functions.py

December 17, 2018

```
In [ ]: # Functions for project

        import numpy as np
        import pandas as pd
        import re
        import tensorflow as tf
        from sklearn.model_selection import train_test_split
        import matplotlib
        import matplotlib.pyplot as plt
        from mpl_toolkits.axes_grid1 import make_axes_locatable
        import random
        from sklearn.metrics import r2_score, mean_squared_error, accuracy_score, log_loss
        from sklearn import svm #support vector machines
        from sklearn.feature_extraction.text import CountVectorizer

        from sklearn.model_selection import cross_val_score
        from sklearn.ensemble import RandomForestClassifier
        from sklearn.linear_model import LogisticRegression
        from sklearn.neural_network import MLPClassifier

        from sklearn.metrics import confusion_matrix


        #######################
        #######################
        # Heatmap plotting functions
        #######################
        #######################


        ############## This is code to do neat plotting, taken from:
        # https://matplotlib.org/gallery/images_contours_and_fields/image_annotated_heatmap.htm

        def heatmap(data, row_labels, col_labels, ax=None,
                    cbar_kw={}, cbarlabel="", **kwargs):
            """
            Create a heatmap from a numpy array and two lists of labels.
```

```
Arguments:
    data       : A 2D numpy array of shape (N,M)
    row_labels : A list or array of length N with the labels
                 for the rows
    col_labels : A list or array of length M with the labels
                 for the columns
Optional arguments:
    ax         : A matplotlib.axes.Axes instance to which the heatmap
                 is plotted. If not provided, use current axes or
                 create a new one.
    cbar_kw    : A dictionary with arguments to
                 :meth:`matplotlib.Figure.colorbar`.
    cbarlabel  : The label for the colorbar
All other arguments are directly passed on to the imshow call.
"""

if not ax:
    ax = plt.gca()

# Plot the heatmap
im = ax.imshow(data, **kwargs)

# Create colorbar
cbar = ax.figure.colorbar(im, ax=ax, **cbar_kw)
cbar.ax.set_ylabel(cbarlabel, rotation=-90, va="bottom")

# We want to show all ticks...
ax.set_xticks(np.arange(data.shape[1]))
ax.set_yticks(np.arange(data.shape[0]))
# ... and label them with the respective list entries.
ax.set_xticklabels(col_labels)
ax.set_yticklabels(row_labels)

# Let the horizontal axes labeling appear on top.
ax.tick_params(top=True, bottom=False,
               labeltop=True, labelbottom=False)

# Rotate the tick labels and set their alignment.
plt.setp(ax.get_xticklabels(), rotation=-30, ha="right",
         rotation_mode="anchor")

# Turn spines off and create white grid.
for edge, spine in ax.spines.items():
    spine.set_visible(False)

ax.set_xticks(np.arange(data.shape[1]+1)-.5, minor=True)
ax.set_yticks(np.arange(data.shape[0]+1)-.5, minor=True)
```

```
    ax.grid(which="minor", color="w", linestyle='-', linewidth=3)
    ax.tick_params(which="minor", bottom=False, left=False)

    return im, cbar


def annotate_heatmap(im, data=None, valfmt="{x:.2f}",
                     textcolors=["black", "white"],
                     threshold=None, **textkw):
    """
    A function to annotate a heatmap.

    Arguments:
        im         : The AxesImage to be labeled.
    Optional arguments:
        data       : Data used to annotate. If None, the image's data is used.
        valfmt     : The format of the annotations inside the heatmap.
                     This should either use the string format method, e.g.
                     "$ {x:.2f}", or be a :class:`matplotlib.ticker.Formatter`.
        textcolors : A list or array of two color specifications. The first is
                     used for values below a threshold, the second for those
                     above.
        threshold  : Value in data units according to which the colors from
                     textcolors are applied. If None (the default) uses the
                     middle of the colormap as separation.

    Further arguments are passed on to the created text labels.
    """

    if not isinstance(data, (list, np.ndarray)):
        data = im.get_array()

    # Normalize the threshold to the images color range.
    if threshold is not None:
        threshold = im.norm(threshold)
    else:
        threshold = im.norm(data.max())/2.

    # Set default alignment to center, but allow it to be
    # overwritten by textkw.
    kw = dict(horizontalalignment="center",
              verticalalignment="center")
    kw.update(textkw)

    # Get the formatter in case a string is supplied
    if isinstance(valfmt, str):
        valfmt = matplotlib.ticker.StrMethodFormatter(valfmt)
    # Loop over the data and create a `Text` for each "pixel".
    # Change the text's color depending on the data.
    texts = []
    for i in range(data.shape[0]):
        for j in range(data.shape[1]):
            kw.update(color=textcolors[im.norm(data[i, j]) > threshold])
            text = im.axes.text(j, i, valfmt(data[i, j], None), **kw)
            texts.append(text)

    return texts

#######################
#######################
# plotting code over
#######################
#######################


#######################
#######################
# plotting code for confusion matrices
#######################
#######################


def plot_confusion_matrix(prediction, true_vals, labels, size = (12,12), normalize = '
    confusion = confusion_matrix(prediction, true_vals, labels = labels)
    fig, ax = plt.subplots(figsize=size)

    if normalize == 'rows' :
        im, cbar = heatmap((confusion.T/confusion.sum(axis=1)).T, labels, labels, ax =
    elif normalize == 'columns' :
        im, cbar = heatmap(confusion/confusion.sum(axis=1), labels, labels, ax = ax, c
    else :
        im, cbar = heatmap(confusion, labels, labels, ax = ax, cmap = "YlGn", cbarlabe

    texts = annotate_heatmap(im, valfmt="{x:.2f}")
    fig.tight_layout()
    plt.show()

def clf_confusion(clf, x_train, y_train, x_test, y_test, lables, size = (12,12), norma
    clf.fit(x_train, y_train)
    predictions = clf.predict(x_test)
    plot_confusion_matrix(predictions, y_test, lables, size = size, normalize = normal


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

```
# plotting code for confusion matrices over
#######################
#######################


#######################
#######################
# Functions for reading in the data
#######################
#######################


def get_design_matrix(cleaning_function = lambda x : x, min_df = 0.0, max_df = 1.0) :
    """
    Take a data frame data, and convert to a matrix.
    Use cleaning_function to clear up data.
    """
    data = pd.read_json('train.json')
    recipie_list_list = data.ingredients.values.tolist()
    recipie_string_list = [cleaning_function(" ".join(ing)) for ing in recipie_list_lis
    vectorizer = CountVectorizer(min_df = min_df, max_df = max_df)
    X = vectorizer.fit_transform(recipie_string_list)
    y = data.cuisine.values
    return X, y, vectorizer.get_feature_names()


def clean(s) :
    clean_s = s.replace('-',' ') # treat low-fat and low fat as the same thing
    clean_s = ''.join([c for c in clean_s if (c.isalpha() or c ==' ')]) # drop numbers


    return clean_s


#######################
#######################
# Method validation functions
#######################
#######################


def clf_cross_validator(X_train, y_train, clf_constructor, p_list, q_list = [], folds =
    """
    A general method to preform cross validation of sci_kit learn method.
    Takes:
    Training data (X_train, y_train)
    A function clf_constructor which builds a sci_kit learn classifier
    p_list a list of parameters to varry
    q_list a possible second list to varry,
    folds the number of folds to use for cross validation
    plot determines if a heatmap of the results should be printed
    label is the label of the plotting
    """

    scores = []
    # we rename the constructor,
    # only playes a role if q_list is empty
    constructor = clf_constructor

    ####
    # If we are only passed one list,
    # modify the constructor to take a second dummy argument
    # We make q_list be a singleton list,
    # and flip the role of p_list and q_list, the latter being only for printing purpo
    if not q_list : # true if q_list is empty
        q_list = p_list
        p_list = ['']
        constructor = (lambda p,q : clf_constructor(q))

    ###
    # We loop over the parameters
    # and record the avarage of each folds score
    for p in p_list :
        print("   p=%s" % str(p))
        for q in q_list :
            print("      q=%s" % str(q))
            clf = constructor(p,q)
            score = cross_val_score(clf, X_train, y_train, cv=folds)
            scores.append(np.mean(score))

    # transform the scores to a len(p_list) x len(q_list) shape array
    scores_array = np.array(scores).reshape(len(p_list), len(q_list))

    ####
    # make a heat map of the scores
    if plot :
        fig, ax = plt.subplots()
        im, cbar = heatmap(scores_array, np.array(p_list), np.array(q_list) , ax = ax,
        texts = annotate_heatmap(im, valfmt="{x:.4f}")
        fig.tight_layout()
        plt.show()

    return scores_array


def svm_tester(X_train, y_train, C_list = [0.1], folds = 10, plot = False) :
    """
    Test the svm parameter C using cross validation
    For each C in C_list do folds fold cross validation,
    returns an array, of the same size as C_list,
    of the avarages of the accuracies for each fold.

    If plot is set to true, show a heatmap of the results
    """

    svm_constructor = (lambda p : svm.LinearSVC(C = p))
    scores = clf_cross_validator(X_train, y_train, svm_constructor, C_list, folds = fol
    return scores

def forrest_tester(X_train, y_train, trees_list = [1], depth_list = [1], folds = 10, pl
    """
    Test the random forrest for the parameters of number of trees and max depth using 
    For each pair of parameters do folds fold cross validation,
    returns an array, of shape len(trees_list) x len(depth_list)
    of the avarages of the accuracies for each fold.

    If plot is set to true, show a heatmap of the results
    """

    forrest_constructor = (lambda p,q : RandomForestClassifier(n_estimators = p, max_de
    scores = clf_cross_validator(X_train, y_train, forrest_constructor, trees_list, dep
    return scores

def logistic_tester(X_train, y_train, C_list = [0.1], folds = 10, plot = False) :
    """
    Test the logistic regression parameter C using cross validation
    For each C in C_list do folds fold cross validation,
    returns an array, of the same size as C_list,
    of the avarages of the accuracies for each fold.

    If plot is set to true, show a heatmap of the results
    """

    # Not really sure about the solver
    logistic_constructor = (lambda p : LogisticRegression(solver='lbfgs', multi_class=
    scores = clf_cross_validator(X_train, y_train, logistic_constructor, C_list, folds 
    return scores

def mlp_tester(X_train, y_train, nodes = [1], alpha_list = [0.001], folds = 10, plot =
    """
    Test the mlp classifier (neural net) for the parameters of
    number of nodes in a single layer and regularization constant using cross validati
    For each pair of parameters do folds fold cross validation,
    returns an array, of shape len(nodes) x len(alpha_list)
    of the avarages of the accuracies for each fold.
    If plot is set to true, show a heatmap of the results
    """

    mlp_constructor = (lambda p,q : MLPClassifier(hidden_layer_sizes = p, alpha = q, ma
    scores = clf_cross_validator(X_train, y_train, mlp_constructor, nodes, alpha_list,
    return scores


def accuracy_with_min_df(min_df_list = [0], svm_parms = [0.1], log_parms = [1], forrest

    svm_accuracies = []
    log_accuracies = []
    forrest_accuracies = []

    # rather than calling get_design_matrix every time,
    # which loads the file train.json every time,
    # we just read it in once.
    data = pd.read_json('train.json')
    recipie_list_list = data.ingredients.values.tolist()
    recipie_string_list = [" ".join(ing) for ing in recipie_list_list]
    y = data.cuisine.values
    del data, recipie_list_list

    for df in min_df_list :
        print("Testing min_df = %f" % df)
        vectorizer = CountVectorizer(min_df = df)
        X = vectorizer.fit_transform(recipie_string_list)
        # Pick the highst cv scores (avarage of fold accuracies)
        print("Now doing svm cross validation")
        svm_accuracies.append(np.amax(svm_tester(X, y, C_list = svm_parms, folds = fol
        print("Now doing logistic cross validation")
        log_accuracies.append(np.amax(logistic_tester(X, y, C_list = log_parms, folds =
        print("Now doing forrest cross validation")
        forrest_accuracies.append(np.amax(forrest_tester(X, y, trees_list = forrest_tre
        print("")

    return svm_accuracies, log_accuracies, forrest_accuracies
```