Neural Networks – Code

Line Kruse - 201608877

Study Group 6

```python

######################################################################################################
# LOAD DATA AND PREPARE TRAINING AND TEST SETS
######################################################################################################
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
import matplotlib.pyplot as plt

batch_size = 128
num_classes = 10
epochs = 5 # Original code has 12 epochs

# 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()
print(x_train.shape)

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)

x_train = x_train.astype('float32')
x_test = x_test.astype('float32')
x_train /= 255
x_test /= 255

#Take 5000 out of 60000 subset of training data to improve speed
x_train =x_train[0:5000,:,:,:]
y_train=y_train[0:5000]
print(y_train[0:10])


#Take 1000 out of 10000 subset of test data to improve speed
x_test =x_test[0:1000,:,:,:]
y_test=y_test[0:1000]
print(y_test[0:10])

print('x_train shape:', x_train.shape)
print(x_train.shape[0], 'train samples')
print(x_test.shape[0], 'test samples')
```

```python
# convert class label vectors to binary class matrices
y_train = keras.utils.to_categorical(y_train, num_classes)
y_test = keras.utils.to_categorical(y_test, num_classes)
print(y_train[0])



##############################################################################################################
#CREATE MODEL 1 - FULLY CONNECTED FEEDFORWARD NETWORK
#Loss function: categorical crossentropy
#Optimizer: SGD

x_train_flat = x_train.reshape((x_train.shape[0], -1))
x_test_flat = x_test.reshape((x_test.shape[0], -1))

model = Sequential()
model.add(Dense(32, input_shape=(x_train_flat.shape[-1],), activation='relu'))
model.add(Dense(64, activation='relu'))
model.add(Dense(32, activation='relu'))
model.add(Dense(num_classes, activation='softmax'))

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

#TRAIN AND EVALUATE MODEL 1
model.fit(x_train_flat, y_train,
          batch_size=batch_size,
          epochs=epochs,
          verbose=1,
          validation_data=(x_test_flat, y_test))
score = model.evaluate(x_test_flat, y_test, verbose=0)
print('Test loss:', score[0])
print('Test accuracy:', score[1])



##############################################################################################################
#CREATE MODEL 2 - CONVOLUTIONAL NETWORK
#Loss function: categorical crossentropy
#Optimizer: SGD

model = Sequential()
model.add(Conv2D(32, kernel_size=(3, 3),
                 activation='relu',
                 input_shape=input_shape))
model.add(Conv2D(64, (3, 3), activation='relu'))
model.add(MaxPooling2D(pool_size=(2,2)))
#model.add(Dropout(0.25)) #To prevent overfitting
model.add(Flatten()) #To input to fully connected layer
model.add(Dense(32, activation='relu')) #Fully connected layer
#model.add(Dropout(0.5)) #To avoid overfitting
```

```
#model.add(Dropout(0.5)) #To avoid overfitting
model.add(Dense(num_classes, activation='softmax')) #Output layer

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

#TRAIN AND EVALUATE MODEL 2
model.fit(x_train_flat, y_train,
          batch_size=batch_size,
          epochs=epochs,
          verbose=1,
          validation_data=(x_test_flat, y_test))
score = model.evaluate(x_test_flat, y_test, verbose=0)
print('Test loss:', score[0])
print('Test accuracy:', score[1])



######################################################################################################
#COMPARE OPTIMIZERS

#MODEL 2 - Fully connected feedforward neural network:
#Loss function: categorical crossentropy
#Optimizer: Adadelta

x_train_flat = x_train.reshape((x_train.shape[0], -1))
x_test_flat = x_test.reshape((x_test.shape[0], -1))

model = Sequential()
model.add(Dense(32, input_shape=(x_train_flat.shape[-1],), activation='relu'))
model.add(Dense(64, activation='relu'))
model.add(Dense(32, activation='relu'))
model.add(Dense(num_classes, activation='softmax'))

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

#Train and evaluate model
model.fit(x_train_flat, y_train,
          batch_size=batch_size,
          epochs=epochs,
          verbose=1,
          validation_data=(x_test_flat, y_test))
score = model.evaluate(x_test_flat, y_test, verbose=0)
print('Test loss:', score[0])
print('Test accuracy:', score[1])



#############################################################################
```

```
##########################################################################################
#MODEL 4 - Convolutional neural network:
#Loss function: categorical crossentropy
#Optimizer: Adadelta
model = Sequential()
model.add(Conv2D(32, kernel_size=(3, 3),
                 activation='relu',
                 input_shape=input_shape))
model.add(Conv2D(64, (3, 3), activation='relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))
#model.add(Dropout(0.25)) #To prevent overfitting
model.add(Flatten()) #To input to fully connected layer
model.add(Dense(32, activation='relu')) #Fully connected layer
#model.add(Dropout(0.5)) #To avoid overfitting
model.add(Dense(num_classes, activation='softmax')) #Output layer

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

#TRAIN AND EVALUATE MODEL 3
model.fit(x_train_flat, y_train,
          batch_size=batch_size,
          epochs=epochs,
          verbose=1,
          validation_data=(x_test_flat, y_test))
score = model.evaluate(x_test_flat, y_test, verbose=0)
print('Test loss:', score[0])
print('Test accuracy:', score[1])
```

MEG Analysis – Code

Study Group 6

```python
"""
MEG analysis - Classification algorithms

@author: Martin, Christoffer, Line and Simon (Study Group 6)
"""

# Import libraries
import numpy as np
import pandas as pd
from sklearn.naive_bayes import GaussianNB
from sklearn.model_selection import cross_val_score
from sklearn.svm import SVC
from sklearn.neighbors import KNeighborsClassifier
import matplotlib.pyplot as plt
from sklearn.pipeline import make_pipeline
from sklearn.preprocessing import StandardScaler
import statsmodels.stats.api as sms

# Import data and labels
labels = np.load('pos_neg_img_labels.npy')
data = np.load('pos_neg_img_trials.npy')

# Loop through each sample and perform cross-validation (linear SVM)
n_samples = data.shape[2] # Get number of samples
index=np.arange(0)
Classification  = pd.DataFrame(columns = ["Sample", "Score", "SD", "CI_low", "CI_high"], index = index) # Create empty dataframe

for sample_index in range(n_samples):
    temp = data[:, :, sample_index] # Use all datapoints for the given sample index
    clf = SVC(kernel="linear", C=1) # Linear suport vector classifier
    clf = make_pipeline(StandardScaler(copy=False), clf) #Standardize data
    cv_score = cross_val_score(clf, temp, labels, cv=10) #perform 10-fold cross-validation
    mean_cv = np.mean(cv_score) # Mean
    sd_cv = np.std(cv_score) # Standard Deviation
    CI_low, CI_high=sms.DescrStatsW(cv_score).tconfint_mean() # 95 % Confidence intervals
    Classification = Classification.append({"Sample": sample_index, "Score": mean_cv, "SD": sd_cv, "CI_low": CI_low, "CI_high": CI_high}, ignore_index = True) # Append to dataframe

# Create a preliminary plot
plt.plot(Classification["Sample"], Classification["Score"])
plt.show()

# Save results csv file
Classification.to_csv('linearSVM.csv')

# Loop through each sample and perform cross-validation (GaussianNB)
n_samples = data.shape[2]
index=np.arange(0)
Classification  = pd.DataFrame(columns = ["Sample", "Score", "SD", "CI_low", "CI_high"], index = index)

for sample_index in range(n_samples):
    temp = data[:, :, sample_index]
    clf = GaussianNB()
    clf = make_pipeline(StandardScaler(copy=False), clf)
    cv_score = cross_val_score(clf, temp, labels, cv=10)
    mean_cv = np.mean(cv_score)
```

```
    sd_cv = np.std(cv_score)
    CI_low, CI_high=sms.DescrStatsW(cv_score).tconfint_mean()
    Classification = Classification.append({"Sample": sample_index, "Score": mean_cv, "SD": sd_cv, "CI_low": CI_low, "CI_high": CI_high}, ignore_index = True)

# Make preliminary plot
plt.plot(Classification["Sample"], Classification["Score"])
plt.show()

# Save results csv file
Classification.to_csv('GaussianNB.csv')

# Loop through each sample and perform cross-validation (K-nearest neighbour )
n_samples = data.shape[2]
index=np.arange(0)
Classification  = pd.DataFrame(columns = ["Sample", "Score", "SD", "CI_low", "CI_high"], index = index)

for sample_index in range(n_samples):
    temp = data[:, :, sample_index]
    clf = KNeighborsClassifier(3)
    clf = make_pipeline(StandardScaler(copy=False), clf)
    cv_score = cross_val_score(clf, temp, labels, cv=10)
    mean_cv = np.mean(cv_score)
    sd_cv = np.std(cv_score)
    CI_low, CI_high=sms.DescrStatsW(cv_score).tconfint_mean()
    Classification = Classification.append({"Sample": sample_index, "Score": mean_cv, "SD": sd_cv, "CI_low": CI_low, "CI_high": CI_high}, ignore_index = True)

# Make preliminary plots
plt.plot(Classification["Sample"], Classification["Score"])
plt.show()

# Save results as csv file
Classification.to_csv('Knearest(3).csv')

"""
MEG analysis - Plots

@author: Martin, Christoffer, Line and Simon (Study Group 6) (Adapted script from Lau)
"""

# Import relevant libraries
from os.path import join ## connects strings into filepaths
import mne
import matplotlib.pyplot as plt # for plotting control
import numpy as np

# Set data path
data_path = 'C:\\Users\\Simon\\Documents\\MEG_numpy\\ME'

# Choose what subjects to analyse
subjects = [
#                         'Group1',
#                         'Group2', # noisy: change rejection parameters
#                         'Group3',
#                         'Group4',
#                         'Group5',
                         'Group6',
#                         'Group7'
```

```
                      ]

# Specify filename
filename = 'Session_70_Hz-ave.fif'

# Close all exisiting figures
plt.close('all')

# Loop over choosen subjects
for subject in subjects:
    # Read in the evokeds
    evokeds = mne.read_evokeds(join(data_path, subject, filename))
    for evoked in evokeds: # loop through evokeds
        # Pick the two stimulus classes ("img_pos" and "img_neg")
        if evoked.comment == 'img_pos' or evoked.comment == 'img_neg':
            # Create butterfly plots
            evoked.plot(window_title=subject + '_' + evoked.comment)
            evoked.plot(spatial_colors=True, gfp=True, picks='meg')
            # Create topomaps
            evoked.plot_topomap(times='peaks', ch_type='mag', time_unit='s')


# Plot Global Field Power
conditions = ["img_pos", "img_neg"] # Specify conditon
evoked_dict = dict() # Create empty dictionary
for condition in conditions: # Loop through conditions
    evoked_dict[condition.replace(" ", "/")] = mne.read_evokeds(
        join(data_path, subject, filename), baseline=(None, 0), proj=True, condition=condition) # Read in evokeds and append to dictionary
print(evoked_dict) # Print to check

colors = dict(img_pos="Crimson", img_neg="CornFlowerBlue") # Choose colours for each condition

# Create the plot based on the evoked dictionary
mne.viz.plot_compare_evokeds(evoked_dict, colors=colors, split_legend=True)
```

```
"""
MEG analysis - Classification performance plots

@author: Martin, Christoffer, Line and Simon (Study Group 6)
"""

#----- Libraries -----
library(ggplot2)

#----- Paths -----
#Set working directory
setwd("C:\\Users\\Martin\\Documents\\UNI\\7th semester\\Advanced Neuro\\exam\\meg")

#Paths to data
NB_path = "GaussianNB.csv"
SVC_path = "SVCC1.csv"
KN_path = "KNearest(3).csv"

#----- Load data -----
#Naive bayes classifier
NB_data = read.csv(NB_path) #Read file
NB_data$Sample = NB_data$Sample-500 #move stimulus presentation to 0
NB_data[NB_data$Score == max(NB_data$Score),] #Find sample with max accuracy

#Linear support vector machine
SVC_data = read.csv(SVC_path) #Read file
SVC_data$Sample = SVC_data$Sample-500 #move stimulus presentation to 0
SVC_data[SVC_data$Score == max(SVC_data$Score),] #Find sample with max accuracy

#Knearest neighbours, 3
#Linear support vector machine
KN_data = read.csv(KN_path) #Read file
KN_data$Sample = KN_data$Sample-500 #move stimulus presentation to 0
KN_data[KN_data$Score == max(KN_data$Score),] #Find sample with max accuracy


#----- Plot data -----
#Naive bayes
ggplot(NB_data, aes(x = Sample, y = Score))+
  scale_x_continuous(breaks=seq(-500,1000,100))+ #Fix x axis interval and increment
  ylim(0.25, 1)+ #Y-axis limits
  geom_ribbon(aes(ymin = CI_low, ymax = CI_high), fill = "grey", alpha = 0.5)+ #CI
  geom_vline(xintercept = 0, color = "red")+ #Vertical line at x = 0
  geom_line()+ #Line between scores
  geom_hline(yintercept = 0.5, linetype = "dashed", color = "black")+ #Dashed line at chance level
  labs(title ="Accuracy of Naive Bayes Classifier Over Time", #Labels
       x = "Time after stimulus in ms",
       y = "Accuracy")+
  theme_minimal() #Theme

#Linear support vector machine
ggplot(SVC_data, aes(x = Sample, y = Score))+
  scale_x_continuous(breaks=seq(-500,1000,100))+ #Fix x axis interval and increment
  ylim(0.25, 1)+ #Y-axis limits
  geom_ribbon(aes(ymin = CI_low, ymax = CI_high), fill = "grey", alpha = 0.5)+ #CI
  geom_vline(xintercept = 0, color = "red")+ #Vertical line at x = 0
  geom_line()+ #Line between scores
  geom_hline(yintercept = 0.5, linetype = "dashed", color = "black")+ #Dashed line at chance level
  labs(title ="Accuracy of Linear Support Vector Machine Over Time", #Labels
```

```
      x = "Time after stimulus in ms",
      y = "Accuracy")+
  theme_minimal() #Theme

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

fMRI Analysis – Code

Study Group 6

```
"""
fMRI analysis - Classification algorithms

@author: Martin, Christoffer, Line and Simon (Study Group 6)
"""

%matplotlib inline

#Adding data paths

#The fMRI data
fmri_filename='/Users/christoffer/Documents/CogSci/MA/Neuro/fMRI/subject064_results/beta4D.nii'

#Normalized Structural file
anat_filename='/Users/christoffer/Documents/CogSci/MA/Neuro/spm12/canonical/single_subj_T1.nii'

#A whole brain mask
mask_wb_filename='/Users/christoffer/Documents/CogSci/MA/Neuro/fMRI/subject064_results/mask.nii'

#The classification labels
class_filename='/Users/christoffer/Documents/CogSci/MA/Neuro/fMRI/scripts/class_labels_faces.txt'




# Making training and test split

import pandas as pd
import numpy as np
#from nilearn import datasets
from nilearn.image import new_img_like, load_img, index_img, clean_img
from sklearn.model_selection import train_test_split

# Reshaping data------------------------------
from nilearn.image import index_img, concat_imgs

#load fMRI data
fmri_img = load_img(fmri_filename)
#Print dimensions of data to get overview
print(fmri_img.shape)


#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)

#load csv-file with class labels
conditions = pd.read_csv(class_filename, sep=",")
conditions = conditions['labels']
#Print dimensions of data
print(conditions.shape)

#Make an index for spliting fMRI data with same size as class labels
idx=np.arange(conditions.shape[0])
# create training and testing vars on the basis of class labels
```

```
#the function train_test_split outputs 4 values --> we need to make sure the names of the outputs are correct
idx1,idx2, conditions1,  conditions2 = train_test_split(idx,conditions, test_size=0.2)
print(idx1, idx2)

# Reshaping data------------------------------
from nilearn.image import index_img
fmri_img1 = index_img(fmri_img, idx1)
fmri_img2 = index_img(fmri_img, idx2)
#Check data sizes
print(fmri_img1.shape)
print(fmri_img2.shape)



# SEARCHLIGHT
import pandas as pd
import numpy as np
from nilearn.image import new_img_like, load_img
from nilearn.plotting import plot_stat_map, plot_img, show
from nilearn import decoding
from nilearn.decoding import SearchLight
from sklearn import naive_bayes, model_selection #import GaussianNB
from sklearn.naive_bayes import GaussianNB

########################################################################

#Load the whole brain mask
mask_img = load_img(mask_wb_filename)
process_mask = mask_img.get_data().astype(np.int)
process_mask_img = new_img_like(mask_img, process_mask)

#Plot the searchlight scores on an anatomical background
plot_img(process_mask_img, bg_img=anat_filename,#bg_img=mean_fmri,
         title="Mask", display_mode="x",cut_coords=[28,32,36,40,44,48,52,56,60,64,68,72],
         vmin=.40, cmap='jet', threshold=0.9, black_bg=True)
########################################################################

# USING NB
# The radius is the one of the Searchlight sphere that will scan the volume
searchlight = SearchLight(
    mask_img,
    estimator=GaussianNB(),
    process_mask_img=process_mask_img,
    radius=5, n_jobs=1,
    verbose=1, cv=8)
searchlight.fit(fmri_img1, conditions1) #Run function on training data (fmri_img1) for condition 1.

#USING SVM
# The radius is the one of the Searchlight sphere that will scan the volume
searchlight = SearchLight(
    mask_img,
    process_mask_img=process_mask_img,
    radius=5, n_jobs=1,
    verbose=1, cv=8)
searchlight.fit(fmri_img1, conditions1) #Run function on training data (fmri_img1) for condition 1.
```

```
#USING k-NN
from sklearn import neighbors, datasets
searchlight = SearchLight(
    mask_img,
    estimator=neighbors.KNeighborsClassifier(3),
    process_mask_img=process_mask_img,
    radius=5, n_jobs=1,
    verbose=1, cv=8)
searchlight.fit(fmri_img1, conditions1) #Run function on training data (fmri_img1) for condition 1.


# 500 BEST VOXELS
print(searchlight.scores_.size)
#Find the percentile that makes the cutoff for the 500 best voxels
perc=100*(1-500.0/searchlight.scores_.size)
#Print percentile
print(perc)
#Find the cutoff
cut=np.percentile(searchlight.scores_,perc)
#Print cutoff
print(cut)

#Make a mask using cutoff

#Load the whole brain mask
mask_img2 = load_img(mask_wb_filename)


# MAKE GLASS BRAIN PLOT
# .astype() makes a copy.
process_mask2 = mask_img2.get_data().astype(np.int)
process_mask2[searchlight.scores_<=cut] = 0
process_mask2_img = new_img_like(mask_img2, process_mask2)

rom nilearn import image
from nilearn.plotting import plot_stat_map, plot_img, show
from nilearn import plotting
%matplotlib inline
#Create an image of the searchlight scores
searchlight_img = new_img_like(anat_filename, searchlight.scores_)
print(searchlight_img.shape)

#Ploting all voxels
plotting.plot_glass_brain(searchlight_img)

#Plotting 500 best voxels
plotting.plot_glass_brain(searchlight_img, threshold=cut)




#CLASSIFICATION
#using NB
from nilearn.input_data import NiftiMasker
masker = NiftiMasker(mask_img=process_mask2_img, standardize=False)

# We use masker to retrieve a 2D array ready
# for machine learning with scikit-learn
```

```
fmri_masked = masker.fit_transform(fmri_img2)
#Print size of matrix (images x voxels)
print(fmri_masked.shape)

from sklearn.naive_bayes import GaussianNB
from sklearn.model_selection import cross_val_score
cv_score = cross_val_score(GaussianNB(), fmri_masked, conditions2, cv=4)
print(cv_score)
print('Mean prediction score:')
print(np.mean(cv_score))

#Using SVM
from nilearn.input_data import NiftiMasker
masker = NiftiMasker(mask_img=process_mask2_img, standardize=False)

# We use masker to retrieve a 2D array ready
# for machine learning with scikit-learn
fmri_masked = masker.fit_transform(fmri_img2)
#Print size of matrix (images x voxels)
print(fmri_masked.shape)

from sklearn.naive_bayes import GaussianNB
from sklearn.model_selection import cross_val_score
from sklearn import svm
cv_score = cross_val_score(svm.SVC(kernel='linear', C=1), fmri_masked,
conditions2, cv=4)
print(cv_score)
print('Mean prediction score:')
print(np.mean(cv_score))

#Using k-NN
from nilearn.input_data import NiftiMasker
masker = NiftiMasker(mask_img=process_mask2_img, standardize=False)

# We use masker to retrieve a 2D array ready
# for machine learning with scikit-learn
fmri_masked = masker.fit_transform(fmri_img2)
#Print size of matrix (images x voxels)
print(fmri_masked.shape)

from sklearn.naive_bayes import GaussianNB
from sklearn.model_selection import cross_val_score
cv_score = cross_val_score(neighbors.KNeighborsClassifier(3), fmri_masked,
conditions2, cv=4)
print(cv_score)
print('Mean prediction score:')
print(np.mean(cv_score))

#PERMUTATION TESTS
#using NB
from sklearn.model_selection import permutation_test_score
score, permutation_scores, pvalue= permutation_test_score(
    GaussianNB(), fmri_masked, conditions2, cv=4, n_permutations=100,
    n_jobs=1, random_state=0, verbose=0, scoring=None)
print("Classification Accuracy: %s (pvalue : %s)" % (score, pvalue))

#using SVM
from sklearn.model_selection import permutation_test_score
score, permutation_scores, pvalue= permutation_test_score(
```

```python
    svm.SVC(kernel='linear', C=1), fmri_masked, conditions2, cv=4, n_permutations=100,
    n_jobs=1, random_state=0, verbose=0, scoring=None)
print("Classification Accuracy: %s (pvalue : %s)" % (score, pvalue))

#using k-NN
from sklearn.model_selection import permutation_test_score
score, permutation_scores, pvalue= permutation_test_score(
    neighbors.KNeighborsClassifier(3), fmri_masked, conditions2, cv=4, n_permutations=100,
    n_jobs=1, random_state=0, verbose=0, scoring=None)
print("Classification Accuracy: %s (pvalue : %s)" % (score, pvalue))



#Plot permutation histogram
import numpy as np
import matplotlib.pyplot as plt
#How many classes
n_classes = np.unique(conditions2).size

plt.hist(permutation_scores, 20, label='Permutation scores',
         edgecolor='black')
ylim = plt.ylim()
plt.plot(2 * [score], ylim, '--g', linewidth=3,
         label='Classification Score'
         ' (pvalue %s)' % pvalue)
plt.plot(2 * [1. / n_classes], ylim, '--k', linewidth=3, label='Chance level')

plt.ylim(ylim)
plt.legend()
plt.xlabel('Score')
plt.show()
```