt.plot(Classification["Sample"], Classification["Score"])
45 plt.show()
46
47 # Save results csv file
48 Classification.to_csv('linearSVM.csv')
49
50 # Loop through each sample and perform cross-validation (GaussianNB)
51 n_samples = data.shape[2]
52 index=np.arange(0)
53 Classification = pd.DataFrame(columns = ["Sample", "Score", "SD", "CI_low",
54 "CI_high"], index = index)
55
56 for sample_index in range(n_samples):
57     temp = data[:, :, sample_index]
58     clf = GaussianNB()
59     clf = make_pipeline(StandardScaler(copy=False), clf)
60     cv_score = cross_val_score(clf, temp, labels, cv=10)
61     mean_cv = np.mean(cv_score)

```

```

56     sd_cv = np.std(cv_score)
57     CI_low, CI_high=sms.DescrStatsW(cv_score).tconfint_mean()
58     Classification = Classification.append({"Sample": sample_index, "Score":
mean_cv, "SD": sd_cv, "CI_low": CI_low, "CI_high": CI_high}, ignore_index = True)
59
60 # Make preliminary plot
61 plt.plot(Classification["Sample"], Classification["Score"])
62 plt.show()
63
64 # Save results csv file
65 Classification.to_csv('GaussianNB.csv')
66
67 # Loop through each sample and perform cross-validation (K-nearest neighbour )
68 n_samples = data.shape[2]
69 index=np.arange(0)
70 Classification = pd.DataFrame(columns = ["Sample", "Score", "SD", "CI_low",
"CI_high"], index = index)
71
72 for sample_index in range(n_samples):
73     temp = data[:, :, sample_index]
74     clf = KNeighborsClassifier(3)
75     clf = make_pipeline(StandardScaler(copy=False), clf)
76     cv_score = cross_val_score(clf, temp, labels, cv=10)
77     mean_cv = np.mean(cv_score)
78     sd_cv = np.std(cv_score)
79     CI_low, CI_high=sms.DescrStatsW(cv_score).tconfint_mean()
80     Classification = Classification.append({"Sample": sample_index, "Score":
mean_cv, "SD": sd_cv, "CI_low": CI_low, "CI_high": CI_high}, ignore_index = True)
81
82 # Make preliminary plots
83 plt.plot(Classification["Sample"], Classification["Score"])
84 plt.show()
85
86 # Save results as csv file
87 Classification.to_csv('Knearest(3).csv')
88
89 """
90 MEG analysis - Plots
91
92 @author: Martin, Christoffer, Line and Simon (Study Group 6) (Adapted script from
Lau)
93 """
94
95 # Import relevant libraries
96 from os.path import join ## connects strings into filepaths
97 import mne
98 import matplotlib.pyplot as plt # for plotting control
99 import numpy as np
100
101 # Set data path
102 data_path = 'C:\\Users\\Simon\\Documents\\MEG_numpy\\ME'
103
104 # Choose what subjects to analyse
105 subjects = [
106     #
107     'Group1',
108     'Group2', # noisy: change rejection parameters
109     'Group3',
110     'Group4',
111     'Group5',
112     'Group6',
113     'Group7'

```



```

113         ]
114
115     # Specify filename
116     filename = 'Session_70_Hz-ave.fif'
117
118     # Close all existing figures
119     plt.close('all')
120
121     # Loop over choosen subjects
122     for subject in subjects:
123         # Read in the evokeds
124         evokeds = mne.read_evokeds(join(data_path, subject, filename))
125         for evoked in evokeds: # loop through evokeds
126             # Pick the two stimulus classes ("img_pos" and "img_neg")
127             if evoked.comment == 'img_pos' or evoked.comment == 'img_neg':
128                 # Create butterfly plots
129                 evoked.plot(window_title=subject + '_' + evoked.comment)
130                 evoked.plot(spatial_colors=True, gfp=True, picks='meg')
131                 # Create topomaps
132                 evoked.plot_topomap(times='peaks', ch_type='mag', time_unit='s')
133
134
135     # Plot Global Field Power
136     conditions = ["img_pos", "img_neg"] # Specify conditon
137     evoked_dict = dict() # Create empty dictionary
138     for condition in conditions: # Loop through conditions
139         evoked_dict[condition.replace(" ", "/")] = mne.read_evokeds(
140             join(data_path, subject, filename), baseline=(None, 0), proj=True,
141             condition=condition) # Read in evokeds and append to dictionary
142     print(evoked_dict) # Print to check
143
144     colors = dict(img_pos="Crimson", img_neg="CornFlowerBlue") # Choose colours for each
145     condition
146
147     # Create the plot based on the evoked dictionary
148     mne.viz.plot_compare_evokeds(evoked_dict, colors=colors, split_legend=True)

```

```

1  ""
2  MEG analysis - Classification performance plots
3
4  @author: Martin, Christoffer, Line and Simon (Study Group 6)
5  ""
6
7  #----- Libraries -----
8  library(ggplot2)
9
10 #----- Paths -----
11 #Set working directory
12 setwd("C:\\Users\\Martin\\Documents\\UNI\\7th semester\\Advanced Neuro\\exam\\meg")
13
14 #Paths to data
15 NB_path = "GaussianNB.csv"
16 SVC_path = "SVCC1.csv"
17 KN_path = "KNearest(3).csv"
18
19 #----- Load data -----
20 #Naive bayes classifier
21 NB_data = read.csv(NB_path) #Read file
22 NB_data$Sample = NB_data$Sample-500 #move stimulus presentation to 0
23 NB_data[NB_data$Score == max(NB_data$Score),] #Find sample with max accuracy
24
25 #Linear support vector machine
26 SVC_data = read.csv(SVC_path) #Read file
27 SVC_data$Sample = SVC_data$Sample-500 #move stimulus presentation to 0
28 SVC_data[SVC_data$Score == max(SVC_data$Score),] #Find sample with max accuracy
29
30 #Knearest neighbours, 3
31 #Linear support vector machine
32 KN_data = read.csv(KN_path) #Read file
33 KN_data$Sample = KN_data$Sample-500 #move stimulus presentation to 0
34 KN_data[KN_data$Score == max(KN_data$Score),] #Find sample with max accuracy
35
36
37 #----- Plot data -----
38 #Naive bayes
39 ggplot(NB_data, aes(x = Sample, y = Score))+
40   scale_x_continuous(breaks=seq(-500,1000,100))+ #Fix x axis interval and increment
41   ylim(0.25, 1)+ #Y-axis limits
42   geom_ribbon(aes(ymin = CI_low, ymax = CI_high), fill = "grey", alpha = 0.5)+ #CI
43   geom_vline(xintercept = 0, color = "red")+ #Vertical line at x = 0
44   geom_line()+ #Line between scores
45   geom_hline(yintercept = 0.5, linetype = "dashed", color = "black")+ #Dashed line
   at chance level
46   labs(title = "Accuracy of Naive Bayes Classifier Over Time", #Labels
47         x = "Time after stimulus in ms",
48         y = "Accuracy")+
49   theme_minimal() #Theme
50
51 #Linear support vector machine
52 ggplot(SVC_data, aes(x = Sample, y = Score))+
53   scale_x_continuous(breaks=seq(-500,1000,100))+ #Fix x axis interval and increment
54   ylim(0.25, 1)+ #Y-axis limits
55   geom_ribbon(aes(ymin = CI_low, ymax = CI_high), fill = "grey", alpha = 0.5)+ #CI
56   geom_vline(xintercept = 0, color = "red")+ #Vertical line at x = 0
57   geom_line()+ #Line between scores
58   geom_hline(yintercept = 0.5, linetype = "dashed", color = "black")+ #Dashed line
   at chance level
59   labs(title = "Accuracy of Linear Support Vector Machine Over Time", #Labels

```

```

60     x = "Time after stimulus in ms",
61     y = "Accuracy")+
62     theme_minimal() #Theme
63
64 #K nearest neighbours, k=3
65 ggplot(KN_data, aes(x = Sample, y = Score))+
66     scale_x_continuous(breaks=seq(-500,1000,100))+ #Fix x axis interval and increment
67     ylim(0.25, 1)+ #Y-axis limits
68     geom_ribbon(aes(ymin = CI_low, ymax = CI_high), fill = "grey", alpha = 0.5)+ #CI
69     geom_vline(xintercept = 0, color = "red")+ #Vertical line at x = 0
70     geom_line()+ #Line between scores
71     geom_hline(yintercept = 0.5, linetype = "dashed", color = "black")+ #Dashed line
    at chance level
72     labs(title ="Accuracy of K Nearest Neighbour Classifier (K = 3) Over Time",
#Labels
73     x = "Time after stimulus in ms",
74     y = "Accuracy")+
75     theme_minimal() #Theme

```

fMRI Analysis – Code

Study Group 6

```

1  """
2  fMRI analysis - Classification algorithms
3
4  @author: Martin, Christoffer, Line and Simon (Study Group 6)
5  """
6
7  %matplotlib inline
8
9  #Adding data paths
10
11  #The fMRI data
12  fmri_filename='/Users/christoffer/Documents/CogSci/MA/Neuro/fMRI/subject064_r
    esults/beta4D.nii'
13
14  #Normalized Structural file
15  anat_filename='/Users/christoffer/Documents/CogSci/MA/Neuro/spm12/canonical/s
    ingle_subj_T1.nii'
16
17  #A whole brain mask
18  mask_wb_filename='/Users/christoffer/Documents/CogSci/MA/Neuro/fMRI/subject06
    4_results/mask.nii'
19
20  #The classification labels
21  class_filename='/Users/christoffer/Documents/CogSci/MA/Neuro/fMRI/scripts/cla
    ss_labels_faces.txt'
22
23
24
25
26  # Making training and test split
27
28  import pandas as pd
29  import numpy as np
30  #from nilearn import datasets
31  from nilearn.image import new_img_like, load_img, index_img, clean_img
32  from sklearn.model_selection import train_test_split
33
34  # Reshaping data-----
35  from nilearn.image import index_img, concat_imgs
36
37  #load fMRI data
38  fmri_img = load_img(fmri_filename)
39  #Print dimensions of data to get overview
40  print(fmri_img.shape)
41
42
43  #fmri_img=clean_img(fmri_img_raw, sessions=None, detrend=True,
    standardize=True, confounds=None, low_pass=None, high_pass=1/128, t_r=2,
    ensure_finite=False, mask_img=None)
44
45  #load csv-file with class labels
46  conditions = pd.read_csv(class_filename, sep=",")
47  conditions = conditions['labels']
48  #Print dimensions of data
49  print(conditions.shape)
50
51  #Make an index for splitting fMRI data with same size as class labels
52  idx=np.arange(conditions.shape[0])
53  # create training and testing vars on the basis of class labels

```

```

54 #the function train_test_split outputs 4 values --> we need to make sure the
names of the outputs are correct
55 idx1,idx2, conditions1, conditions2 = train_test_split(idx,conditions,
test_size=0.2)
56 print(idx1, idx2)
57
58 # Reshaping data-----
59 from nilearn.image import index_img
60 fmri_img1 = index_img(fmri_img, idx1)
61 fmri_img2 = index_img(fmri_img, idx2)
62 #Check data sizes
63 print(fmri_img1.shape)
64 print(fmri_img2.shape)
65
66
67
68 # SEARCHLIGHT
69 import pandas as pd
70 import numpy as np
71 from nilearn.image import new_img_like, load_img
72 from nilearn.plotting import plot_stat_map, plot_img, show
73 from nilearn import decoding
74 from nilearn.decoding import SearchLight
75 from sklearn import naive_bayes, model_selection #import GaussianNB
76 from sklearn.naive_bayes import GaussianNB
77
78 #####
79
80 #Load the whole brain mask
81 mask_img = load_img(mask_wb_filename)
82 process_mask = mask_img.get_data().astype(np.int)
83 process_mask_img = new_img_like(mask_img, process_mask)
84
85 #Plot the searchlight scores on an anatomical background
86 plot_img(process_mask_img, bg_img=anat_filename,#bg_img=mean_fmri,
87         title="Mask", display_mode="x",cut_coords=
88         [28,32,36,40,44,48,52,56,60,64,68,72],
89         vmin=.40, cmap='jet', threshold=0.9, black_bg=True)
90 #####
91
92 # USING NB
93 # The radius is the one of the Searchlight sphere that will scan the volume
94 searchlight = SearchLight(
95     mask_img,
96     estimator=GaussianNB(),
97     process_mask_img=process_mask_img,
98     radius=5, n_jobs=1,
99     verbose=1, cv=8)
100
101 searchlight.fit(fmri_img1, conditions1) #Run function on training data
(fmri_img1) for condition 1.
102
103 #USING SVM
104 # The radius is the one of the Searchlight sphere that will scan the volume
105 searchlight = SearchLight(
106     mask_img,
107     process_mask_img=process_mask_img,
108     radius=5, n_jobs=1,
109     verbose=1, cv=8)
110
111 searchlight.fit(fmri_img1, conditions1) #Run function on training data
(fmri_img1) for condition 1.

```

```

109
110 #USING k-NN
111 from sklearn import neighbors, datasets
112 searchlight = SearchLight(
113     mask_img,
114     estimator=neighbors.KNeighborsClassifier(3),
115     process_mask_img=process_mask_img,
116     radius=5, n_jobs=1,
117     verbose=1, cv=8)
118 searchlight.fit(fmri_img1, conditions1) #Run function on training data
    (fmri_img1) for condition 1.
119
120
121 # 500 BEST VOXELS
122 print(searchlight.scores_.size)
123 #Find the percentile that makes the cutoff for the 500 best voxels
124 perc=100*(1-500.0/searchlight.scores_.size)
125 #Print percentile
126 print(perc)
127 #Find the cutoff
128 cut=np.percentile(searchlight.scores_,perc)
129 #Print cutoff
130 print(cut)
131
132 #Make a mask using cutoff
133
134 #Load the whole brain mask
135 mask_img2 = load_img(mask_wb_filename)
136
137
138 # MAKE GLASS BRAIN PLOT
139 # .astype() makes a copy.
140 process_mask2 = mask_img2.get_data().astype(np.int)
141 process_mask2[searchlight.scores_<=cut] = 0
142 process_mask2_img = new_img_like(mask_img2, process_mask2)
143
144 from nilearn import image
145 from nilearn.plotting import plot_stat_map, plot_img, show
146 from nilearn import plotting
147 %matplotlib inline
148 #Create an image of the searchlight scores
149 searchlight_img = new_img_like(anat_filename, searchlight.scores_)
150 print(searchlight_img.shape)
151
152 #Plotting all voxels
153 plotting.plot_glass_brain(searchlight_img)
154
155 #Plotting 500 best voxels
156 plotting.plot_glass_brain(searchlight_img, threshold=cut)
157
158
159
160
161 #CLASSIFICATION
162 #using NB
163 from nilearn.input_data import NiftiMasker
164 masker = NiftiMasker(mask_img=process_mask2_img, standardize=False)
165
166 # We use masker to retrieve a 2D array ready
167 # for machine learning with scikit-learn

```

```

168 fmri_masked = masker.fit_transform(fmri_img2)
169 #Print size of matrix (images x voxels)
170 print(fmri_masked.shape)
171
172 from sklearn.naive_bayes import GaussianNB
173 from sklearn.model_selection import cross_val_score
174 cv_score = cross_val_score(GaussianNB(), fmri_masked, conditions2, cv=4)
175 print(cv_score)
176 print('Mean prediction score:')
177 print(np.mean(cv_score))
178
179 #Using SVM
180 from nilearn.input_data import NiftiMasker
181 masker = NiftiMasker(mask_img=process_mask2_img, standardize=False)
182
183 # We use masker to retrieve a 2D array ready
184 # for machine learning with scikit-learn
185 fmri_masked = masker.fit_transform(fmri_img2)
186 #Print size of matrix (images x voxels)
187 print(fmri_masked.shape)
188
189 from sklearn.naive_bayes import GaussianNB
190 from sklearn.model_selection import cross_val_score
191 from sklearn import svm
192 cv_score = cross_val_score(svm.SVC(kernel='linear', C=1), fmri_masked,
193                             conditions2, cv=4)
194 print(cv_score)
195 print('Mean prediction score:')
196 print(np.mean(cv_score))
197
198 #Using k-NN
199 from nilearn.input_data import NiftiMasker
200 masker = NiftiMasker(mask_img=process_mask2_img, standardize=False)
201
202 # We use masker to retrieve a 2D array ready
203 # for machine learning with scikit-learn
204 fmri_masked = masker.fit_transform(fmri_img2)
205 #Print size of matrix (images x voxels)
206 print(fmri_masked.shape)
207
208 from sklearn.naive_bayes import GaussianNB
209 from sklearn.model_selection import cross_val_score
210 cv_score = cross_val_score(neighbors.KNeighborsClassifier(3), fmri_masked,
211                             conditions2, cv=4)
212 print(cv_score)
213 print('Mean prediction score:')
214 print(np.mean(cv_score))
215
216 #PERMUTATION TESTS
217 #using NB
218 from sklearn.model_selection import permutation_test_score
219 score, permutation_scores, pvalue = permutation_test_score(
220     GaussianNB(), fmri_masked, conditions2, cv=4, n_permutations=100,
221     n_jobs=1, random_state=0, verbose=0, scoring=None)
222 print("Classification Accuracy: %s (pvalue : %s)" % (score, pvalue))
223
224 #using SVM
225 from sklearn.model_selection import permutation_test_score
226 score, permutation_scores, pvalue = permutation_test_score(

```



```

225     svm.SVC(kernel='linear', C=1), fmri_masked, conditions2, cv=4,
n_permutations=100,
226     n_jobs=1, random_state=0, verbose=0, scoring=None)
227 print("Classification Accuracy: %s (pvalue : %s)" % (score, pvalue))
228
229 #using k-NN
230 from sklearn.model_selection import permutation_test_score
231 score, permutation_scores, pvalue= permutation_test_score(
232     neighbors.KNeighborsClassifier(3), fmri_masked, conditions2, cv=4,
n_permutations=100,
233     n_jobs=1, random_state=0, verbose=0, scoring=None)
234 print("Classification Accuracy: %s (pvalue : %s)" % (score, pvalue))
235
236
237
238 #Plot permutation histogram
239 import numpy as np
240 import matplotlib.pyplot as plt
241 #How many classes
242 n_classes = np.unique(conditions2).size
243
244 plt.hist(permutation_scores, 20, label='Permutation scores',
245         edgecolor='black')
246 ylim = plt.ylim()
247 plt.plot(2 * [score], ylim, '--g', linewidth=3,
248         label='Classification Score'
249         ' (pvalue %s)' % pvalue)
250 plt.plot(2 * [1. / n_classes], ylim, '--k', linewidth=3, label='Chance
level')
251
252 plt.ylim(ylim)
253 plt.legend()
254 plt.xlabel('Score')
255 plt.show()

```