Descriptive_analysis

December 17, 2018

1 Descriptive analysis

Here is a little insight in to how the dataset looks

Most of this code is taken from:

https://www.kaggle.com/gloriahristova/a-walkthrough-eda-vizualizations-unigram-model/notebook

1.0.1 Imports

```
In [1]: # Data processing
        import pandas as pd
        import json
        from collections import Counter
        from itertools import chain
        from sklearn.feature_extraction.text import TfidfVectorizer
        import numpy as np
        import re

        # Data vizualizations
        import random
        import plotly
        from plotly import tools
        from plotly.offline import download_plotlyjs, init_notebook_mode, plot, iplot
        init_notebook_mode(connected=True)
        import plotly.offline as offline
        import plotly.graph_objs as go
```

1.0.2 Reading in the data file

```
In [2]: train_data = pd.read_json('train.json')
        print(train_data.head())
```

```
       cuisine     id                                        ingredients
0        greek  10259  [romaine lettuce, black olives, grape tomatoes...
1  southern_us  25693  [plain flour, ground pepper, salt, tomatoes, g...
2     filipino  20130  [eggs, pepper, salt, mayonaise, cooking oil, g...
3       indian  22213                [water, vegetable oil, wheat, salt]
```

```
4       indian  13162  [black pepper, shallots, cornflour, cayenne pe...
```

```
In [3]: print("The training data consists of {} recipes".format(len(train_data)))
```

```
The training data consists of 39774 recipes
```

A function for producing random colors for plots:

```
In [4]: def random_colours(number_of_colors):
            '''
            Simple function for random colours generation.
            Input:
                number_of_colors - integer value indicating the number of colours which are go
            Output:
                Color in the following format: ['#E86DA4'] .
            '''
            colors = []
            for i in range(number_of_colors):
                colors.append("#"+''.join([random.choice('0123456789ABCDEF') for j in range(6)]
            return colors
```

1.0.3 Number of recipes in each cuisine

```
In [5]: trace = go.Table(
                        header=dict(values=['Cuisine','Number of recipes'],
                        fill = dict(color=['#EABEB0']),
                        align = ['left'] * 5),
                        cells=dict(values=[train_data.cuisine.value_counts().index,train_data.c
                       align = ['left'] * 5))

        layout = go.Layout(title='Number of recipes in each cuisine category',
                           titlefont = dict(size = 20),
                           width=500, height=650,
                           paper_bgcolor =  'rgba(0,0,0,0)',
                           plot_bgcolor = 'rgba(0,0,0,0)',
                           autosize = False,
                           margin=dict(l=30,r=30,b=1,t=50,pad=1),
                           )
        data = [trace]
        fig = dict(data=data, layout=layout)
        iplot(fig)
```

1.1 Percentage of each cuisine

```
In [6]: #  Label distribution in percents
        labelpercents = []
        for i in train_data.cuisine.value_counts():
```

```
        percent = (i/sum(train_data.cuisine.value_counts()))*100
        percent = "%.2f" % percent
        percent = str(percent + '%')
        labelpercents.append(percent)

    trace = go.Bar(
                x=train_data.cuisine.value_counts().values[::-1],
                y= [i for i in train_data.cuisine.value_counts().index][::-1],
                text =labelpercents[::-1],  textposition = 'outside',
                orientation = 'h',marker = dict(color = random_colours(20)))
    layout = go.Layout(title='Number of recipes in each cuisine category',
                       titlefont = dict(size = 25),
                       width=1000, height=450,
                       plot_bgcolor = 'rgba(0,0,0,0)',
                       paper_bgcolor = 'rgba(255, 255, 255, 0.88)',
                       margin=dict(l=75,r=110,b=50,t=60),
                       )
    data = [trace]
    fig = dict(data=data, layout=layout)
    iplot(fig, filename='horizontal-bar')
```

So the italian and the mexican cuisine represents around 36% of the data and the bottom 10 cuisines represents around 18 % of the training data. We can expect from our predictors that when it is uncertain it might predict italian or mexican by "default", since it will be more right to guess there than other cuisines.

1.1.1 Distribution of the Recipe Length

```
In [7]: trace = go.Histogram(
            x= train_data['ingredients'].str.len(),
            xbins=dict(start=0,end=90,size=1),
           marker=dict(color='#7CFDF0'),
            opacity=0.75)
        data = [trace]
        layout = go.Layout(
            title='Distribution of Recipe Length',
            xaxis=dict(title='Number of ingredients'),
            yaxis=dict(title='Count of recipes'),
            bargap=0.1,
            bargroupgap=0.2)

        fig = go.Figure(data=data, layout=layout)
        iplot(fig)
```

So the recipes has a mean around 10 ingridients and there are very few recipes which contains ingridients more than 30 ingridients.

1.2 Abnormal recipes

Some recipes have very short recipes with only 1 or two ingridients.
There are 22 recipes with only one ingridient:

```
In [8]: print("Explore the ingredients in the shortest recipes in our training set:" + "\n")
        print(train_data.loc[train_data.ingredients.str.len() == 1])

        #print(list(train_data[train_data['ingredients'].str.len() == 1].ingredients.values))
        #print("And there corresponding labels" + "\n")
        #print(list(train_data[train_data['ingredients'].str.len() == 1].cuisine.values))
```

```
Explore the ingredients in the shortest recipes in our training set:

           cuisine     id                 ingredients
940       japanese   4734               [sushi rice]
2088    vietnamese   7833       [dried rice noodles]
6787        indian  36818     [plain low-fat yogurt]
7011        indian  19772          [unsalted butter]
8181      japanese  16116                    [udon]
8852          thai  29738              [sticky rice]
8990        indian  41124                   [butter]
10506      mexican  32631           [corn tortillas]
13178         thai  29570                 [grained]
17804  southern_us  29849     [lemonade concentrate]
18136         thai  39186             [jasmine rice]
18324       indian  14335          [unsalted butter]
21008      italian  39221          [cherry tomatoes]
22119       french  41135                   [butter]
22387       indian  36874              [cumin seed]
23512       french  35028           [haricots verts]
26887      mexican  18593            [vegetable oil]
29294      spanish   7460          [spanish chorizo]
30636      spanish  32772  [sweetened condensed milk]
32105     japanese  12805                    [water]
34531        greek  10816                   [phyllo]
37220       indian  27192          [unsalted butter]
```

1.3 Most common ingridients

Lets have a look at the most common ingridients.

```
In [9]: allingredients = [] # this list stores all the ingredients in all recipes (with duplic
        for item in train_data['ingredients']:
            for ingr in item:
                allingredients.append(ingr)

        # Count how many times each ingredient occurs
```

```
countingr = Counter()
for ingr in allingredients:
     countingr[ingr] += 1

# Extract the first 20 most common ingredients in order to vizualize them for better u
mostcommon = countingr.most_common(20)
mostcommoningr = [i[0] for i in mostcommon]
mostcommoningr_count = [i[1] for i in mostcommon]

trace = go.Bar(
            x=mostcommoningr_count[::-1],
            y= mostcommoningr[::-1],
            orientation = 'h',marker = dict(color = random_colours(20),
))
layout = go.Layout(
    xaxis = dict(title= 'Number of occurences in all recipes (training sample)', ),
    yaxis = dict(title='Ingredient',),
    title= '20 Most Common Ingredients', titlefont = dict(size = 20),
    margin=dict(l=150,r=10,b=60,t=60,pad=5),
    width=800, height=500,
)
data = [trace]
fig = go.Figure(data=data, layout=layout)
iplot(fig, filename='horizontal-bar')
```

We can tell that salt is the most common ingridient by far. We can also assume that salt isn't very specific for a certain cuisine, so it will probably not be a good predicator.

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.ht

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, cm
    else :
        im, cbar = heatmap(confusion, labels, labels, ax = ax, cmap = "YlGn", cbarlabel

    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), normal
    clf.fit(x_train, y_train)
    predictions = clf.predict(x_test)
    plot_confusion_matrix(predictions, y_test, lables, size = size, normalize = normali


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

```
# 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 = fo
    return scores

def forrest_tester(X_train, y_train, trees_list = [1], depth_list = [1], folds = 10, p
    """
    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_d
    scores = clf_cross_validator(X_train, y_train, forrest_constructor, trees_list, de
    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 = fold
        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
```

Testing all four methods over parameters

December 17, 2018

1 Searching over parametergrid with Log-reg, SVM's, forests and MLP's

In [1]:
```
import numpy as np
import pandas as pd
import re
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, VotingClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.neural_network import MLPClassifier

from sklearn.decomposition import PCA


from functions import *
```

```
C:\Users\Admin\Anaconda3\lib\site-packages\h5py\__init__.py:36: FutureWarning: Conversion of t
  from ._conv import register_converters as _register_converters
```

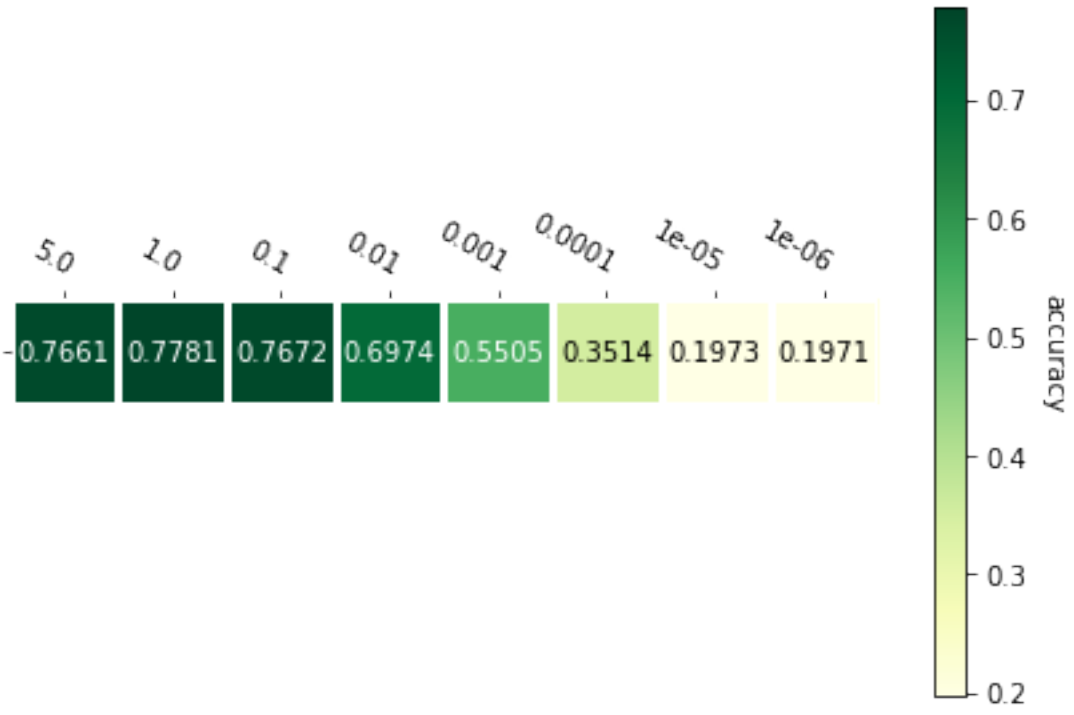
In [2]:
```
print("Importing design matrix ...")
X_train, y_train, features = get_design_matrix(cleaning_function = clean, min_df = 3)
print("Done.")
k = 3 # folds in the k-fold cross validation
plot = True # Set functions to plot heatmaps
```

```
Importing design matrix ...
Done.
```

2 Logistic regression

```
In [3]: # Setting up parameterlist
        log_p_list = [5, 1, .1, .01, .001, .0001, .00001, .000001]

        # k-fold cross validation over all parameteres and plotting
        print("Logistic regression: Cross validation over all parameters ...")
        log_scores = logistic_tester(X_train, y_train, C_list = log_p_list, folds = k, plot =
        print("Done.")
```

```
Logistic regression: Cross validation over all parameters ...
   p=
      q=5
      q=1
      q=0.1
      q=0.01
      q=0.001
      q=0.0001
      q=1e-05
      q=1e-06
```



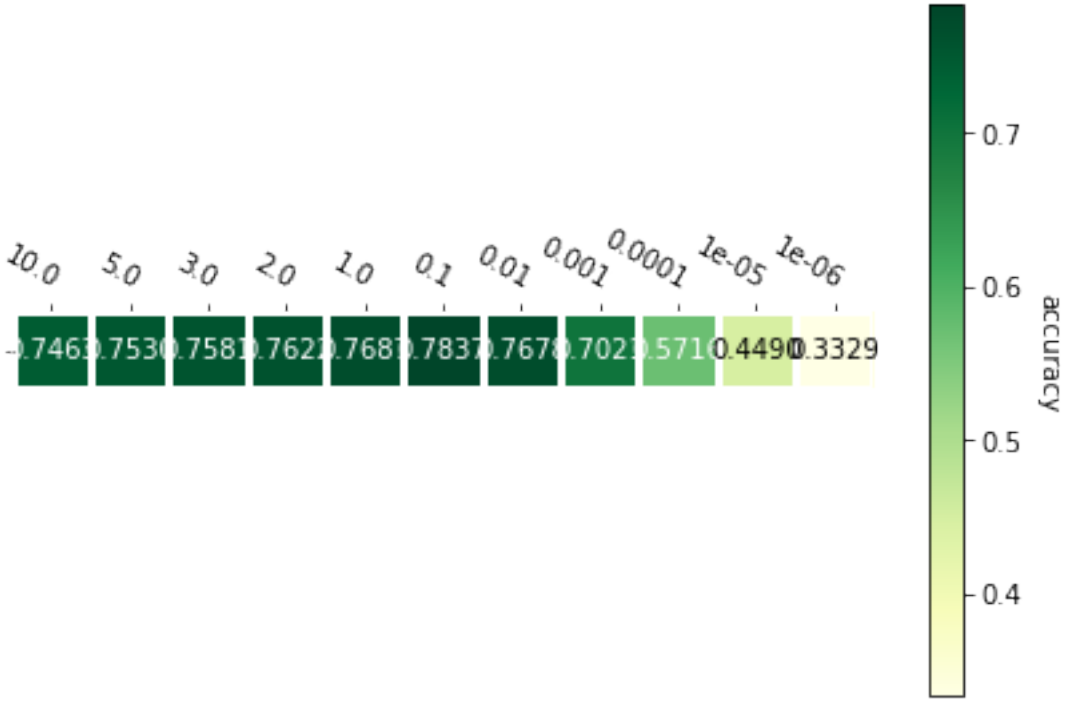

```
Done.
```

So we will continue with the regularization parameter set to 1.

3 Support vector machines

In [12]:
```
# Setting up parameterlist
svm_p_list = [10, 5, 3, 2, 1, .1, .01, .001, .0001, .00001, .000001]

# k-fold cross validation over all parameteres and plotting
print("Support vector machines: Cross validation over all parameters ...")
svm_scores = svm_tester(X_train, y_train, C_list = svm_p_list, folds = k, plot = plot)
print("Done.")
```

```
Support vector machines: Cross validation over all parameters ...
   p=
      q=10
      q=5
      q=3
      q=2
      q=1
      q=0.1
      q=0.01
      q=0.001
      q=0.0001
      q=1e-05
      q=1e-06
```



```
Done.
```

```
In [13]: print(svm_scores)
```

```
[[0.74629234 0.75295483 0.75813407 0.7621565  0.76869312 0.78365274
  0.76781258 0.70206656 0.57162994 0.44901306 0.33285615]]
```

So we will continue with margin parameter set to 0.1.

4 Random forests

```
In [6]: # Setting up parameterlist
        forest_trees_list = [10, 20, 30, 50, 100]
        forest_depth_list = [5, 10, 15, 30, 50, None]

        # k-fold cross validation over all parameteres and plotting
        print("Forest: Cross validation over all parameters ...")
        forest_scores = forrest_tester(
            X_train, y_train, trees_list = forest_trees_list, depth_list = forest_depth_list, r
        print("Done.")
```

```
Forest: Cross validation over all parameters ...
p=10
q=5
q=10
q=15
q=30
q=50
q=None
p=20
q=5
q=10
q=15
q=30
q=50
q=None
p=30
q=5
q=10
q=15
q=30
q=50
q=None
p=50
q=5
q=10
```

```
q=15
q=30
q=50
q=None
p=100
q=5
q=10
q=15
q=30
q=50
q=None
```


```
Done.
```

```
In [16]: # Setting up parameterlist
         forest_trees_list = [200, 300, 500, 700, 1000]
         forest_depth_list = [5, 10, 15, 30, 50, None]

         # k-fold cross validation over all parameteres and plotting
         print("Forest: Cross validation over all parameters ...")
         forest_scores = forrest_tester(
             X_train, y_train, trees_list = forest_trees_list, depth_list = forest_depth_list,
         print("Done.")
```

```
Forest: Cross validation over all parameters ...
   p=200
      q=5
      q=10
      q=15
      q=30
      q=50
      q=None
   p=300
      q=5
      q=10
      q=15
      q=30
      q=50
      q=None
   p=500
      q=5
      q=10
      q=15
      q=30
      q=50
      q=None
   p=700
      q=5
      q=10
      q=15
      q=30
      q=50
      q=None
   p=1000
      q=5
      q=10
      q=15
      q=30
      q=50
      q=None
```


```
Done.
```

So we will continue with unlimited tree depth and 1000 trees in the forest. (PS: Takes a long time to train)

5 Multilayered perceptrons

```
In [3]: # Setting up parameterlist
        net_nodes_list = [(10), (10,10), (20,20), (30,30), (40,40), (50,50)]
        net_alpha_list = [.00001, .0001, .001, .01, .1, 1]

        # k-fold cross validation over all parameteres and plotting
        print("Neural networks: Cross validation over all parameters ...")
        mlp_scores = mlp_tester(
            X_train, y_train, nodes = net_nodes_list, alpha_list = net_alpha_list, folds = k, p

        print("Done.")
```

```
Neural networks: Cross validation over all parameters ...
   p=10
      q=1e-05
```

```
/home/jeanpylon/anaconda3/lib/python3.6/site-packages/sklearn/neural_network/multilayer_percept
  % self.max_iter, ConvergenceWarning)
/home/jeanpylon/anaconda3/lib/python3.6/site-packages/sklearn/neural_network/multilayer_percept
  % self.max_iter, ConvergenceWarning)
/home/jeanpylon/anaconda3/lib/python3.6/site-packages/sklearn/neural_network/multilayer_percept
  % self.max_iter, ConvergenceWarning)
```

```
    q=0.0001
```

```
/home/jeanpylon/anaconda3/lib/python3.6/site-packages/sklearn/neural_network/multilayer_percept
  % self.max_iter, ConvergenceWarning)
/home/jeanpylon/anaconda3/lib/python3.6/site-packages/sklearn/neural_network/multilayer_percept
  % self.max_iter, ConvergenceWarning)
/home/jeanpylon/anaconda3/lib/python3.6/site-packages/sklearn/neural_network/multilayer_percept
  % self.max_iter, ConvergenceWarning)
```

```
    q=0.001
```

```
/home/jeanpylon/anaconda3/lib/python3.6/site-packages/sklearn/neural_network/multilayer_percept
  % self.max_iter, ConvergenceWarning)
/home/jeanpylon/anaconda3/lib/python3.6/site-packages/sklearn/neural_network/multilayer_percept
  % self.max_iter, ConvergenceWarning)
/home/jeanpylon/anaconda3/lib/python3.6/site-packages/sklearn/neural_network/multilayer_percept
  % self.max_iter, ConvergenceWarning)
```

```
    q=0.01
```

```
/home/jeanpylon/anaconda3/lib/python3.6/site-packages/sklearn/neural_network/multilayer_percept
  % self.max_iter, ConvergenceWarning)
/home/jeanpylon/anaconda3/lib/python3.6/site-packages/sklearn/neural_network/multilayer_percept
  % self.max_iter, ConvergenceWarning)
/home/jeanpylon/anaconda3/lib/python3.6/site-packages/sklearn/neural_network/multilayer_percept
  % self.max_iter, ConvergenceWarning)
```

```
    q=0.1
```

```
/home/jeanpylon/anaconda3/lib/python3.6/site-packages/sklearn/neural_network/multilayer_percept
  % self.max_iter, ConvergenceWarning)
/home/jeanpylon/anaconda3/lib/python3.6/site-packages/sklearn/neural_network/multilayer_percept
  % self.max_iter, ConvergenceWarning)
/home/jeanpylon/anaconda3/lib/python3.6/site-packages/sklearn/neural_network/multilayer_percept
  % self.max_iter, ConvergenceWarning)
```

```
      q=1

/home/jeanpylon/anaconda3/lib/python3.6/site-packages/sklearn/neural_network/multilayer_percept
  % self.max_iter, ConvergenceWarning)
/home/jeanpylon/anaconda3/lib/python3.6/site-packages/sklearn/neural_network/multilayer_percept
  % self.max_iter, ConvergenceWarning)
/home/jeanpylon/anaconda3/lib/python3.6/site-packages/sklearn/neural_network/multilayer_percept
  % self.max_iter, ConvergenceWarning)


   p=(10, 10)
      q=1e-05

/home/jeanpylon/anaconda3/lib/python3.6/site-packages/sklearn/neural_network/multilayer_percept
  % self.max_iter, ConvergenceWarning)
/home/jeanpylon/anaconda3/lib/python3.6/site-packages/sklearn/neural_network/multilayer_percept
  % self.max_iter, ConvergenceWarning)
/home/jeanpylon/anaconda3/lib/python3.6/site-packages/sklearn/neural_network/multilayer_percept
  % self.max_iter, ConvergenceWarning)


      q=0.0001

/home/jeanpylon/anaconda3/lib/python3.6/site-packages/sklearn/neural_network/multilayer_percept
  % self.max_iter, ConvergenceWarning)
/home/jeanpylon/anaconda3/lib/python3.6/site-packages/sklearn/neural_network/multilayer_percept
  % self.max_iter, ConvergenceWarning)
/home/jeanpylon/anaconda3/lib/python3.6/site-packages/sklearn/neural_network/multilayer_percept
  % self.max_iter, ConvergenceWarning)


      q=0.001

/home/jeanpylon/anaconda3/lib/python3.6/site-packages/sklearn/neural_network/multilayer_percept
  % self.max_iter, ConvergenceWarning)
/home/jeanpylon/anaconda3/lib/python3.6/site-packages/sklearn/neural_network/multilayer_percept
  % self.max_iter, ConvergenceWarning)
/home/jeanpylon/anaconda3/lib/python3.6/site-packages/sklearn/neural_network/multilayer_percept
  % self.max_iter, ConvergenceWarning)


      q=0.01

/home/jeanpylon/anaconda3/lib/python3.6/site-packages/sklearn/neural_network/multilayer_percept
  % self.max_iter, ConvergenceWarning)
```

```
/home/jeanpylon/anaconda3/lib/python3.6/site-packages/sklearn/neural_network/multilayer_percept
  % self.max_iter, ConvergenceWarning)
/home/jeanpylon/anaconda3/lib/python3.6/site-packages/sklearn/neural_network/multilayer_percept
  % self.max_iter, ConvergenceWarning)


      q=0.1


/home/jeanpylon/anaconda3/lib/python3.6/site-packages/sklearn/neural_network/multilayer_percept
  % self.max_iter, ConvergenceWarning)
/home/jeanpylon/anaconda3/lib/python3.6/site-packages/sklearn/neural_network/multilayer_percept
  % self.max_iter, ConvergenceWarning)
/home/jeanpylon/anaconda3/lib/python3.6/site-packages/sklearn/neural_network/multilayer_percept
  % self.max_iter, ConvergenceWarning)


      q=1


/home/jeanpylon/anaconda3/lib/python3.6/site-packages/sklearn/neural_network/multilayer_percept
  % self.max_iter, ConvergenceWarning)
/home/jeanpylon/anaconda3/lib/python3.6/site-packages/sklearn/neural_network/multilayer_percept
  % self.max_iter, ConvergenceWarning)
/home/jeanpylon/anaconda3/lib/python3.6/site-packages/sklearn/neural_network/multilayer_percept
  % self.max_iter, ConvergenceWarning)


   p=(20, 20)
      q=1e-05


/home/jeanpylon/anaconda3/lib/python3.6/site-packages/sklearn/neural_network/multilayer_percept
  % self.max_iter, ConvergenceWarning)
/home/jeanpylon/anaconda3/lib/python3.6/site-packages/sklearn/neural_network/multilayer_percept
  % self.max_iter, ConvergenceWarning)
/home/jeanpylon/anaconda3/lib/python3.6/site-packages/sklearn/neural_network/multilayer_percept
  % self.max_iter, ConvergenceWarning)


      q=0.0001


/home/jeanpylon/anaconda3/lib/python3.6/site-packages/sklearn/neural_network/multilayer_percept
  % self.max_iter, ConvergenceWarning)
/home/jeanpylon/anaconda3/lib/python3.6/site-packages/sklearn/neural_network/multilayer_percept
  % self.max_iter, ConvergenceWarning)
/home/jeanpylon/anaconda3/lib/python3.6/site-packages/sklearn/neural_network/multilayer_percept
  % self.max_iter, ConvergenceWarning)
```

```
      q=0.001


/home/jeanpylon/anaconda3/lib/python3.6/site-packages/sklearn/neural_network/multilayer_percept
  % self.max_iter, ConvergenceWarning)
/home/jeanpylon/anaconda3/lib/python3.6/site-packages/sklearn/neural_network/multilayer_percept
  % self.max_iter, ConvergenceWarning)
/home/jeanpylon/anaconda3/lib/python3.6/site-packages/sklearn/neural_network/multilayer_percept
  % self.max_iter, ConvergenceWarning)


      q=0.01


/home/jeanpylon/anaconda3/lib/python3.6/site-packages/sklearn/neural_network/multilayer_percept
  % self.max_iter, ConvergenceWarning)
/home/jeanpylon/anaconda3/lib/python3.6/site-packages/sklearn/neural_network/multilayer_percept
  % self.max_iter, ConvergenceWarning)
/home/jeanpylon/anaconda3/lib/python3.6/site-packages/sklearn/neural_network/multilayer_percept
  % self.max_iter, ConvergenceWarning)


      q=0.1


/home/jeanpylon/anaconda3/lib/python3.6/site-packages/sklearn/neural_network/multilayer_percept
  % self.max_iter, ConvergenceWarning)
/home/jeanpylon/anaconda3/lib/python3.6/site-packages/sklearn/neural_network/multilayer_percept
  % self.max_iter, ConvergenceWarning)
/home/jeanpylon/anaconda3/lib/python3.6/site-packages/sklearn/neural_network/multilayer_percept
  % self.max_iter, ConvergenceWarning)


      q=1


/home/jeanpylon/anaconda3/lib/python3.6/site-packages/sklearn/neural_network/multilayer_percept
  % self.max_iter, ConvergenceWarning)
/home/jeanpylon/anaconda3/lib/python3.6/site-packages/sklearn/neural_network/multilayer_percept
  % self.max_iter, ConvergenceWarning)
/home/jeanpylon/anaconda3/lib/python3.6/site-packages/sklearn/neural_network/multilayer_percept
  % self.max_iter, ConvergenceWarning)


   p=(30, 30)
      q=1e-05


/home/jeanpylon/anaconda3/lib/python3.6/site-packages/sklearn/neural_network/multilayer_percept
  % self.max_iter, ConvergenceWarning)
```

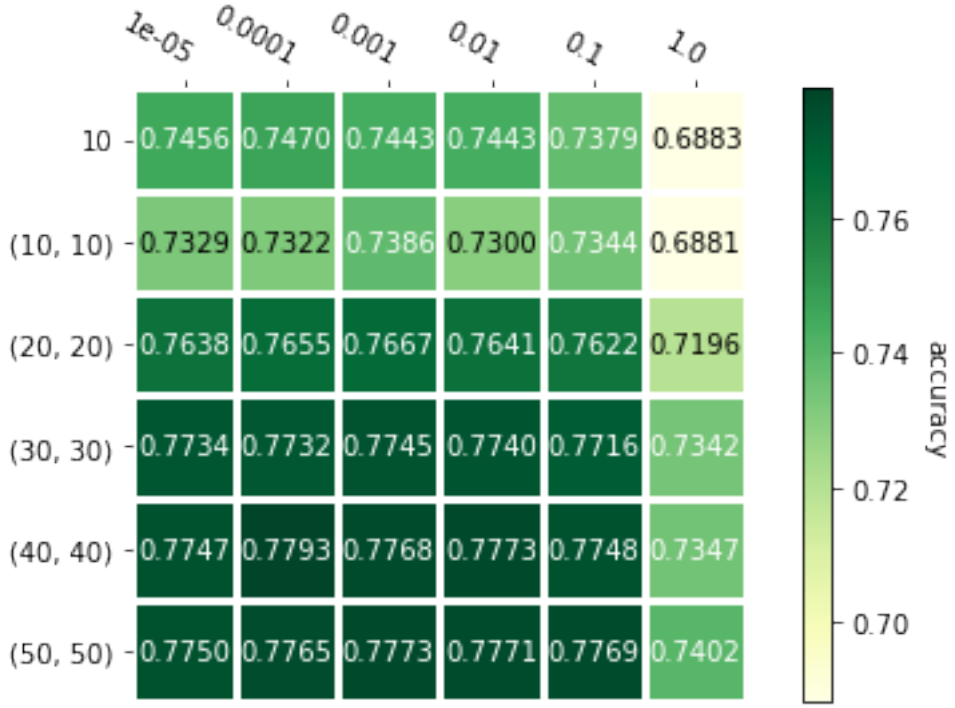
```
/home/jeanpylon/anaconda3/lib/python3.6/site-packages/sklearn/neural_network/multilayer_percept
  % self.max_iter, ConvergenceWarning)
/home/jeanpylon/anaconda3/lib/python3.6/site-packages/sklearn/neural_network/multilayer_percept
  % self.max_iter, ConvergenceWarning)


      q=0.0001


/home/jeanpylon/anaconda3/lib/python3.6/site-packages/sklearn/neural_network/multilayer_percept
  % self.max_iter, ConvergenceWarning)
/home/jeanpylon/anaconda3/lib/python3.6/site-packages/sklearn/neural_network/multilayer_percept
  % self.max_iter, ConvergenceWarning)
/home/jeanpylon/anaconda3/lib/python3.6/site-packages/sklearn/neural_network/multilayer_percept
  % self.max_iter, ConvergenceWarning)


      q=0.001


/home/jeanpylon/anaconda3/lib/python3.6/site-packages/sklearn/neural_network/multilayer_percept
  % self.max_iter, ConvergenceWarning)
/home/jeanpylon/anaconda3/lib/python3.6/site-packages/sklearn/neural_network/multilayer_percept
  % self.max_iter, ConvergenceWarning)
/home/jeanpylon/anaconda3/lib/python3.6/site-packages/sklearn/neural_network/multilayer_percept
  % self.max_iter, ConvergenceWarning)


      q=0.01


/home/jeanpylon/anaconda3/lib/python3.6/site-packages/sklearn/neural_network/multilayer_percept
  % self.max_iter, ConvergenceWarning)
/home/jeanpylon/anaconda3/lib/python3.6/site-packages/sklearn/neural_network/multilayer_percept
  % self.max_iter, ConvergenceWarning)
/home/jeanpylon/anaconda3/lib/python3.6/site-packages/sklearn/neural_network/multilayer_percept
  % self.max_iter, ConvergenceWarning)


      q=0.1


/home/jeanpylon/anaconda3/lib/python3.6/site-packages/sklearn/neural_network/multilayer_percept
  % self.max_iter, ConvergenceWarning)
/home/jeanpylon/anaconda3/lib/python3.6/site-packages/sklearn/neural_network/multilayer_percept
  % self.max_iter, ConvergenceWarning)
/home/jeanpylon/anaconda3/lib/python3.6/site-packages/sklearn/neural_network/multilayer_percept
  % self.max_iter, ConvergenceWarning)
```

```
      q=1


/home/jeanpylon/anaconda3/lib/python3.6/site-packages/sklearn/neural_network/multilayer_percept
  % self.max_iter, ConvergenceWarning)
/home/jeanpylon/anaconda3/lib/python3.6/site-packages/sklearn/neural_network/multilayer_percept
  % self.max_iter, ConvergenceWarning)
/home/jeanpylon/anaconda3/lib/python3.6/site-packages/sklearn/neural_network/multilayer_percept
  % self.max_iter, ConvergenceWarning)


   p=(40, 40)
      q=1e-05


/home/jeanpylon/anaconda3/lib/python3.6/site-packages/sklearn/neural_network/multilayer_percept
  % self.max_iter, ConvergenceWarning)
/home/jeanpylon/anaconda3/lib/python3.6/site-packages/sklearn/neural_network/multilayer_percept
  % self.max_iter, ConvergenceWarning)
/home/jeanpylon/anaconda3/lib/python3.6/site-packages/sklearn/neural_network/multilayer_percept
  % self.max_iter, ConvergenceWarning)


      q=0.0001


/home/jeanpylon/anaconda3/lib/python3.6/site-packages/sklearn/neural_network/multilayer_percept
  % self.max_iter, ConvergenceWarning)
/home/jeanpylon/anaconda3/lib/python3.6/site-packages/sklearn/neural_network/multilayer_percept
  % self.max_iter, ConvergenceWarning)
/home/jeanpylon/anaconda3/lib/python3.6/site-packages/sklearn/neural_network/multilayer_percept
  % self.max_iter, ConvergenceWarning)


      q=0.001


/home/jeanpylon/anaconda3/lib/python3.6/site-packages/sklearn/neural_network/multilayer_percept
  % self.max_iter, ConvergenceWarning)
/home/jeanpylon/anaconda3/lib/python3.6/site-packages/sklearn/neural_network/multilayer_percept
  % self.max_iter, ConvergenceWarning)
/home/jeanpylon/anaconda3/lib/python3.6/site-packages/sklearn/neural_network/multilayer_percept
  % self.max_iter, ConvergenceWarning)


      q=0.01


/home/jeanpylon/anaconda3/lib/python3.6/site-packages/sklearn/neural_network/multilayer_percept
  % self.max_iter, ConvergenceWarning)
```

```
/home/jeanpylon/anaconda3/lib/python3.6/site-packages/sklearn/neural_network/multilayer_percept
  % self.max_iter, ConvergenceWarning)
/home/jeanpylon/anaconda3/lib/python3.6/site-packages/sklearn/neural_network/multilayer_percept
  % self.max_iter, ConvergenceWarning)


      q=0.1


/home/jeanpylon/anaconda3/lib/python3.6/site-packages/sklearn/neural_network/multilayer_percept
  % self.max_iter, ConvergenceWarning)
/home/jeanpylon/anaconda3/lib/python3.6/site-packages/sklearn/neural_network/multilayer_percept
  % self.max_iter, ConvergenceWarning)
/home/jeanpylon/anaconda3/lib/python3.6/site-packages/sklearn/neural_network/multilayer_percept
  % self.max_iter, ConvergenceWarning)


      q=1


/home/jeanpylon/anaconda3/lib/python3.6/site-packages/sklearn/neural_network/multilayer_percept
  % self.max_iter, ConvergenceWarning)
/home/jeanpylon/anaconda3/lib/python3.6/site-packages/sklearn/neural_network/multilayer_percept
  % self.max_iter, ConvergenceWarning)
/home/jeanpylon/anaconda3/lib/python3.6/site-packages/sklearn/neural_network/multilayer_percept
  % self.max_iter, ConvergenceWarning)


   p=(50, 50)
      q=1e-05


/home/jeanpylon/anaconda3/lib/python3.6/site-packages/sklearn/neural_network/multilayer_percept
  % self.max_iter, ConvergenceWarning)
/home/jeanpylon/anaconda3/lib/python3.6/site-packages/sklearn/neural_network/multilayer_percept
  % self.max_iter, ConvergenceWarning)
/home/jeanpylon/anaconda3/lib/python3.6/site-packages/sklearn/neural_network/multilayer_percept
  % self.max_iter, ConvergenceWarning)


      q=0.0001


/home/jeanpylon/anaconda3/lib/python3.6/site-packages/sklearn/neural_network/multilayer_percept
  % self.max_iter, ConvergenceWarning)
/home/jeanpylon/anaconda3/lib/python3.6/site-packages/sklearn/neural_network/multilayer_percept
  % self.max_iter, ConvergenceWarning)
/home/jeanpylon/anaconda3/lib/python3.6/site-packages/sklearn/neural_network/multilayer_percept
  % self.max_iter, ConvergenceWarning)
```

```
      q=0.001


/home/jeanpylon/anaconda3/lib/python3.6/site-packages/sklearn/neural_network/multilayer_percept
  % self.max_iter, ConvergenceWarning)
/home/jeanpylon/anaconda3/lib/python3.6/site-packages/sklearn/neural_network/multilayer_percept
  % self.max_iter, ConvergenceWarning)
/home/jeanpylon/anaconda3/lib/python3.6/site-packages/sklearn/neural_network/multilayer_percept
  % self.max_iter, ConvergenceWarning)


      q=0.01


/home/jeanpylon/anaconda3/lib/python3.6/site-packages/sklearn/neural_network/multilayer_percept
  % self.max_iter, ConvergenceWarning)
/home/jeanpylon/anaconda3/lib/python3.6/site-packages/sklearn/neural_network/multilayer_percept
  % self.max_iter, ConvergenceWarning)
/home/jeanpylon/anaconda3/lib/python3.6/site-packages/sklearn/neural_network/multilayer_percept
  % self.max_iter, ConvergenceWarning)


      q=0.1


/home/jeanpylon/anaconda3/lib/python3.6/site-packages/sklearn/neural_network/multilayer_percept
  % self.max_iter, ConvergenceWarning)
/home/jeanpylon/anaconda3/lib/python3.6/site-packages/sklearn/neural_network/multilayer_percept
  % self.max_iter, ConvergenceWarning)
/home/jeanpylon/anaconda3/lib/python3.6/site-packages/sklearn/neural_network/multilayer_percept
  % self.max_iter, ConvergenceWarning)


      q=1


/home/jeanpylon/anaconda3/lib/python3.6/site-packages/sklearn/neural_network/multilayer_percept
  % self.max_iter, ConvergenceWarning)
/home/jeanpylon/anaconda3/lib/python3.6/site-packages/sklearn/neural_network/multilayer_percept
  % self.max_iter, ConvergenceWarning)


Done.


/home/jeanpylon/anaconda3/lib/python3.6/site-packages/sklearn/neural_network/multilayer_percept
  % self.max_iter, ConvergenceWarning)
```

In [5]:
```
# Plotting needs some special treatment because one of the axis are tuples in this cas
p_list = [str(n) for n in net_nodes_list]
q_list = net_alpha_list
scores_array = mlp_scores

fig, ax = plt.subplots()
im, cbar = heatmap(scores_array, np.array(p_list), np.array(q_list) , ax = ax, cmap =
texts = annotate_heatmap(im, valfmt="{x:.4f}")
fig.tight_layout()
plt.show()
```



In [4]:
```
# Setting up parameterlist
net_nodes_list = [(1500), (750, 750), (500,500,500), (375,375,375,375), (300,300,300,3
net_alpha_list = [.00001, .0001, .001, .01, .1]

# k-fold cross validation over all parameteres and plotting
print("Neural networks: Cross validation over all parameters ...")
mlp_scores2 = mlp_tester(
    X_train, y_train, nodes = net_nodes_list, alpha_list = net_alpha_list, folds = k, p
print("Done.")

# Plotting needs some special treatment because one of the axis are tuples in this cas
p_list = [str(n) for n in net_nodes_list]
q_list = net_alpha_list
```

```
        scores_array = mlp_scores2

        fig, ax = plt.subplots()
        im, cbar = heatmap(scores_array, np.array(p_list), np.array(q_list) , ax = ax, cmap = '
        texts = annotate_heatmap(im, valfmt="{x:.4f}")
        fig.tight_layout()
        plt.show()

Neural networks: Cross validation over all parameters ...
   p=1500
      q=1e-05


/home/jeanpylon/anaconda3/lib/python3.6/site-packages/sklearn/neural_network/multilayer_percept
  % self.max_iter, ConvergenceWarning)
/home/jeanpylon/anaconda3/lib/python3.6/site-packages/sklearn/neural_network/multilayer_percept
  % self.max_iter, ConvergenceWarning)
/home/jeanpylon/anaconda3/lib/python3.6/site-packages/sklearn/neural_network/multilayer_percept
  % self.max_iter, ConvergenceWarning)


      q=0.0001


/home/jeanpylon/anaconda3/lib/python3.6/site-packages/sklearn/neural_network/multilayer_percept
  % self.max_iter, ConvergenceWarning)
/home/jeanpylon/anaconda3/lib/python3.6/site-packages/sklearn/neural_network/multilayer_percept
  % self.max_iter, ConvergenceWarning)
/home/jeanpylon/anaconda3/lib/python3.6/site-packages/sklearn/neural_network/multilayer_percept
  % self.max_iter, ConvergenceWarning)


      q=0.001


/home/jeanpylon/anaconda3/lib/python3.6/site-packages/sklearn/neural_network/multilayer_percept
  % self.max_iter, ConvergenceWarning)
/home/jeanpylon/anaconda3/lib/python3.6/site-packages/sklearn/neural_network/multilayer_percept
  % self.max_iter, ConvergenceWarning)
/home/jeanpylon/anaconda3/lib/python3.6/site-packages/sklearn/neural_network/multilayer_percept
  % self.max_iter, ConvergenceWarning)


      q=0.01


/home/jeanpylon/anaconda3/lib/python3.6/site-packages/sklearn/neural_network/multilayer_percept
  % self.max_iter, ConvergenceWarning)
/home/jeanpylon/anaconda3/lib/python3.6/site-packages/sklearn/neural_network/multilayer_percept
```

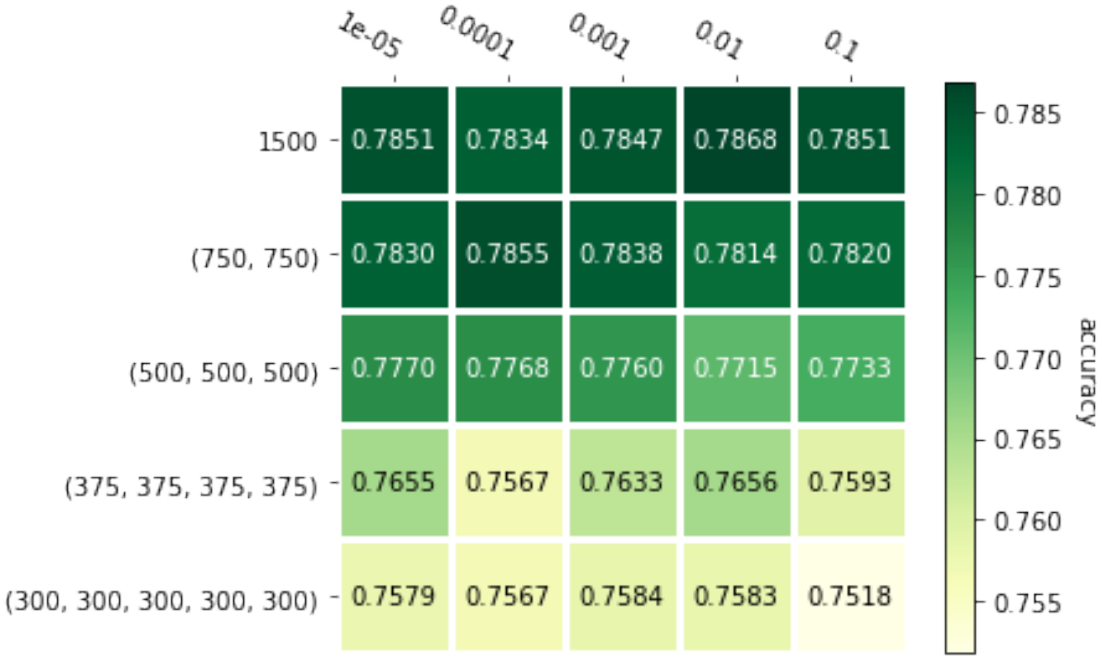
```
  % self.max_iter, ConvergenceWarning)
/home/jeanpylon/anaconda3/lib/python3.6/site-packages/sklearn/neural_network/multilayer_percept
  % self.max_iter, ConvergenceWarning)


      q=0.1


/home/jeanpylon/anaconda3/lib/python3.6/site-packages/sklearn/neural_network/multilayer_percept
  % self.max_iter, ConvergenceWarning)
/home/jeanpylon/anaconda3/lib/python3.6/site-packages/sklearn/neural_network/multilayer_percept
  % self.max_iter, ConvergenceWarning)
/home/jeanpylon/anaconda3/lib/python3.6/site-packages/sklearn/neural_network/multilayer_percept
  % self.max_iter, ConvergenceWarning)


   p=(750, 750)
      q=1e-05


/home/jeanpylon/anaconda3/lib/python3.6/site-packages/sklearn/neural_network/multilayer_percept
  % self.max_iter, ConvergenceWarning)
/home/jeanpylon/anaconda3/lib/python3.6/site-packages/sklearn/neural_network/multilayer_percept
  % self.max_iter, ConvergenceWarning)
/home/jeanpylon/anaconda3/lib/python3.6/site-packages/sklearn/neural_network/multilayer_percept
  % self.max_iter, ConvergenceWarning)


      q=0.0001


/home/jeanpylon/anaconda3/lib/python3.6/site-packages/sklearn/neural_network/multilayer_percept
  % self.max_iter, ConvergenceWarning)
/home/jeanpylon/anaconda3/lib/python3.6/site-packages/sklearn/neural_network/multilayer_percept
  % self.max_iter, ConvergenceWarning)
/home/jeanpylon/anaconda3/lib/python3.6/site-packages/sklearn/neural_network/multilayer_percept
  % self.max_iter, ConvergenceWarning)


      q=0.001


/home/jeanpylon/anaconda3/lib/python3.6/site-packages/sklearn/neural_network/multilayer_percept
  % self.max_iter, ConvergenceWarning)
/home/jeanpylon/anaconda3/lib/python3.6/site-packages/sklearn/neural_network/multilayer_percept
  % self.max_iter, ConvergenceWarning)
/home/jeanpylon/anaconda3/lib/python3.6/site-packages/sklearn/neural_network/multilayer_percept
  % self.max_iter, ConvergenceWarning)
```

```
      q=0.01


/home/jeanpylon/anaconda3/lib/python3.6/site-packages/sklearn/neural_network/multilayer_percept
  % self.max_iter, ConvergenceWarning)
/home/jeanpylon/anaconda3/lib/python3.6/site-packages/sklearn/neural_network/multilayer_percept
  % self.max_iter, ConvergenceWarning)
/home/jeanpylon/anaconda3/lib/python3.6/site-packages/sklearn/neural_network/multilayer_percept
  % self.max_iter, ConvergenceWarning)


      q=0.1


/home/jeanpylon/anaconda3/lib/python3.6/site-packages/sklearn/neural_network/multilayer_percept
  % self.max_iter, ConvergenceWarning)
/home/jeanpylon/anaconda3/lib/python3.6/site-packages/sklearn/neural_network/multilayer_percept
  % self.max_iter, ConvergenceWarning)
/home/jeanpylon/anaconda3/lib/python3.6/site-packages/sklearn/neural_network/multilayer_percept
  % self.max_iter, ConvergenceWarning)


   p=(500, 500, 500)
      q=1e-05


/home/jeanpylon/anaconda3/lib/python3.6/site-packages/sklearn/neural_network/multilayer_percept
  % self.max_iter, ConvergenceWarning)
/home/jeanpylon/anaconda3/lib/python3.6/site-packages/sklearn/neural_network/multilayer_percept
  % self.max_iter, ConvergenceWarning)
/home/jeanpylon/anaconda3/lib/python3.6/site-packages/sklearn/neural_network/multilayer_percept
  % self.max_iter, ConvergenceWarning)


      q=0.0001


/home/jeanpylon/anaconda3/lib/python3.6/site-packages/sklearn/neural_network/multilayer_percept
  % self.max_iter, ConvergenceWarning)
/home/jeanpylon/anaconda3/lib/python3.6/site-packages/sklearn/neural_network/multilayer_percept
  % self.max_iter, ConvergenceWarning)
/home/jeanpylon/anaconda3/lib/python3.6/site-packages/sklearn/neural_network/multilayer_percept
  % self.max_iter, ConvergenceWarning)


      q=0.001


/home/jeanpylon/anaconda3/lib/python3.6/site-packages/sklearn/neural_network/multilayer_percept
  % self.max_iter, ConvergenceWarning)
```

```
/home/jeanpylon/anaconda3/lib/python3.6/site-packages/sklearn/neural_network/multilayer_percept
  % self.max_iter, ConvergenceWarning)
/home/jeanpylon/anaconda3/lib/python3.6/site-packages/sklearn/neural_network/multilayer_percept
  % self.max_iter, ConvergenceWarning)


      q=0.01


/home/jeanpylon/anaconda3/lib/python3.6/site-packages/sklearn/neural_network/multilayer_percept
  % self.max_iter, ConvergenceWarning)
/home/jeanpylon/anaconda3/lib/python3.6/site-packages/sklearn/neural_network/multilayer_percept
  % self.max_iter, ConvergenceWarning)
/home/jeanpylon/anaconda3/lib/python3.6/site-packages/sklearn/neural_network/multilayer_percept
  % self.max_iter, ConvergenceWarning)


      q=0.1


/home/jeanpylon/anaconda3/lib/python3.6/site-packages/sklearn/neural_network/multilayer_percept
  % self.max_iter, ConvergenceWarning)
/home/jeanpylon/anaconda3/lib/python3.6/site-packages/sklearn/neural_network/multilayer_percept
  % self.max_iter, ConvergenceWarning)
/home/jeanpylon/anaconda3/lib/python3.6/site-packages/sklearn/neural_network/multilayer_percept
  % self.max_iter, ConvergenceWarning)


   p=(375, 375, 375, 375)
      q=1e-05


/home/jeanpylon/anaconda3/lib/python3.6/site-packages/sklearn/neural_network/multilayer_percept
  % self.max_iter, ConvergenceWarning)
/home/jeanpylon/anaconda3/lib/python3.6/site-packages/sklearn/neural_network/multilayer_percept
  % self.max_iter, ConvergenceWarning)
/home/jeanpylon/anaconda3/lib/python3.6/site-packages/sklearn/neural_network/multilayer_percept
  % self.max_iter, ConvergenceWarning)


      q=0.0001


/home/jeanpylon/anaconda3/lib/python3.6/site-packages/sklearn/neural_network/multilayer_percept
  % self.max_iter, ConvergenceWarning)
/home/jeanpylon/anaconda3/lib/python3.6/site-packages/sklearn/neural_network/multilayer_percept
  % self.max_iter, ConvergenceWarning)
/home/jeanpylon/anaconda3/lib/python3.6/site-packages/sklearn/neural_network/multilayer_percept
  % self.max_iter, ConvergenceWarning)
```

```
      q=0.001


/home/jeanpylon/anaconda3/lib/python3.6/site-packages/sklearn/neural_network/multilayer_percept
  % self.max_iter, ConvergenceWarning)
/home/jeanpylon/anaconda3/lib/python3.6/site-packages/sklearn/neural_network/multilayer_percept
  % self.max_iter, ConvergenceWarning)
/home/jeanpylon/anaconda3/lib/python3.6/site-packages/sklearn/neural_network/multilayer_percept
  % self.max_iter, ConvergenceWarning)


      q=0.01


/home/jeanpylon/anaconda3/lib/python3.6/site-packages/sklearn/neural_network/multilayer_percept
  % self.max_iter, ConvergenceWarning)
/home/jeanpylon/anaconda3/lib/python3.6/site-packages/sklearn/neural_network/multilayer_percept
  % self.max_iter, ConvergenceWarning)
/home/jeanpylon/anaconda3/lib/python3.6/site-packages/sklearn/neural_network/multilayer_percept
  % self.max_iter, ConvergenceWarning)


      q=0.1


/home/jeanpylon/anaconda3/lib/python3.6/site-packages/sklearn/neural_network/multilayer_percept
  % self.max_iter, ConvergenceWarning)
/home/jeanpylon/anaconda3/lib/python3.6/site-packages/sklearn/neural_network/multilayer_percept
  % self.max_iter, ConvergenceWarning)
/home/jeanpylon/anaconda3/lib/python3.6/site-packages/sklearn/neural_network/multilayer_percept
  % self.max_iter, ConvergenceWarning)


   p=(300, 300, 300, 300, 300)
      q=1e-05


/home/jeanpylon/anaconda3/lib/python3.6/site-packages/sklearn/neural_network/multilayer_percept
  % self.max_iter, ConvergenceWarning)
/home/jeanpylon/anaconda3/lib/python3.6/site-packages/sklearn/neural_network/multilayer_percept
  % self.max_iter, ConvergenceWarning)
/home/jeanpylon/anaconda3/lib/python3.6/site-packages/sklearn/neural_network/multilayer_percept
  % self.max_iter, ConvergenceWarning)


      q=0.0001


/home/jeanpylon/anaconda3/lib/python3.6/site-packages/sklearn/neural_network/multilayer_percept
  % self.max_iter, ConvergenceWarning)
```

```
/home/jeanpylon/anaconda3/lib/python3.6/site-packages/sklearn/neural_network/multilayer_percept
  % self.max_iter, ConvergenceWarning)
/home/jeanpylon/anaconda3/lib/python3.6/site-packages/sklearn/neural_network/multilayer_percept
  % self.max_iter, ConvergenceWarning)


      q=0.001


/home/jeanpylon/anaconda3/lib/python3.6/site-packages/sklearn/neural_network/multilayer_percept
  % self.max_iter, ConvergenceWarning)
/home/jeanpylon/anaconda3/lib/python3.6/site-packages/sklearn/neural_network/multilayer_percept
  % self.max_iter, ConvergenceWarning)
/home/jeanpylon/anaconda3/lib/python3.6/site-packages/sklearn/neural_network/multilayer_percept
  % self.max_iter, ConvergenceWarning)


      q=0.01


/home/jeanpylon/anaconda3/lib/python3.6/site-packages/sklearn/neural_network/multilayer_percept
  % self.max_iter, ConvergenceWarning)
/home/jeanpylon/anaconda3/lib/python3.6/site-packages/sklearn/neural_network/multilayer_percept
  % self.max_iter, ConvergenceWarning)
/home/jeanpylon/anaconda3/lib/python3.6/site-packages/sklearn/neural_network/multilayer_percept
  % self.max_iter, ConvergenceWarning)


      q=0.1


/home/jeanpylon/anaconda3/lib/python3.6/site-packages/sklearn/neural_network/multilayer_percept
  % self.max_iter, ConvergenceWarning)
/home/jeanpylon/anaconda3/lib/python3.6/site-packages/sklearn/neural_network/multilayer_percept
  % self.max_iter, ConvergenceWarning)
/home/jeanpylon/anaconda3/lib/python3.6/site-packages/sklearn/neural_network/multilayer_percept
  % self.max_iter, ConvergenceWarning)


Done.
```



In [3]:
```
# Setting up parameterlist
net_nodes_list = [(900), (1000), (1100), (1200), (1300), (1400), (1500), (1600), (1700)
net_alpha_list = [.00001, .0001, .001, .01, .1]

# k-fold cross validation over all parameteres and plotting
print("Neural networks: Cross validation over all parameters ...")
mlp_scores2 = mlp_tester(
    X_train, y_train, nodes = net_nodes_list, alpha_list = net_alpha_list, folds = k, p
print("Done.")

# Plotting needs some special treatment because one of the axis are tuples in this cas
p_list = [str(n) for n in net_nodes_list]
q_list = net_alpha_list
scores_array = mlp_scores2

fig, ax = plt.subplots()
im, cbar = heatmap(scores_array, np.array(p_list), np.array(q_list) , ax = ax, cmap =
texts = annotate_heatmap(im, valfmt="{x:.4f}")
fig.tight_layout()
plt.show()
```

```
Neural networks: Cross validation over all parameters ...
   p=900
      q=1e-05
```

```
/home/jeanpylon/anaconda3/lib/python3.6/site-packages/sklearn/neural_network/multilayer_percept
  % self.max_iter, ConvergenceWarning)
/home/jeanpylon/anaconda3/lib/python3.6/site-packages/sklearn/neural_network/multilayer_percept
  % self.max_iter, ConvergenceWarning)
/home/jeanpylon/anaconda3/lib/python3.6/site-packages/sklearn/neural_network/multilayer_percept
  % self.max_iter, ConvergenceWarning)
```

```
    q=0.0001
```

```
/home/jeanpylon/anaconda3/lib/python3.6/site-packages/sklearn/neural_network/multilayer_percept
  % self.max_iter, ConvergenceWarning)
/home/jeanpylon/anaconda3/lib/python3.6/site-packages/sklearn/neural_network/multilayer_percept
  % self.max_iter, ConvergenceWarning)
/home/jeanpylon/anaconda3/lib/python3.6/site-packages/sklearn/neural_network/multilayer_percept
  % self.max_iter, ConvergenceWarning)
```

```
    q=0.001
```

```
/home/jeanpylon/anaconda3/lib/python3.6/site-packages/sklearn/neural_network/multilayer_percept
  % self.max_iter, ConvergenceWarning)
/home/jeanpylon/anaconda3/lib/python3.6/site-packages/sklearn/neural_network/multilayer_percept
  % self.max_iter, ConvergenceWarning)
/home/jeanpylon/anaconda3/lib/python3.6/site-packages/sklearn/neural_network/multilayer_percept
  % self.max_iter, ConvergenceWarning)
```

```
    q=0.01
```

```
/home/jeanpylon/anaconda3/lib/python3.6/site-packages/sklearn/neural_network/multilayer_percept
  % self.max_iter, ConvergenceWarning)
/home/jeanpylon/anaconda3/lib/python3.6/site-packages/sklearn/neural_network/multilayer_percept
  % self.max_iter, ConvergenceWarning)
/home/jeanpylon/anaconda3/lib/python3.6/site-packages/sklearn/neural_network/multilayer_percept
  % self.max_iter, ConvergenceWarning)
```

```
    q=0.1
```

```
/home/jeanpylon/anaconda3/lib/python3.6/site-packages/sklearn/neural_network/multilayer_percept
  % self.max_iter, ConvergenceWarning)
/home/jeanpylon/anaconda3/lib/python3.6/site-packages/sklearn/neural_network/multilayer_percept
  % self.max_iter, ConvergenceWarning)
/home/jeanpylon/anaconda3/lib/python3.6/site-packages/sklearn/neural_network/multilayer_percept
  % self.max_iter, ConvergenceWarning)
```

```
   p=1000
      q=1e-05


/home/jeanpylon/anaconda3/lib/python3.6/site-packages/sklearn/neural_network/multilayer_percept
  % self.max_iter, ConvergenceWarning)
/home/jeanpylon/anaconda3/lib/python3.6/site-packages/sklearn/neural_network/multilayer_percept
  % self.max_iter, ConvergenceWarning)
/home/jeanpylon/anaconda3/lib/python3.6/site-packages/sklearn/neural_network/multilayer_percept
  % self.max_iter, ConvergenceWarning)


      q=0.0001


/home/jeanpylon/anaconda3/lib/python3.6/site-packages/sklearn/neural_network/multilayer_percept
  % self.max_iter, ConvergenceWarning)
/home/jeanpylon/anaconda3/lib/python3.6/site-packages/sklearn/neural_network/multilayer_percept
  % self.max_iter, ConvergenceWarning)
/home/jeanpylon/anaconda3/lib/python3.6/site-packages/sklearn/neural_network/multilayer_percept
  % self.max_iter, ConvergenceWarning)


      q=0.001


/home/jeanpylon/anaconda3/lib/python3.6/site-packages/sklearn/neural_network/multilayer_percept
  % self.max_iter, ConvergenceWarning)
/home/jeanpylon/anaconda3/lib/python3.6/site-packages/sklearn/neural_network/multilayer_percept
  % self.max_iter, ConvergenceWarning)
/home/jeanpylon/anaconda3/lib/python3.6/site-packages/sklearn/neural_network/multilayer_percept
  % self.max_iter, ConvergenceWarning)


      q=0.01


/home/jeanpylon/anaconda3/lib/python3.6/site-packages/sklearn/neural_network/multilayer_percept
  % self.max_iter, ConvergenceWarning)
/home/jeanpylon/anaconda3/lib/python3.6/site-packages/sklearn/neural_network/multilayer_percept
  % self.max_iter, ConvergenceWarning)
/home/jeanpylon/anaconda3/lib/python3.6/site-packages/sklearn/neural_network/multilayer_percept
  % self.max_iter, ConvergenceWarning)


      q=0.1


/home/jeanpylon/anaconda3/lib/python3.6/site-packages/sklearn/neural_network/multilayer_percept
  % self.max_iter, ConvergenceWarning)
```

```
/home/jeanpylon/anaconda3/lib/python3.6/site-packages/sklearn/neural_network/multilayer_percept
  % self.max_iter, ConvergenceWarning)
/home/jeanpylon/anaconda3/lib/python3.6/site-packages/sklearn/neural_network/multilayer_percept
  % self.max_iter, ConvergenceWarning)


   p=1100
      q=1e-05


/home/jeanpylon/anaconda3/lib/python3.6/site-packages/sklearn/neural_network/multilayer_percept
  % self.max_iter, ConvergenceWarning)
/home/jeanpylon/anaconda3/lib/python3.6/site-packages/sklearn/neural_network/multilayer_percept
  % self.max_iter, ConvergenceWarning)
/home/jeanpylon/anaconda3/lib/python3.6/site-packages/sklearn/neural_network/multilayer_percept
  % self.max_iter, ConvergenceWarning)


      q=0.0001


/home/jeanpylon/anaconda3/lib/python3.6/site-packages/sklearn/neural_network/multilayer_percept
  % self.max_iter, ConvergenceWarning)
/home/jeanpylon/anaconda3/lib/python3.6/site-packages/sklearn/neural_network/multilayer_percept
  % self.max_iter, ConvergenceWarning)
/home/jeanpylon/anaconda3/lib/python3.6/site-packages/sklearn/neural_network/multilayer_percept
  % self.max_iter, ConvergenceWarning)


      q=0.001


/home/jeanpylon/anaconda3/lib/python3.6/site-packages/sklearn/neural_network/multilayer_percept
  % self.max_iter, ConvergenceWarning)
/home/jeanpylon/anaconda3/lib/python3.6/site-packages/sklearn/neural_network/multilayer_percept
  % self.max_iter, ConvergenceWarning)
/home/jeanpylon/anaconda3/lib/python3.6/site-packages/sklearn/neural_network/multilayer_percept
  % self.max_iter, ConvergenceWarning)


      q=0.01


/home/jeanpylon/anaconda3/lib/python3.6/site-packages/sklearn/neural_network/multilayer_percept
  % self.max_iter, ConvergenceWarning)
/home/jeanpylon/anaconda3/lib/python3.6/site-packages/sklearn/neural_network/multilayer_percept
  % self.max_iter, ConvergenceWarning)
/home/jeanpylon/anaconda3/lib/python3.6/site-packages/sklearn/neural_network/multilayer_percept
  % self.max_iter, ConvergenceWarning)
```

```
      q=0.1


/home/jeanpylon/anaconda3/lib/python3.6/site-packages/sklearn/neural_network/multilayer_percept
  % self.max_iter, ConvergenceWarning)
/home/jeanpylon/anaconda3/lib/python3.6/site-packages/sklearn/neural_network/multilayer_percept
  % self.max_iter, ConvergenceWarning)
/home/jeanpylon/anaconda3/lib/python3.6/site-packages/sklearn/neural_network/multilayer_percept
  % self.max_iter, ConvergenceWarning)


   p=1200
      q=1e-05


/home/jeanpylon/anaconda3/lib/python3.6/site-packages/sklearn/neural_network/multilayer_percept
  % self.max_iter, ConvergenceWarning)
/home/jeanpylon/anaconda3/lib/python3.6/site-packages/sklearn/neural_network/multilayer_percept
  % self.max_iter, ConvergenceWarning)
/home/jeanpylon/anaconda3/lib/python3.6/site-packages/sklearn/neural_network/multilayer_percept
  % self.max_iter, ConvergenceWarning)


      q=0.0001


/home/jeanpylon/anaconda3/lib/python3.6/site-packages/sklearn/neural_network/multilayer_percept
  % self.max_iter, ConvergenceWarning)
/home/jeanpylon/anaconda3/lib/python3.6/site-packages/sklearn/neural_network/multilayer_percept
  % self.max_iter, ConvergenceWarning)
/home/jeanpylon/anaconda3/lib/python3.6/site-packages/sklearn/neural_network/multilayer_percept
  % self.max_iter, ConvergenceWarning)


      q=0.001


/home/jeanpylon/anaconda3/lib/python3.6/site-packages/sklearn/neural_network/multilayer_percept
  % self.max_iter, ConvergenceWarning)
/home/jeanpylon/anaconda3/lib/python3.6/site-packages/sklearn/neural_network/multilayer_percept
  % self.max_iter, ConvergenceWarning)
/home/jeanpylon/anaconda3/lib/python3.6/site-packages/sklearn/neural_network/multilayer_percept
  % self.max_iter, ConvergenceWarning)


      q=0.01


/home/jeanpylon/anaconda3/lib/python3.6/site-packages/sklearn/neural_network/multilayer_percept
  % self.max_iter, ConvergenceWarning)
```

```
/home/jeanpylon/anaconda3/lib/python3.6/site-packages/sklearn/neural_network/multilayer_percept
  % self.max_iter, ConvergenceWarning)
/home/jeanpylon/anaconda3/lib/python3.6/site-packages/sklearn/neural_network/multilayer_percept
  % self.max_iter, ConvergenceWarning)


      q=0.1


/home/jeanpylon/anaconda3/lib/python3.6/site-packages/sklearn/neural_network/multilayer_percept
  % self.max_iter, ConvergenceWarning)
/home/jeanpylon/anaconda3/lib/python3.6/site-packages/sklearn/neural_network/multilayer_percept
  % self.max_iter, ConvergenceWarning)
/home/jeanpylon/anaconda3/lib/python3.6/site-packages/sklearn/neural_network/multilayer_percept
  % self.max_iter, ConvergenceWarning)


   p=1300
      q=1e-05


/home/jeanpylon/anaconda3/lib/python3.6/site-packages/sklearn/neural_network/multilayer_percept
  % self.max_iter, ConvergenceWarning)
/home/jeanpylon/anaconda3/lib/python3.6/site-packages/sklearn/neural_network/multilayer_percept
  % self.max_iter, ConvergenceWarning)
/home/jeanpylon/anaconda3/lib/python3.6/site-packages/sklearn/neural_network/multilayer_percept
  % self.max_iter, ConvergenceWarning)


      q=0.0001


/home/jeanpylon/anaconda3/lib/python3.6/site-packages/sklearn/neural_network/multilayer_percept
  % self.max_iter, ConvergenceWarning)
/home/jeanpylon/anaconda3/lib/python3.6/site-packages/sklearn/neural_network/multilayer_percept
  % self.max_iter, ConvergenceWarning)
/home/jeanpylon/anaconda3/lib/python3.6/site-packages/sklearn/neural_network/multilayer_percept
  % self.max_iter, ConvergenceWarning)


      q=0.001


/home/jeanpylon/anaconda3/lib/python3.6/site-packages/sklearn/neural_network/multilayer_percept
  % self.max_iter, ConvergenceWarning)
/home/jeanpylon/anaconda3/lib/python3.6/site-packages/sklearn/neural_network/multilayer_percept
  % self.max_iter, ConvergenceWarning)
/home/jeanpylon/anaconda3/lib/python3.6/site-packages/sklearn/neural_network/multilayer_percept
  % self.max_iter, ConvergenceWarning)
```

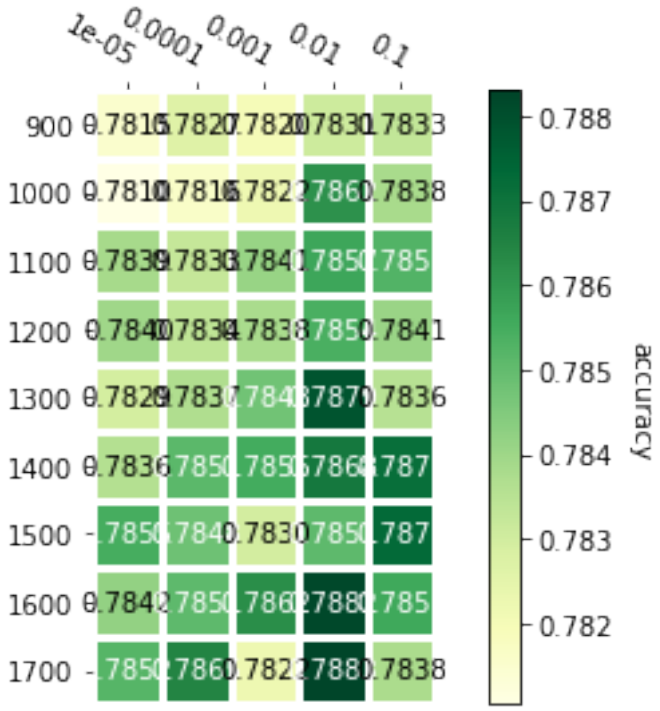
```
      q=0.01


/home/jeanpylon/anaconda3/lib/python3.6/site-packages/sklearn/neural_network/multilayer_percept
  % self.max_iter, ConvergenceWarning)
/home/jeanpylon/anaconda3/lib/python3.6/site-packages/sklearn/neural_network/multilayer_percept
  % self.max_iter, ConvergenceWarning)
/home/jeanpylon/anaconda3/lib/python3.6/site-packages/sklearn/neural_network/multilayer_percept
  % self.max_iter, ConvergenceWarning)


      q=0.1


/home/jeanpylon/anaconda3/lib/python3.6/site-packages/sklearn/neural_network/multilayer_percept
  % self.max_iter, ConvergenceWarning)
/home/jeanpylon/anaconda3/lib/python3.6/site-packages/sklearn/neural_network/multilayer_percept
  % self.max_iter, ConvergenceWarning)
/home/jeanpylon/anaconda3/lib/python3.6/site-packages/sklearn/neural_network/multilayer_percept
  % self.max_iter, ConvergenceWarning)


   p=1400
      q=1e-05


/home/jeanpylon/anaconda3/lib/python3.6/site-packages/sklearn/neural_network/multilayer_percept
  % self.max_iter, ConvergenceWarning)
/home/jeanpylon/anaconda3/lib/python3.6/site-packages/sklearn/neural_network/multilayer_percept
  % self.max_iter, ConvergenceWarning)
/home/jeanpylon/anaconda3/lib/python3.6/site-packages/sklearn/neural_network/multilayer_percept
  % self.max_iter, ConvergenceWarning)


      q=0.0001


/home/jeanpylon/anaconda3/lib/python3.6/site-packages/sklearn/neural_network/multilayer_percept
  % self.max_iter, ConvergenceWarning)
/home/jeanpylon/anaconda3/lib/python3.6/site-packages/sklearn/neural_network/multilayer_percept
  % self.max_iter, ConvergenceWarning)
/home/jeanpylon/anaconda3/lib/python3.6/site-packages/sklearn/neural_network/multilayer_percept
  % self.max_iter, ConvergenceWarning)


      q=0.001


/home/jeanpylon/anaconda3/lib/python3.6/site-packages/sklearn/neural_network/multilayer_percept
  % self.max_iter, ConvergenceWarning)
```

```
/home/jeanpylon/anaconda3/lib/python3.6/site-packages/sklearn/neural_network/multilayer_percept
  % self.max_iter, ConvergenceWarning)
/home/jeanpylon/anaconda3/lib/python3.6/site-packages/sklearn/neural_network/multilayer_percept
  % self.max_iter, ConvergenceWarning)


      q=0.01


/home/jeanpylon/anaconda3/lib/python3.6/site-packages/sklearn/neural_network/multilayer_percept
  % self.max_iter, ConvergenceWarning)
/home/jeanpylon/anaconda3/lib/python3.6/site-packages/sklearn/neural_network/multilayer_percept
  % self.max_iter, ConvergenceWarning)
/home/jeanpylon/anaconda3/lib/python3.6/site-packages/sklearn/neural_network/multilayer_percept
  % self.max_iter, ConvergenceWarning)


      q=0.1


/home/jeanpylon/anaconda3/lib/python3.6/site-packages/sklearn/neural_network/multilayer_percept
  % self.max_iter, ConvergenceWarning)
/home/jeanpylon/anaconda3/lib/python3.6/site-packages/sklearn/neural_network/multilayer_percept
  % self.max_iter, ConvergenceWarning)
/home/jeanpylon/anaconda3/lib/python3.6/site-packages/sklearn/neural_network/multilayer_percept
  % self.max_iter, ConvergenceWarning)


   p=1500
      q=1e-05


/home/jeanpylon/anaconda3/lib/python3.6/site-packages/sklearn/neural_network/multilayer_percept
  % self.max_iter, ConvergenceWarning)
/home/jeanpylon/anaconda3/lib/python3.6/site-packages/sklearn/neural_network/multilayer_percept
  % self.max_iter, ConvergenceWarning)
/home/jeanpylon/anaconda3/lib/python3.6/site-packages/sklearn/neural_network/multilayer_percept
  % self.max_iter, ConvergenceWarning)


      q=0.0001


/home/jeanpylon/anaconda3/lib/python3.6/site-packages/sklearn/neural_network/multilayer_percept
  % self.max_iter, ConvergenceWarning)
/home/jeanpylon/anaconda3/lib/python3.6/site-packages/sklearn/neural_network/multilayer_percept
  % self.max_iter, ConvergenceWarning)
/home/jeanpylon/anaconda3/lib/python3.6/site-packages/sklearn/neural_network/multilayer_percept
  % self.max_iter, ConvergenceWarning)
```

```
      q=0.001


/home/jeanpylon/anaconda3/lib/python3.6/site-packages/sklearn/neural_network/multilayer_percept
  % self.max_iter, ConvergenceWarning)
/home/jeanpylon/anaconda3/lib/python3.6/site-packages/sklearn/neural_network/multilayer_percept
  % self.max_iter, ConvergenceWarning)
/home/jeanpylon/anaconda3/lib/python3.6/site-packages/sklearn/neural_network/multilayer_percept
  % self.max_iter, ConvergenceWarning)


      q=0.01


/home/jeanpylon/anaconda3/lib/python3.6/site-packages/sklearn/neural_network/multilayer_percept
  % self.max_iter, ConvergenceWarning)
/home/jeanpylon/anaconda3/lib/python3.6/site-packages/sklearn/neural_network/multilayer_percept
  % self.max_iter, ConvergenceWarning)
/home/jeanpylon/anaconda3/lib/python3.6/site-packages/sklearn/neural_network/multilayer_percept
  % self.max_iter, ConvergenceWarning)


      q=0.1


/home/jeanpylon/anaconda3/lib/python3.6/site-packages/sklearn/neural_network/multilayer_percept
  % self.max_iter, ConvergenceWarning)
/home/jeanpylon/anaconda3/lib/python3.6/site-packages/sklearn/neural_network/multilayer_percept
  % self.max_iter, ConvergenceWarning)
/home/jeanpylon/anaconda3/lib/python3.6/site-packages/sklearn/neural_network/multilayer_percept
  % self.max_iter, ConvergenceWarning)


   p=1600
      q=1e-05


/home/jeanpylon/anaconda3/lib/python3.6/site-packages/sklearn/neural_network/multilayer_percept
  % self.max_iter, ConvergenceWarning)
/home/jeanpylon/anaconda3/lib/python3.6/site-packages/sklearn/neural_network/multilayer_percept
  % self.max_iter, ConvergenceWarning)
/home/jeanpylon/anaconda3/lib/python3.6/site-packages/sklearn/neural_network/multilayer_percept
  % self.max_iter, ConvergenceWarning)


      q=0.0001


/home/jeanpylon/anaconda3/lib/python3.6/site-packages/sklearn/neural_network/multilayer_percept
  % self.max_iter, ConvergenceWarning)
```

```
/home/jeanpylon/anaconda3/lib/python3.6/site-packages/sklearn/neural_network/multilayer_percept
  % self.max_iter, ConvergenceWarning)
/home/jeanpylon/anaconda3/lib/python3.6/site-packages/sklearn/neural_network/multilayer_percept
  % self.max_iter, ConvergenceWarning)


      q=0.001


/home/jeanpylon/anaconda3/lib/python3.6/site-packages/sklearn/neural_network/multilayer_percept
  % self.max_iter, ConvergenceWarning)
/home/jeanpylon/anaconda3/lib/python3.6/site-packages/sklearn/neural_network/multilayer_percept
  % self.max_iter, ConvergenceWarning)
/home/jeanpylon/anaconda3/lib/python3.6/site-packages/sklearn/neural_network/multilayer_percept
  % self.max_iter, ConvergenceWarning)


      q=0.01


/home/jeanpylon/anaconda3/lib/python3.6/site-packages/sklearn/neural_network/multilayer_percept
  % self.max_iter, ConvergenceWarning)
/home/jeanpylon/anaconda3/lib/python3.6/site-packages/sklearn/neural_network/multilayer_percept
  % self.max_iter, ConvergenceWarning)
/home/jeanpylon/anaconda3/lib/python3.6/site-packages/sklearn/neural_network/multilayer_percept
  % self.max_iter, ConvergenceWarning)


      q=0.1


/home/jeanpylon/anaconda3/lib/python3.6/site-packages/sklearn/neural_network/multilayer_percept
  % self.max_iter, ConvergenceWarning)
/home/jeanpylon/anaconda3/lib/python3.6/site-packages/sklearn/neural_network/multilayer_percept
  % self.max_iter, ConvergenceWarning)
/home/jeanpylon/anaconda3/lib/python3.6/site-packages/sklearn/neural_network/multilayer_percept
  % self.max_iter, ConvergenceWarning)


   p=1700
      q=1e-05


/home/jeanpylon/anaconda3/lib/python3.6/site-packages/sklearn/neural_network/multilayer_percept
  % self.max_iter, ConvergenceWarning)
/home/jeanpylon/anaconda3/lib/python3.6/site-packages/sklearn/neural_network/multilayer_percept
  % self.max_iter, ConvergenceWarning)
/home/jeanpylon/anaconda3/lib/python3.6/site-packages/sklearn/neural_network/multilayer_percept
  % self.max_iter, ConvergenceWarning)
```

```
      q=0.0001


/home/jeanpylon/anaconda3/lib/python3.6/site-packages/sklearn/neural_network/multilayer_percept
  % self.max_iter, ConvergenceWarning)
/home/jeanpylon/anaconda3/lib/python3.6/site-packages/sklearn/neural_network/multilayer_percept
  % self.max_iter, ConvergenceWarning)
/home/jeanpylon/anaconda3/lib/python3.6/site-packages/sklearn/neural_network/multilayer_percept
  % self.max_iter, ConvergenceWarning)


      q=0.001


/home/jeanpylon/anaconda3/lib/python3.6/site-packages/sklearn/neural_network/multilayer_percept
  % self.max_iter, ConvergenceWarning)
/home/jeanpylon/anaconda3/lib/python3.6/site-packages/sklearn/neural_network/multilayer_percept
  % self.max_iter, ConvergenceWarning)
/home/jeanpylon/anaconda3/lib/python3.6/site-packages/sklearn/neural_network/multilayer_percept
  % self.max_iter, ConvergenceWarning)


      q=0.01


/home/jeanpylon/anaconda3/lib/python3.6/site-packages/sklearn/neural_network/multilayer_percept
  % self.max_iter, ConvergenceWarning)
/home/jeanpylon/anaconda3/lib/python3.6/site-packages/sklearn/neural_network/multilayer_percept
  % self.max_iter, ConvergenceWarning)
/home/jeanpylon/anaconda3/lib/python3.6/site-packages/sklearn/neural_network/multilayer_percept
  % self.max_iter, ConvergenceWarning)


      q=0.1


/home/jeanpylon/anaconda3/lib/python3.6/site-packages/sklearn/neural_network/multilayer_percept
  % self.max_iter, ConvergenceWarning)
/home/jeanpylon/anaconda3/lib/python3.6/site-packages/sklearn/neural_network/multilayer_percept
  % self.max_iter, ConvergenceWarning)
/home/jeanpylon/anaconda3/lib/python3.6/site-packages/sklearn/neural_network/multilayer_percept
  % self.max_iter, ConvergenceWarning)


Done.
```




```
In [4]: print(scores_array)
```

```
[[0.78154103 0.78267256 0.78199324 0.78310055 0.78332669]
 [0.78103846 0.78164221 0.78222034 0.78616682 0.78382918]
 [0.78385464 0.78327624 0.78410562 0.78568937 0.78528798]
 [0.78398004 0.78337646 0.78380405 0.78543754 0.78408112]
 [0.78294928 0.7837029  0.78475982 0.78790226 0.78355222]
 [0.78360279 0.78513618 0.78553889 0.78684586 0.78714722]
 [0.78548867 0.78483555 0.78297442 0.78508694 0.78729845]
 [0.7842061  0.78508635 0.78624259 0.78820385 0.78561346]
 [0.78521217 0.78646878 0.7822202  0.78830423 0.78377868]]
```

so we proceed with alpha=.01 and 1700 nodes, perhaps

6 Voting classifiers

6.1 Setting up the voting classifier

First we set up the basic classifiers with their optimal parameters

```
In [3]: logistic_clf = LogisticRegression(solver='lbfgs', multi_class='multinomial', C = 1)
        forrest_clf = RandomForestClassifier(n_estimators = 100, max_depth = None)
        mlp_clf = MLPClassifier(hidden_layer_sizes = (1000), alpha = 0.01, max_iter = 10)
        svm_clf = svm.LinearSVC(C = 0.1)
```

```
In [4]: #### We will ignore warnings from here on out
        # Voting classifiers allways throws a warning,
        # and since we have set max_iter = 10,
        # we will also get a warning whenever we try to fit the mlp_clf.
        import warnings
        warnings.simplefilter('ignore')
```

6.2 Trying different voting patterns

Now we go through combinations and voting styles. First we look at two classifiers voting. Here it only makes sense to use soft voting.

```
In [5]: voting_lf_soft = VotingClassifier(estimators=[('logistic', logistic_clf), ('forrest', f
        print("Cross validation for soft voting with logistic and forrest.")
        score_lf_soft = np.mean(cross_val_score(voting_lf_soft, X_train, y_train, cv=3))
        print("The expected accuracy is %f" % score_lf_soft)
```

```
Cross validation for soft voting with logistic and forrest.
The expected accuracy is 0.793308
```

```
In [6]: voting_lmlp_soft = VotingClassifier(estimators=[('logistic', logistic_clf), ('mlp', mlp
        print("Cross validation for soft voting with logistic and mlp.")
        score_lmlp_soft = np.mean(cross_val_score(voting_lmlp_soft, X_train, y_train, cv=3))
        print("The expected accuracy is %f" % score_lmlp_soft)
```

```
Cross validation for soft voting with logistic and mlp.
The expected accuracy is 0.785866
```

```
In [7]: voting_fmlp_soft = VotingClassifier(estimators=[('forrest', forrest_clf), ('mlp', mlp_c
        print("Cross validation for soft voting with forrest and mlp.")
        score_fmlp_soft = np.mean(cross_val_score(voting_fmlp_soft, X_train, y_train, cv=3))
        print("The expected accuracy is %f" % score_fmlp_soft)
```

```
Cross validation for soft voting with forrest and mlp.
The expected accuracy is 0.798361
```

Then we try soft voting with all the probabalistic classifiers.

```
In [8]: voting_all_soft = VotingClassifier(estimators=[('forrest', forrest_clf), ('mlp', mlp_cl
        print("Cross validation for soft voting with forrest, mlp, and logistic.")
        score_all_soft = np.mean(cross_val_score(voting_all_soft, X_train, y_train, cv=3))
        print("The expected accuracy is %f" % score_all_soft)
```

```
Cross validation for soft voting with forrest, mlp, and logistic.
The expected accuracy is 0.796149
```

Then we try hard voting, using mlp, forrest and svm

```
In [9]: voting_hard = VotingClassifier(estimators=[('svm', svm_clf), ('forrest', forrest_clf),
        print("Cross validation for hard voting with svm and forrests and mlp.")
        score_hard = np.mean(cross_val_score(voting_hard, X_train, y_train, cv=3))
        print("The expected accuracy is %f" % score_hard)
```

```
Cross validation for hard voting with svm and forrests and mlp.
The expected accuracy is 0.798084
```

We also try hard voting using all four methods

```
In [10]: voting_hard_4 = VotingClassifier(estimators=[('svm', svm_clf), ('logistic', logistic_
         print("Cross validation for hard voting with svm, logistic, forrests, and mlp.")
         score_hard_4 = np.mean(cross_val_score(voting_hard_4, X_train, y_train, cv=3))
         print("The expected accuracy is %f" % score_hard_4)
```

```
Cross validation for hard voting with svm, logistic, forrests, and mlp.
The expected accuracy is 0.790843
```

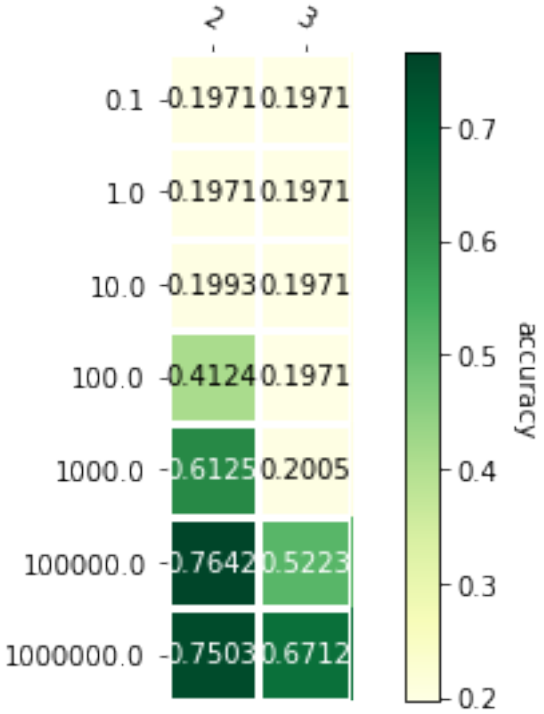
7 SVM Kernels

Now we look at two svm kernels: Poly and rbf.

```
In [12]: def poly_svm_tester(X_train, y_train, C_list = [0.1], degrees = [2], folds = 10, plot
             """
             Test the svm parameter C and the degree of the poly kernel
             using cross validation

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

             svm_constructor = (lambda p,q : svm.SVC(kernel = 'poly', C = p, degree = q))
             scores = clf_cross_validator(X_train, y_train, svm_constructor, C_list, degrees, 
             return scores

         poly_svm_tester(X_train, y_train, C_list = [0.1, 1, 10, 100, 1000, 100000, 1000000], 
```

```
p=0.1
   q=2
   q=3
p=1
   q=2
   q=3
p=10
   q=2
```

```
    q=3
p=100
    q=2
    q=3
p=1000
    q=2
    q=3
p=100000
    q=2
    q=3
p=1000000
    q=2
    q=3
```

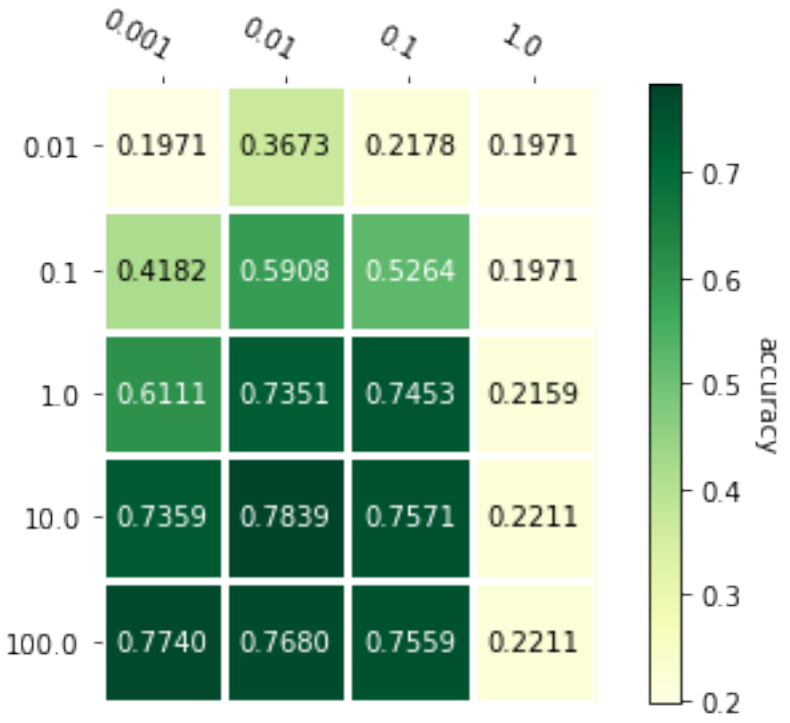



```
Out[12]: array([[0.19706342, 0.19706342],
                [0.19706342, 0.19706342],
                [0.19932623, 0.19706342],
                [0.4123805 , 0.19706342],
                [0.61248489, 0.20045765],
                [0.76421728, 0.52227571],
                [0.75028845, 0.67121641]])
```

```
In [3]: def rbf_svm_tester(X_train, y_train, C_list = [0.1], gamma_list = [0.1], folds = 10, pl
            """
```

```
        Test the svm parameters C and gamma of svm with rbf kernel
        using cross validation

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

        svm_constructor = (lambda p,q : svm.SVC(kernel = 'rbf', C = p, gamma = q))
        scores = clf_cross_validator(X_train, y_train, svm_constructor, C_list, gamma_list
        return scores

    rbf_svm_tester(X_train, y_train, C_list = [0.01, 0.1,1, 10, 100], gamma_list = [0.001,
```

```
p=0.01
   q=0.001
   q=0.01
   q=0.1
   q=1
p=0.1
   q=0.001
   q=0.01
   q=0.1
   q=1
p=1
   q=0.001
   q=0.01
   q=0.1
   q=1
p=10
   q=0.001
   q=0.01
   q=0.1
   q=1
p=100
   q=0.001
   q=0.01
   q=0.1
   q=1
```



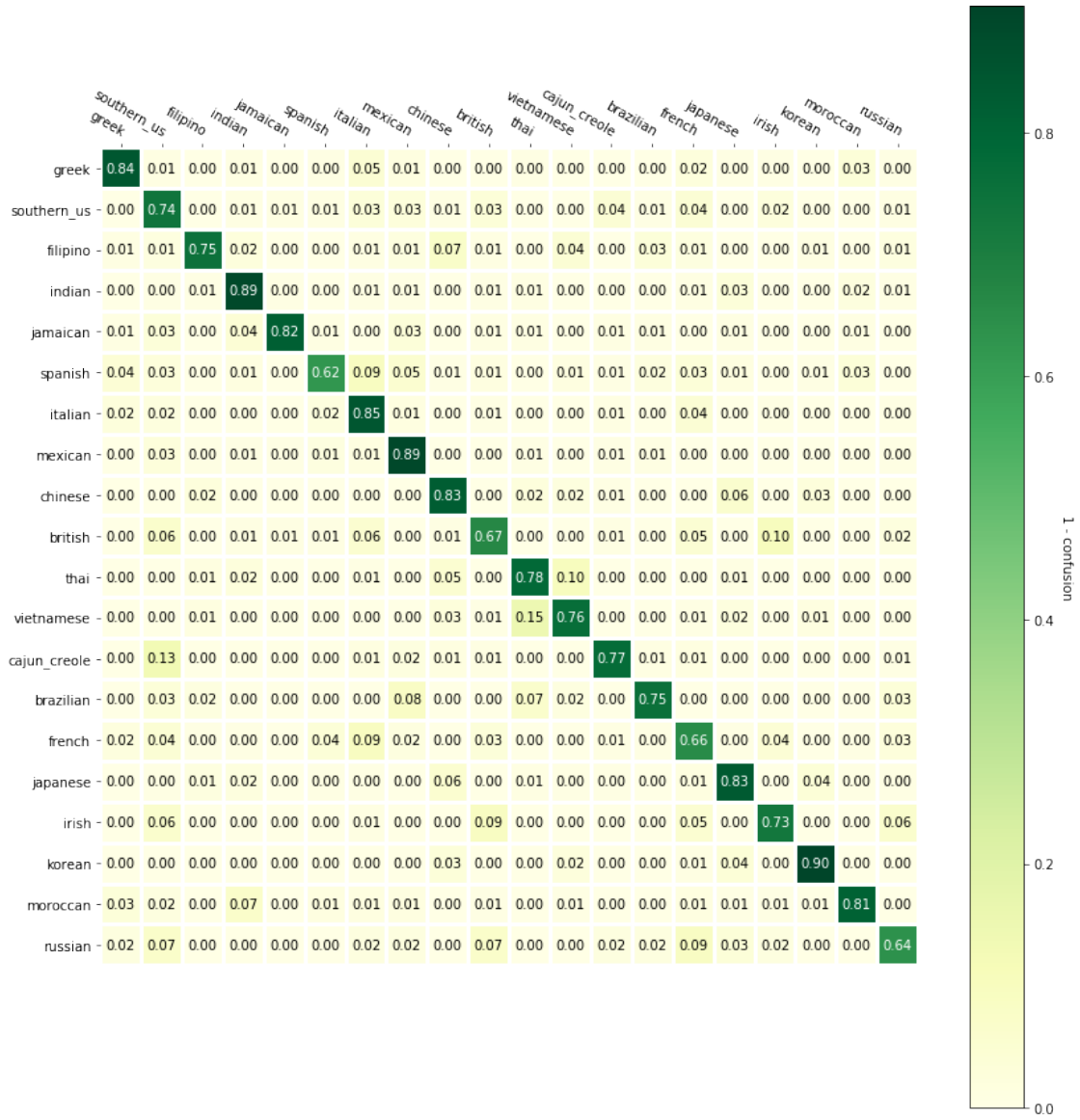

```
Out[3]: array([[0.19706342, 0.36730029, 0.21778012, 0.19706342],
               [0.41816275, 0.59083871, 0.52642476, 0.19706342],
               [0.61107811, 0.73507861, 0.74531125, 0.21594517],
               [0.73593335, 0.78387868, 0.75707763, 0.22109935],
               [0.77397349, 0.76796342, 0.75592098, 0.22109935]])
```

7.1 Confusion matrix

```
In [4]: x_train, x_test, y, y_test = train_test_split(X_train, y_train, test_size = 0.2)

        # The set of different cuisines
        data = pd.read_json('train.json')
        cuisines = data.cuisine.unique()

        logistic_clf = LogisticRegression(solver='lbfgs', multi_class='multinomial', C = 1)
        forrest_clf = RandomForestClassifier(n_estimators = 100, max_depth = None)
        mlp_clf = MLPClassifier(hidden_layer_sizes = (1000), alpha = 0.01, max_iter = 10)
        clf = VotingClassifier(estimators=[('forrest', forrest_clf), ('mlp', mlp_clf), ('logist
                                           voting='soft')
        clf_confusion(clf, x_train, y, x_test, y_test, cuisines, size = (12,12), normalize = '
```

```
C:\Users\Admin\Anaconda3\lib\site-packages\sklearn\neural_network\multilayer_perceptron.py:564
  % self.max_iter, ConvergenceWarning)
C:\Users\Admin\Anaconda3\lib\site-packages\sklearn\preprocessing\label.py:151: DeprecationWarni
```

```
if diff:
```



7.2 PCA test for svm

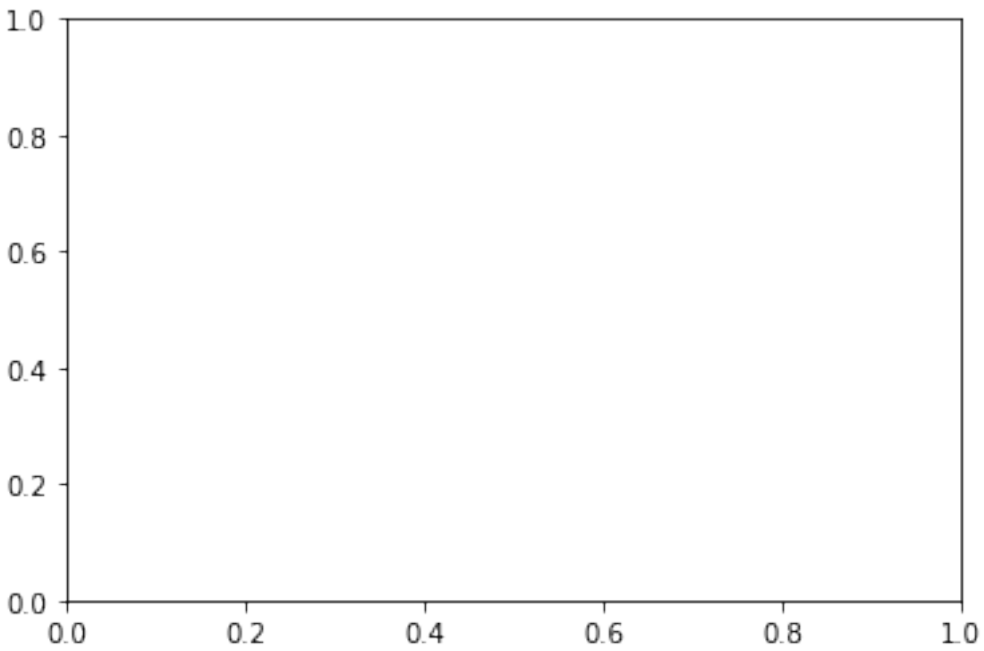
```
In [9]: svm_clf = svm.LinearSVC()

        scores = []

        pca_sizes = [500, 1000, 1500, 1800,2000]
        C_list = [0.01, 0.1, 1]
```

```
        X_array = X_train.toarray()

        for n in pca_sizes :
            print("Testing pca with %d components." % n)
            pca = PCA(n_components = n)
            pca.fit(X_array)
            pca_X = pca.transform(X_array)
            for C in C_list :
                print("    Testing C = %f" % C)
                svm_clf = svm.LinearSVC(C = C)
                score = np.mean(cross_val_score(svm_clf, pca_X, y_train, cv=3))
                scores.append(score)
                print("      Score: %f" % score)
            print("")

        # transform the scores to a len(pca_sizes) x len(X_list) shape array
        scores_array = np.array(scores).reshape(len(pca_sizes), len(C_list))

        # make heat map
        fig, ax = plt.subplots()
        im, cbar = heatmap(scores_array, np.array(pca_sizes), np.array(C_list) , ax = ax, cmap
        texts = annotate_heatmap(im, valfmt="{x:.4f}")
        fig.tight_layout()
        plt.show()
```

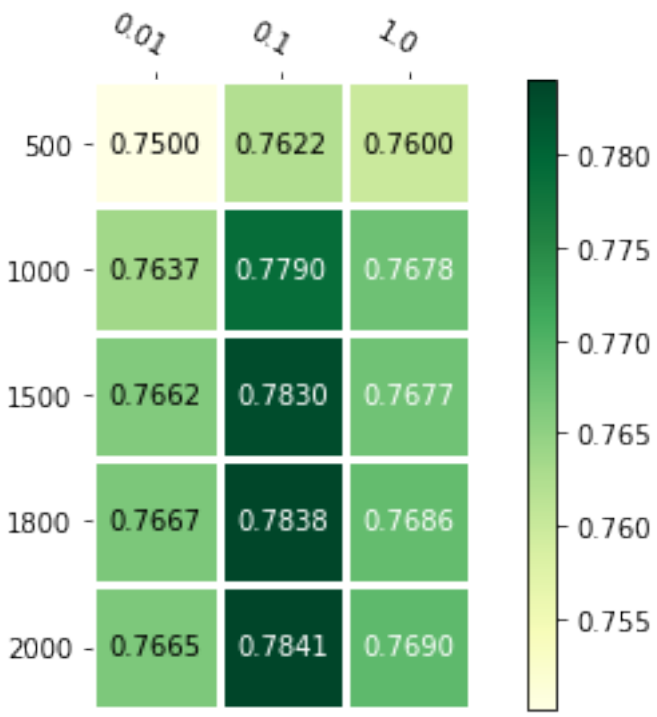
```
Testing pca with 500 components.
    Testing C = 0.010000
      Score: 0.750037
    Testing C = 0.100000
      Score: 0.762181
    Testing C = 1.000000
      Score: 0.759994

Testing pca with 1000 components.
    Testing C = 0.010000
      Score: 0.763689
    Testing C = 0.100000
      Score: 0.778976
    Testing C = 1.000000
      Score: 0.767839

Testing pca with 1500 components.
    Testing C = 0.010000
      Score: 0.766229
    Testing C = 0.100000
      Score: 0.782999
    Testing C = 1.000000
      Score: 0.767688
```

```
Testing pca with 1800 components.
    Testing C = 0.010000
      Score: 0.766706
    Testing C = 0.100000
      Score: 0.783803
    Testing C = 1.000000
      Score: 0.768618

Testing pca with 2000 components.
    Testing C = 0.010000
      Score: 0.766530
    Testing C = 0.100000
      Score: 0.784055
    Testing C = 1.000000
      Score: 0.769020
```

```
NameError                                 Traceback (most recent call last)

<ipython-input-9-ee4bb6a00412> in <module>()
 26 # make heat map
 27 fig, ax = plt.subplots()
---> 28 im, cbar = heatmap(scores_array, np.array(pca_sizes), np.array(C_list) , ax = ax, c
 29 texts = annotate_heatmap(im, valfmt="{x:.4f}")
 30 fig.tight_layout()


NameError: name 'label' is not defined
```



In [10]:
```
# make heat map
fig, ax = plt.subplots()
im, cbar = heatmap(scores_array, np.array(pca_sizes), np.array(C_list) , ax = ax, cma
texts = annotate_heatmap(im, valfmt="{x:.4f}")
fig.tight_layout()

plt.show()
```



In [ ]:
```
# Use 2-dim pca
two_dim_pca = PCA(n_components = 2)
two_dim_pca.fit(X_train)

# random list of 20 colors for plotting
color_list = ['#'+''.join([random.choice('0123456789ABCDEF') for j in range(6)]) for _
cusinine_to_int = {}
x_pca = two_dim_pca.transform(x_train)
colors =[color_list[i] for i in y_train]
```

Kaggle submission script

December 17, 2018

In [5]:
```
# This Python 3 environment comes with many helpful analytics libraries installed
# It is defined by the kaggle/python docker image: https://github.com/kaggle/docker-py
# For example, here's several helpful packages to load in

import numpy as np # linear algebra
import pandas as pd # data processing, CSV file I/O (e.g. pd.read_csv)

# Input data files are available in the "../input/" directory.
# For example, running this (by clicking run or pressing Shift+Enter) will list the fi

# Any results you write to the current directory are saved as output.
from sklearn import svm #support vector machines
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.ensemble import RandomForestClassifier, VotingClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.neural_network import MLPClassifier

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


#### First we read in the training data
data = pd.read_json('../input/train.json')
recipie_list_list = data.ingredients.values.tolist()
recipie_string_list = [clean(" ".join(ing)) for ing in recipie_list_list]
vectorizer = CountVectorizer(min_df = 3)
X_train = vectorizer.fit_transform(recipie_string_list)
y_train = data.cuisine.values

del data, recipie_list_list, recipie_string_list

#### Then we read in the test data
data = pd.read_json('../input/test.json')
recipie_list_list = data.ingredients.values.tolist()
recipie_string_list = [clean(" ".join(ing)) for ing in recipie_list_list]
```

```
        X_test = vectorizer.transform(recipie_string_list)
        test_ids = data.id.values.tolist()
        del data, recipie_list_list, recipie_string_list

        ##############
        ##############
        ## The classifier goes here
        #############
        #############
        logistic_clf = LogisticRegression(solver='lbfgs', multi_class='multinomial', C = 1)
        forrest_clf = RandomForestClassifier(n_estimators = 100, max_depth = None)
        mlp_clf = MLPClassifier(hidden_layer_sizes = (1000), alpha = 0.01, max_iter = 10)
        clf = VotingClassifier(estimators=[('forrest', forrest_clf), ('mlp', mlp_clf), ('logist
                                           voting='soft')


        #############
        #############
        ## train  classifier and predict
        clf.fit(X_train,y_train)
        predictions = clf.predict(X_test)


        ##############
        ##############
        ## Write prediction as output file
        #############
        #############
        idpreds = zip(test_ids, predictions)
        file = open('sub.csv', 'w')
        file.write('id,cuisine\n')
        for t in idpreds :
            file.write(str(t[0])+','+t[1]+'\n')
```

```
<zip object at 0x000001C571393588>
```

RandomForest.py

December 17, 2018

In [ ]:
```
import sys
import warnings

if not sys.warnoptions:
    warnings.simplefilter("ignore")

import numpy as np
import pandas as pd
import re
import tensorflow as tf
from sklearn.model_selection import train_test_split
import matplotlib.pyplot as plt
import random
from sklearn.metrics import r2_score, mean_squared_error, accuracy_score, log_loss
import matplotlib.pyplot as plt

from sklearn.ensemble import RandomForestClassifier
from sklearn.tree import DecisionTreeClassifier



def easydatagen():
    """
    This function generates training data
    It returns:
    x_train, x_test, y_train, y_test
    """

    # Reading in the training file
    data = pd.read_json('train.json')

    # The set of different cuisines
    cuisines = data.cuisine.unique()

    # To find the different ingredients, we need to clean them up a little.
    def clean(string) :
        s = string.replace('-',' ') # read low-fat the same as low fat
```

```
    s = string.replace('&', 'and') # read & and and as the same
    s = re.sub('\((.*?)\)', '', s) # remove everythin g in brackets
    s = re.sub('\d{1,2}\%', '', s) # remove things of the form d% or dd%, where d
    s = ' '.join(s.split()) # remove extra white spaces

    return s

ing_list = data.ingredients.values.tolist()
raw_ingredients = [clean(x) for ing in ing_list for x in ing]

ingredients = sorted(set(raw_ingredients))

# build a dictionary that to each ingredient assigns its index
ingredient_index = {}
for i in range(0,len(ingredients)) :
    ingredient_index[ingredients[i]] = i

# the same for cuisines
cuisine_index = {}
for i in range(0, len(cuisines)) :
    cuisine_index[cuisines[i]] = i

def ingredients_to_vector(ings) :
    vect = np.zeros(len(ingredients))
    for ing in ings :
        vect[ingredient_index[clean(ing)]] = 1

    return vect

def cuisine_to_vector(cus) :
    vect = np.zeros(20)
    vect[cuisine_index[cus]] = 1
    return vect

vect_list = [ingredients_to_vector(ing) for ing in ing_list]
target_list = [cuisine_to_vector(cus) for cus in data.cuisine.values.tolist()]

# Define training data
X = np.c_[vect_list]
Y = np.c_[target_list]

Y_num = np.zeros((Y.shape[0]))
for i in range(Y.shape[0]):
    Y_num[i] = np.argmax(Y[i])

x_train, x_test, y_train, y_test = train_test_split(X, Y_num, test_size = 0.2)

return x_train, x_test, y_train, y_test
```

```
if __name__ == '__main__':

    x_train, x_test, y_train, y_test = easydatagen()

    """
    Next code plots 32treesAcc.png
    """

    print('Starting training:')

    testscores = []
    trainscores = []
    index = []

    for i in range(1,32):
        clf = RandomForestClassifier(n_estimators=i, max_depth=None, max_features='auto
                                     verbose=True, n_jobs=8)
        clf.fit(x_train, y_train)

        index.append(i)
        testscores.append(clf.score(x_test, y_test))
        trainscores.append(clf.score(x_train, y_train))

    plt.plot(index, testscores, '-b' , label='Testing data')
    plt.plot(index, trainscores, '-r', label='Training data')
    plt.legend(loc='center right')
    plt.title('Accuracy of random forests')
    plt.xlabel('Trees')
    plt.ylabel('Accuracy')
    plt.show()
    # Plot tells us that we can use 10 trees to get a decent score




    """
    This next code plots Max_featuresplot.png
    """
    print('Starting training:')
    testscores = []
    trainscores = []
    index = []

    """
    # Auto = sqrt(classifiers) = 81     log2 = 12
    # Use then different values as max_features
    """
```

```
for i in range(1, 200, 10):
    clf = RandomForestClassifier(n_estimators=10, max_depth=None, max_features= i,
                                 verbose=True, n_jobs=8)
    clf.fit(x_train, y_train)

    index.append(i)
    testscores.append(clf.score(x_test, y_test))
    trainscores.append(clf.score(x_train, y_train))

plt.plot(index, testscores, '-b' , label='Testing data')
plt.plot(index, trainscores, '-r', label='Training data')
plt.legend(loc='center right')
plt.title('Accuracy of random forests')
plt.xlabel('Max_features')
plt.ylabel('Accuracy')
plt.show()

# Nothing conclusive
```