

# ANNADULT

March 24, 2018

```
In [1]: import scipy.io as sio
import matplotlib.pyplot as plt
import numpy as np
import seaborn as sns
import pandas as pd
from sklearn.neural_network import MLPClassifier
from sklearn.model_selection import GridSearchCV
%config InlineBackend.figure_format = 'retina'

In [2]: df = pd.read_csv('C:/Users/Preston Wong/Downloads/adult.csv', index_col = 0)
df = df.dropna(axis=0)
onehot = pd.get_dummies(data=df, columns=['Column2', 'Column4', 'Column6', 'Column8', 'Column10', 'Column12', 'Column14', 'Column16', 'Column18', 'Column20', 'Column22', 'Column24', 'Column26', 'Column28', 'Column30', 'Column32', 'Column34', 'Column36', 'Column38', 'Column40', 'Column42', 'Column44', 'Column46', 'Column48', 'Column50', 'Column52', 'Column54', 'Column56', 'Column58', 'Column60', 'Column62', 'Column64', 'Column66', 'Column68', 'Column70', 'Column72', 'Column74', 'Column76', 'Column78', 'Column80', 'Column82', 'Column84', 'Column86', 'Column88', 'Column90', 'Column92', 'Column94', 'Column96', 'Column98', 'Column100', 'Column102', 'Column104', 'Column106', 'Column108', 'Column110', 'Column112', 'Column114', 'Column116', 'Column118', 'Column120', 'Column122', 'Column124', 'Column126', 'Column128', 'Column130', 'Column132', 'Column134', 'Column136', 'Column138', 'Column140', 'Column142', 'Column144', 'Column146', 'Column148', 'Column150', 'Column152', 'Column154', 'Column156', 'Column158', 'Column160', 'Column162', 'Column164', 'Column166', 'Column168', 'Column170', 'Column172', 'Column174', 'Column176', 'Column178', 'Column180', 'Column182', 'Column184', 'Column186', 'Column188', 'Column190', 'Column192', 'Column194', 'Column196', 'Column198', 'Column200', 'Column202', 'Column204', 'Column206', 'Column208', 'Column210', 'Column212', 'Column214', 'Column216', 'Column218', 'Column220', 'Column222', 'Column224', 'Column226', 'Column228', 'Column230', 'Column232', 'Column234', 'Column236', 'Column238', 'Column240', 'Column242', 'Column244', 'Column246', 'Column248', 'Column250', 'Column252', 'Column254', 'Column256', 'Column258', 'Column260', 'Column262', 'Column264', 'Column266', 'Column268', 'Column270', 'Column272', 'Column274', 'Column276', 'Column278', 'Column280', 'Column282', 'Column284', 'Column286', 'Column288', 'Column290', 'Column292', 'Column294', 'Column296', 'Column298', 'Column300', 'Column302', 'Column304', 'Column306', 'Column308', 'Column310', 'Column312', 'Column314', 'Column316', 'Column318', 'Column320', 'Column322', 'Column324', 'Column326', 'Column328', 'Column330', 'Column332', 'Column334', 'Column336', 'Column338', 'Column340', 'Column342', 'Column344', 'Column346', 'Column348', 'Column350', 'Column352', 'Column354', 'Column356', 'Column358', 'Column360', 'Column362', 'Column364', 'Column366', 'Column368', 'Column370', 'Column372', 'Column374', 'Column376', 'Column378', 'Column380', 'Column382', 'Column384', 'Column386', 'Column388', 'Column390', 'Column392', 'Column394', 'Column396', 'Column398', 'Column400', 'Column402', 'Column404', 'Column406', 'Column408', 'Column410', 'Column412', 'Column414', 'Column416', 'Column418', 'Column420', 'Column422', 'Column424', 'Column426', 'Column428', 'Column430', 'Column432', 'Column434', 'Column436', 'Column438', 'Column440', 'Column442', 'Column444', 'Column446', 'Column448', 'Column450', 'Column452', 'Column454', 'Column456', 'Column458', 'Column460', 'Column462', 'Column464', 'Column466', 'Column468', 'Column470', 'Column472', 'Column474', 'Column476', 'Column478', 'Column480', 'Column482', 'Column484', 'Column486', 'Column488', 'Column490', 'Column492', 'Column494', 'Column496', 'Column498', 'Column500', 'Column502', 'Column504', 'Column506', 'Column508', 'Column510', 'Column512', 'Column514', 'Column516', 'Column518', 'Column520', 'Column522', 'Column524', 'Column526', 'Column528', 'Column530', 'Column532', 'Column534', 'Column536', 'Column538', 'Column540', 'Column542', 'Column544', 'Column546', 'Column548', 'Column550', 'Column552', 'Column554', 'Column556', 'Column558', 'Column560', 'Column562', 'Column564', 'Column566', 'Column568', 'Column570', 'Column572', 'Column574', 'Column576', 'Column578', 'Column580', 'Column582', 'Column584', 'Column586', 'Column588', 'Column590', 'Column592', 'Column594', 'Column596', 'Column598', 'Column600', 'Column602', 'Column604', 'Column606', 'Column608', 'Column610', 'Column612', 'Column614', 'Column616', 'Column618', 'Column620', 'Column622', 'Column624', 'Column626', 'Column628', 'Column630', 'Column632', 'Column634', 'Column636', 'Column638', 'Column640', 'Column642', 'Column644', 'Column646', 'Column648', 'Column650', 'Column652', 'Column654', 'Column656', 'Column658', 'Column660', 'Column662', 'Column664', 'Column666', 'Column668', 'Column670', 'Column672', 'Column674', 'Column676', 'Column678', 'Column680', 'Column682', 'Column684', 'Column686', 'Column688', 'Column690', 'Column692', 'Column694', 'Column696', 'Column698', 'Column700', 'Column702', 'Column704', 'Column706', 'Column708', 'Column710', 'Column712', 'Column714', 'Column716', 'Column718', 'Column720', 'Column722', 'Column724', 'Column726', 'Column728', 'Column730', 'Column732', 'Column734', 'Column736', 'Column738', 'Column740', 'Column742', 'Column744', 'Column746', 'Column748', 'Column750', 'Column752', 'Column754', 'Column756', 'Column758', 'Column760', 'Column762', 'Column764', 'Column766', 'Column768', 'Column770', 'Column772', 'Column774', 'Column776', 'Column778', 'Column780', 'Column782', 'Column784', 'Column786', 'Column788', 'Column790', 'Column792', 'Column794', 'Column796', 'Column798', 'Column800', 'Column802', 'Column804', 'Column806', 'Column808', 'Column810', 'Column812', 'Column814', 'Column816', 'Column818', 'Column820', 'Column822', 'Column824', 'Column826', 'Column828', 'Column830', 'Column832', 'Column834', 'Column836', 'Column838', 'Column840', 'Column842', 'Column844', 'Column846', 'Column848', 'Column850', 'Column852', 'Column854', 'Column856', 'Column858', 'Column860', 'Column862', 'Column864', 'Column866', 'Column868', 'Column870', 'Column872', 'Column874', 'Column876', 'Column878', 'Column880', 'Column882', 'Column884', 'Column886', 'Column888', 'Column890', 'Column892', 'Column894', 'Column896', 'Column898', 'Column900', 'Column902', 'Column904', 'Column906', 'Column908', 'Column910', 'Column912', 'Column914', 'Column916', 'Column918', 'Column920', 'Column922', 'Column924', 'Column926', 'Column928', 'Column930', 'Column932', 'Column934', 'Column936', 'Column938', 'Column940', 'Column942', 'Column944', 'Column946', 'Column948', 'Column950', 'Column952', 'Column954', 'Column956', 'Column958', 'Column960', 'Column962', 'Column964', 'Column966', 'Column968', 'Column970', 'Column972', 'Column974', 'Column976', 'Column978', 'Column980', 'Column982', 'Column984', 'Column986', 'Column988', 'Column990', 'Column992', 'Column994', 'Column996', 'Column998', 'Column1000', 'Column1002', 'Column1004', 'Column1006', 'Column1008', 'Column1010', 'Column1012', 'Column1014', 'Column1016', 'Column1018', 'Column1020', 'Column1022', 'Column1024', 'Column1026', 'Column1028', 'Column1030', 'Column1032', 'Column1034', 'Column1036', 'Column1038', 'Column1040', 'Column1042', 'Column1044', 'Column1046', 'Column1048', 'Column1050', 'Column1052', 'Column1054', 'Column1056', 'Column1058', 'Column1060', 'Column1062', 'Column1064', 'Column1066', 'Column1068', 'Column1070', 'Column1072', 'Column1074', 'Column1076', 'Column1078', 'Column1080', 'Column1082', 'Column1084', 'Column1086', 'Column1088', 'Column1090', 'Column1092', 'Column1094', 'Column1096', 'Column1098', 'Column1100', 'Column1102', 'Column1104', 'Column1106', 'Column1108', 'Column1110', 'Column1112', 'Column1114', 'Column1116', 'Column1118', 'Column1120', 'Column1122', 'Column1124', 'Column1126', 'Column1128', 'Column1130', 'Column1132', 'Column1134', 'Column1136', 'Column1138', 'Column1140', 'Column1142', 'Column1144', 'Column1146', 'Column1148', 'Column1150', 'Column1152', 'Column1154', 'Column1156', 'Column1158', 'Column1160', 'Column1162', 'Column1164', 'Column1166', 'Column1168', 'Column1170', 'Column1172', 'Column1174', 'Column1176', 'Column1178', 'Column1180', 'Column1182', 'Column1184', 'Column1186', 'Column1188', 'Column1190', 'Column1192', 'Column1194', 'Column1196', 'Column1198', 'Column1200', 'Column1202', 'Column1204', 'Column1206', 'Column1208', 'Column1210', 'Column1212', 'Column1214', 'Column1216', 'Column1218', 'Column1220', 'Column1222', 'Column1224', 'Column1226', 'Column1228', 'Column1230', 'Column1232', 'Column1234', 'Column1236', 'Column1238', 'Column1240', 'Column1242', 'Column1244', 'Column1246', 'Column1248', 'Column1250', 'Column1252', 'Column1254', 'Column1256', 'Column1258', 'Column1260', 'Column1262', 'Column1264', 'Column1266', 'Column1268', 'Column1270', 'Column1272', 'Column1274', 'Column1276', 'Column1278', 'Column1280', 'Column1282', 'Column1284', 'Column1286', 'Column1288', 'Column1290', 'Column1292', 'Column1294', 'Column129
```

```

0 0 0 0 0 0 0 1 0 0 0 1 0 0 1 0 0 0 0 0 1 1 0 0 1 0 0 0 0 1 0 1 1 0 0 0 0
0 1 0 0 1 0 0 1 0 1 1 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 1 0 0 0 1
0 0 0 1 1 0 0 0 1 1 1 0 0 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 1 0 0 0 1 0]

```

```

In [5]: classifier = MLPClassifier(max_iter = 500)
        hidden_unit_list = [(1,), (2,), (4,), (8,), (32,), (128,)]
        momentum_list = [0,0.1, 0.2, 0.5, 0.8, 0.9]
        params = {'hidden_layer_sizes':hidden_unit_list,'momentum':momentum_list}
        grid_search = GridSearchCV(classifier, params, return_train_score = True,

```

```

In [6]: def draw_heatmap_RBF(acc, acc_desc, momentum_list, hidden_unit_list):
        plt.figure(figsize = (5,4))
        ax = sns.heatmap(acc, annot=True, fmt='.3f',
                           xticklabels=momentum_list, yticklabels=hidden_unit_list)
        ax.collections[0].colorbar.set_label("accuracy")
        ax.set(xlabel = '$momentum$', ylabel='$hidden_unit_list$')
        plt.title(acc_desc + ' w.r.t $hidden_units$ and $momentum$')
        sns.set_style("whitegrid", {'axes.grid' : False})
        plt.show()

```

```

In [7]: Y = Y.astype('int')

        X_train_val = X[:int(0.8*len(X))] # Get features from train + val set.
        X_test       = X[int(0.8*len(X)):] # Get features from test set.
        Y_train_val = Y[:int(0.8*len(Y))] # Get labels from train + val set.
        Y_test       = Y[int(0.8*len(Y)):] # Get labels from test set.
        grid_search.fit(X_train_val, Y_train_val)

```

```

Out[7]: GridSearchCV(cv=5, error_score='raise',
                    estimator=MLPClassifier(activation='relu', alpha=0.0001, batch_size=
                    beta_2=0.999, early_stopping=False, epsilon=1e-08,
                    hidden_layer_sizes=(100,), learning_rate='constant',
                    learning_rate_init=0.001, max_iter=500, momentum=0.9,
                    nesterovs_momentum=True, power_t=0.5, random_state=None,
                    shuffle=True, solver='adam', tol=0.0001, validation_fraction=0.1,
                    verbose=False, warm_start=False),
                    fit_params={}, iid=True, n_jobs=1,
                    param_grid={'hidden_layer_sizes': [(1,), (2,), (4,), (8,), (32,), (1
                    pre_dispatch='2*n_jobs', refit=True, return_train_score=True,
                    scoring=None, verbose=0)

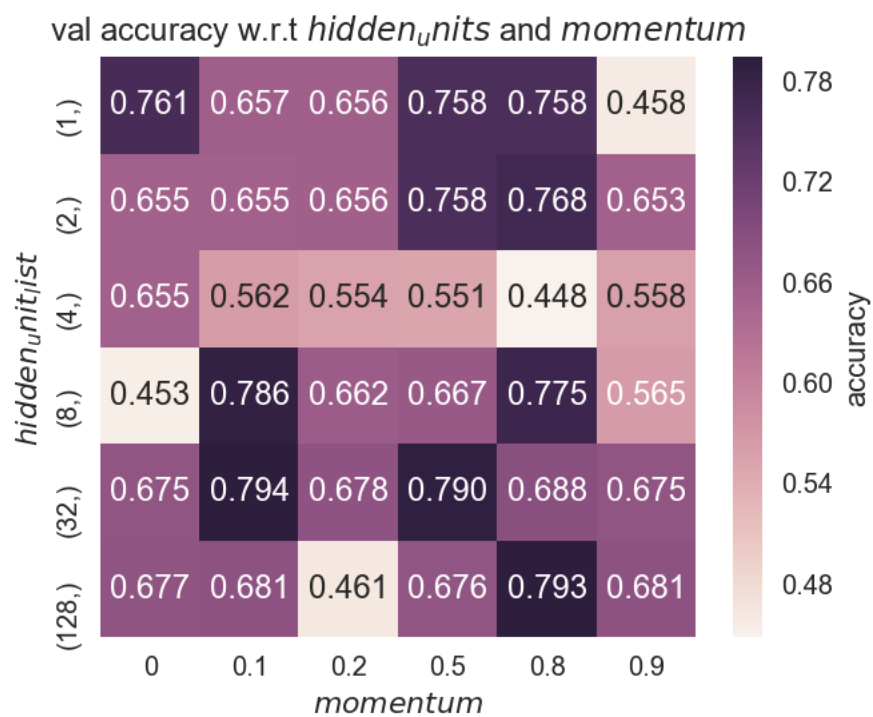
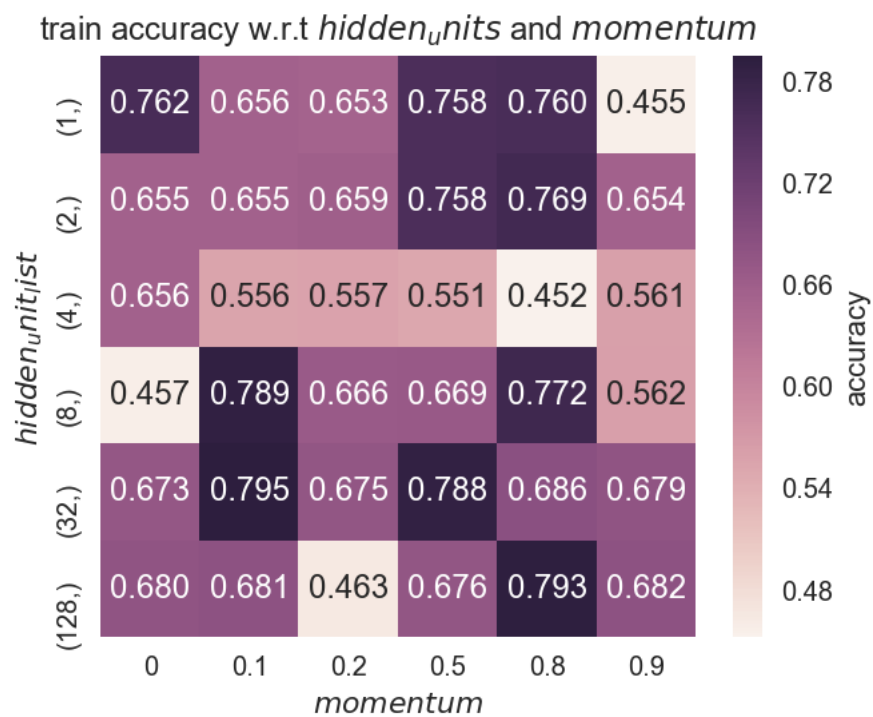
```

```

In [8]: train_acc = grid_search.cv_results_['mean_train_score'].reshape(6,6)
        draw_heatmap_RBF(train_acc, 'train accuracy', momentum_list, hidden_unit_list)

        val_acc = grid_search.cv_results_['mean_test_score'].reshape(6,6)
        draw_heatmap_RBF(val_acc, 'val accuracy', momentum_list, hidden_unit_list)
        print(train_acc.shape, val_acc.shape)

```



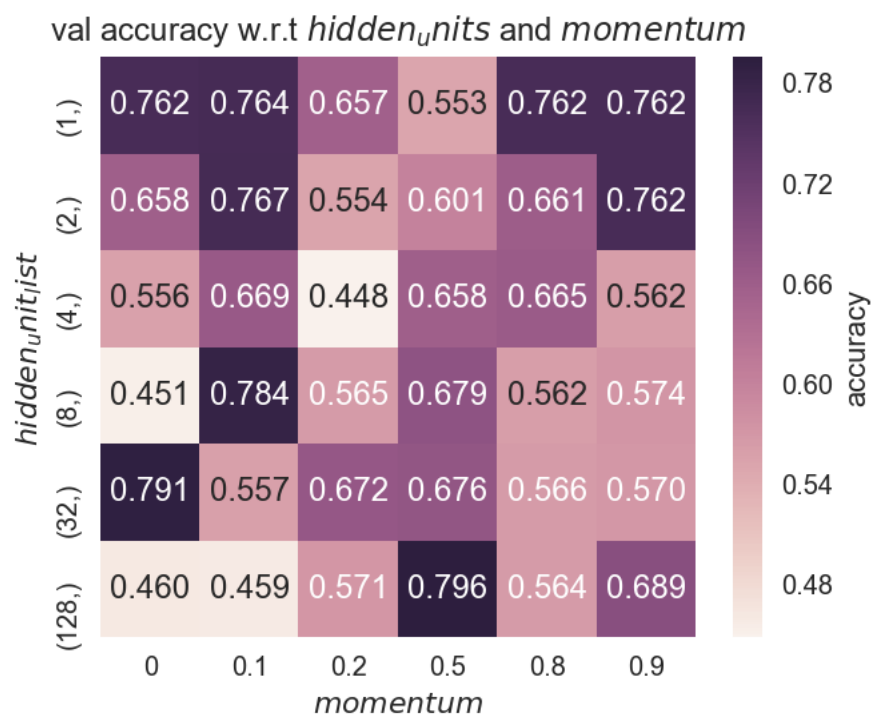
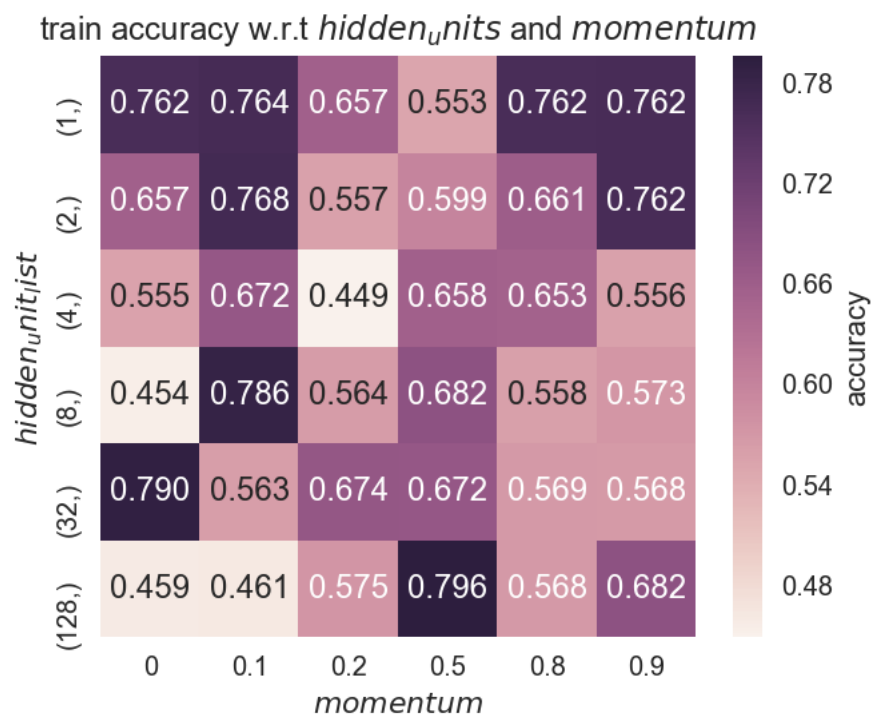
(6, 6) (6, 6)

```
In [9]: test_acc = sum(grid_search.best_estimator_.predict(X_test) == Y_test) / len(X_test)
print(grid_search.best_params_)
print(test_acc)
tot_train = (sum(train_acc)/len(train_acc))
tot_train = (sum(tot_train)/len(tot_train))
print(tot_train)
tot_val = (sum(val_acc)/len(val_acc))
tot_val = (sum(tot_val)/len(tot_val))
print(tot_val)
```

```
{'hidden_layer_sizes': (32,), 'momentum': 0.1}
0.778
0.658777141028
0.658395833333
```

```
In [10]: X_train_val = X[:int(0.5*len(X))] # Get features from train + val set.
X_test = X[int(0.5*len(X)):] # Get features from test set.
Y_train_val = Y[:int(0.5*len(Y))] # Get labels from train + val set.
Y_test = Y[int(0.5*len(Y)):] # Get labels from test set.
grid_search.fit(X_train_val, Y_train_val)
train_acc2 = grid_search.cv_results_['mean_train_score'].reshape(6,6)
draw_heatmap_RBF(train_acc2, 'train accuracy', momentum_list, hidden_unit_list)

val_acc2 = grid_search.cv_results_['mean_test_score'].reshape(6,6)
draw_heatmap_RBF(val_acc2, 'val accuracy', momentum_list, hidden_unit_list)
```



```

In [11]: test_acc2 = sum(grid_search.best_estimator_.predict(X_test) == Y_test) / len(Y_test)
print(grid_search.best_params_)
print(test_acc2)
tot_train2 = (sum(train_acc2))/len(train_acc2)
tot_train2 = (sum(tot_train2))/len(tot_train2)
print(tot_train2)
tot_val2 = (sum(val_acc2))/len(val_acc2)
tot_val2 = (sum(tot_val2))/len(tot_val2)
print(tot_val2)

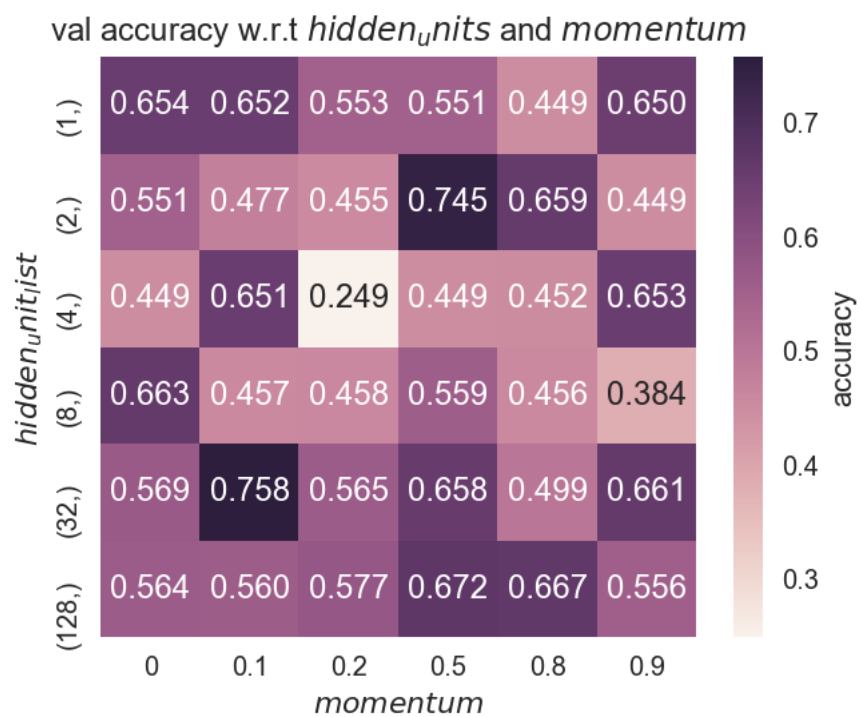
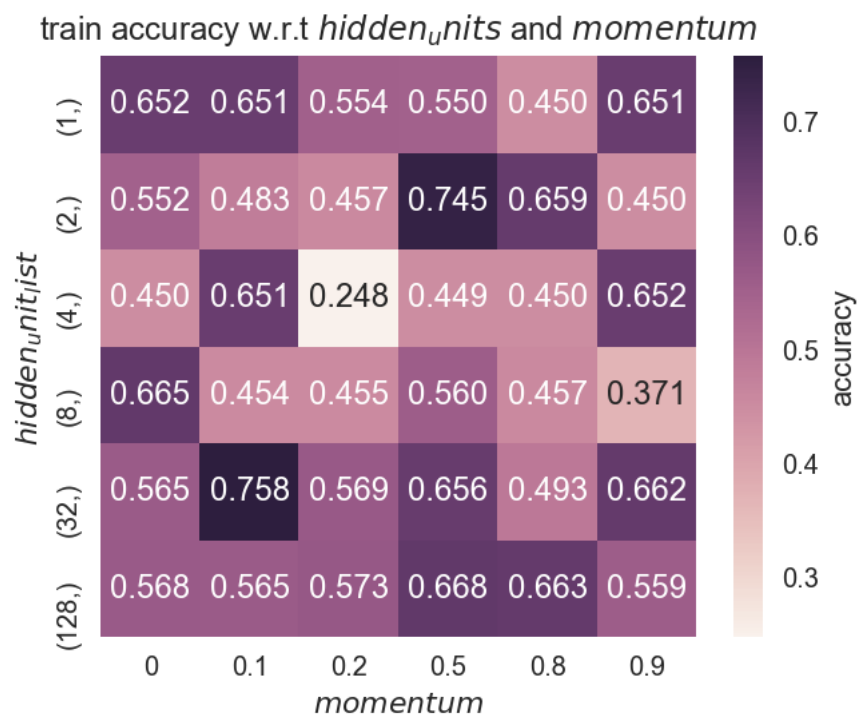
{'hidden_layer_sizes': (128,), 'momentum': 0.5}
0.784
0.633383333333
0.6335

In [12]: X_train_val = X[:int(0.2*len(X))] # Get features from train + val set.
X_test = X[int(0.2*len(X)):] # Get features from test set.
Y_train_val = Y[:int(0.2*len(Y))] # Get labels from train + val set.
Y_test = Y[int(0.2*len(Y)):] # Get labels from test set.
grid_search.fit(X_train_val, Y_train_val)
train_acc3 = grid_search.cv_results_['mean_train_score'].reshape(6,6)
draw_heatmap_RBF(train_acc3, 'train accuracy', momentum_list, hidden_unit_list)

val_acc3 = grid_search.cv_results_['mean_test_score'].reshape(6,6)
draw_heatmap_RBF(val_acc3, 'val accuracy', momentum_list, hidden_unit_list)

C:\ProgramData\Anaconda3\lib\site-packages\sklearn\neural_network\multilayer_perceptron.py:178:
ConvergenceWarning: Stochastic Optimizer: Maximum iterations (1000) reached. Optimization terminated.
% (), ConvergenceWarning)
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\neural_network\multilayer_perceptron.py:178:
ConvergenceWarning: Stochastic Optimizer: Maximum iterations (1000) reached. Optimization terminated.
% (), ConvergenceWarning)
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\neural_network\multilayer_perceptron.py:178:
ConvergenceWarning: Stochastic Optimizer: Maximum iterations (1000) reached. Optimization terminated.
% (), ConvergenceWarning)
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\neural_network\multilayer_perceptron.py:178:
ConvergenceWarning: Stochastic Optimizer: Maximum iterations (1000) reached. Optimization terminated.
% (), ConvergenceWarning)
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\neural_network\multilayer_perceptron.py:178:
ConvergenceWarning: Stochastic Optimizer: Maximum iterations (1000) reached. Optimization terminated.
% (), ConvergenceWarning)
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\neural_network\multilayer_perceptron.py:178:
ConvergenceWarning: Stochastic Optimizer: Maximum iterations (1000) reached. Optimization terminated.
% (), ConvergenceWarning)
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\neural_network\multilayer_perceptron.py:178:
ConvergenceWarning: Stochastic Optimizer: Maximum iterations (1000) reached. Optimization terminated.
% (), ConvergenceWarning)
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\neural_network\multilayer_perceptron.py:178:
ConvergenceWarning: Stochastic Optimizer: Maximum iterations (1000) reached. Optimization terminated.
% (), ConvergenceWarning)
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\neural_network\multilayer_perceptron.py:178:
ConvergenceWarning: Stochastic Optimizer: Maximum iterations (1000) reached. Optimization terminated.
% (), ConvergenceWarning)

```



```

In [13]: test_acc3 = sum(grid_search.best_estimator_.predict(X_test) == Y_test) / len(Y_test)
print(grid_search.best_params_)
print(test_acc3)
tot_train3 = (sum(train_acc3))/len(train_acc3)
tot_train3 = (sum(tot_train3))/len(tot_train3)
print(tot_train3)
tot_val3 = (sum(val_acc3))/len(val_acc3)
tot_val3 = (sum(tot_val3))/len(tot_val3)
print(tot_val3)

```

```

{'hidden_layer_sizes': (32,), 'momentum': 0.1}
0.24025
0.555919453943
0.556416666667

```

```

In [14]: avg_test = (test_acc + test_acc2 + test_acc3)/3
avg_train = (tot_train + tot_train2 + tot_train3)/3
avg_val = (tot_val + tot_val2 + tot_val3)/3
print(avg_test, avg_train, avg_val)

```

```

0.60075 0.616026642768 0.616104166667

```

```

In [ ]:

```



# ANNCov

March 24, 2018

```
In [1]: import scipy.io as sio
import matplotlib.pyplot as plt
import numpy as np
import seaborn as sns
import pandas as pd
from sklearn.neural_network import MLPClassifier
from sklearn.model_selection import GridSearchCV
%config InlineBackend.figure_format = 'retina'

In [2]: df = pd.read_csv('C:/Users/Preston Wong/Downloads/covtype.data.gz', compress
X_and_Y = df.as_matrix()
X_and_Y = X_and_Y[:5000, :55]
X = X_and_Y[:, 0:-1]
Y = X_and_Y[:, -1]
print(X_and_Y.shape, X.shape, Y.shape)

(5000, 55) (5000, 54) (5000,)

In [3]: #change to binary classification
for i in range(len(Y)):
    if Y[i] == 2:
        Y[i] = 1
    else:
        Y[i] = 0
np.random.shuffle(X_and_Y)

In [4]: classifier = MLPClassifier(max_iter = 500)
hidden_unit_list = [(1,), (2,), (4,), (8,), (32,), (128,)]
momentum_list = [0, 0.1, 0.2, 0.5, 0.8, 0.9]
params = {'hidden_layer_sizes':hidden_unit_list, 'momentum':momentum_list}
grid_search = GridSearchCV(classifier, params, return_train_score = True, c

In [5]: def draw_heatmap_RBF(acc, acc_desc, momentum_list, hidden_unit_list):
plt.figure(figsize = (5,4))
ax = sns.heatmap(acc, annot=True, fmt='.3f',
                  xticklabels=momentum_list, yticklabels=hidden_unit_list)
ax.collections[0].colorbar.set_label("accuracy")
```

```

ax.set(xlabel = '$momentum$', ylabel='$hidden_unit_list$')
plt.title(acc_desc + ' w.r.t $hidden_units$ and $momentum$')
sns.set_style("whitegrid", {'axes.grid' : False})
plt.show()

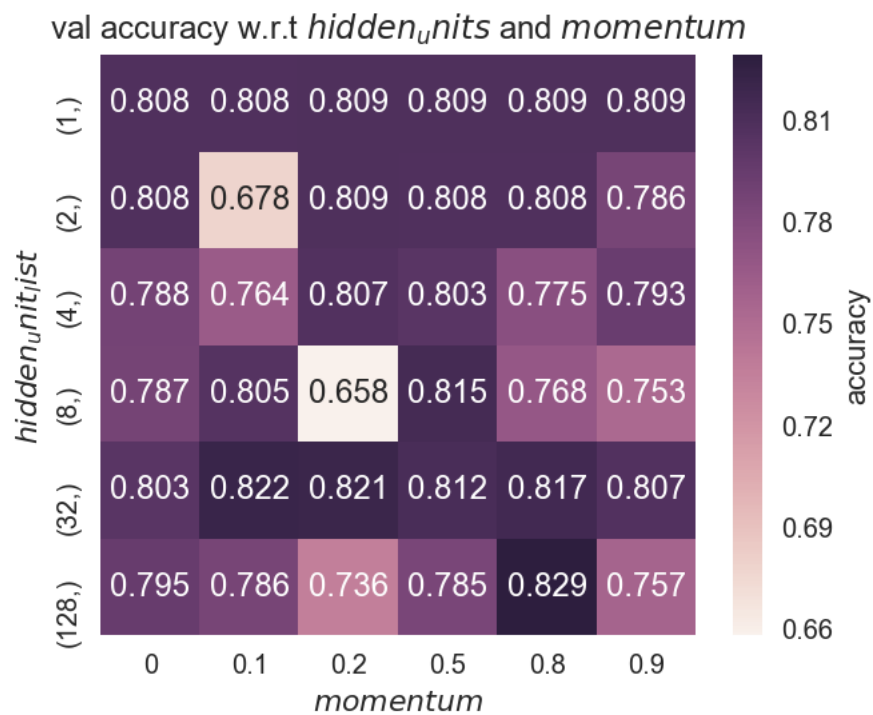
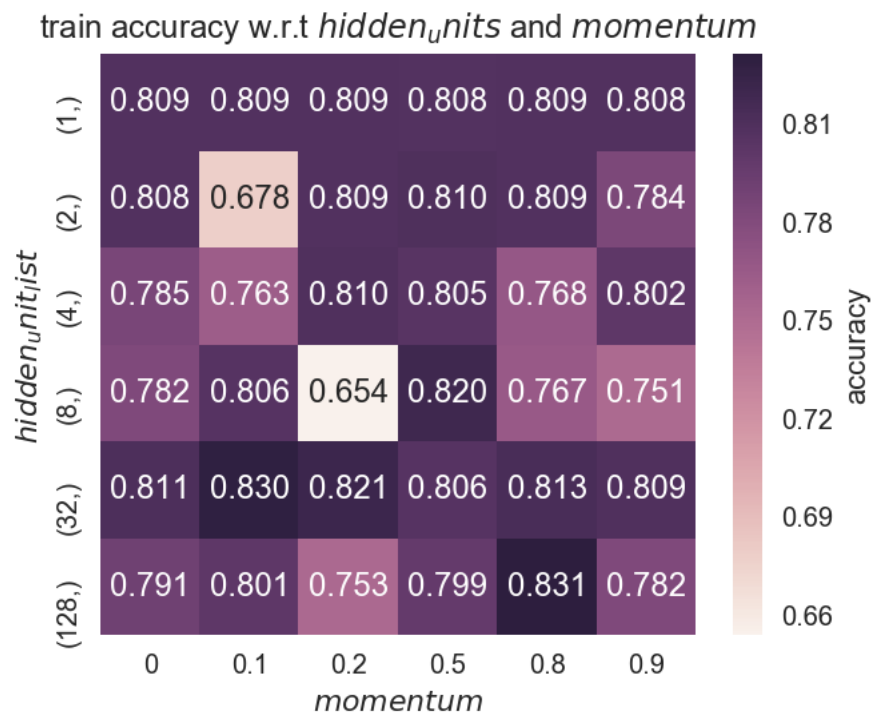
In [6]: X_train_val = X[:int(0.8*len(X))] # Get features from train + val set.
X_test = X[int(0.8*len(X)):] # Get features from test set.
Y_train_val = Y[:int(0.8*len(Y))] # Get labels from train + val set.
Y_test = Y[int(0.8*len(Y)):] # Get labels from test set.
grid_search.fit(X_train_val, Y_train_val)

Out[6]: GridSearchCV(cv=5, error_score='raise',
                    estimator=MLPClassifier(activation='relu', alpha=0.0001, batch_size=
                    beta_2=0.999, early_stopping=False, epsilon=1e-08,
                    hidden_layer_sizes=(100,), learning_rate='constant',
                    learning_rate_init=0.001, max_iter=500, momentum=0.9,
                    nesterovs_momentum=True, power_t=0.5, random_state=None,
                    shuffle=True, solver='adam', tol=0.0001, validation_fraction=0.1,
                    verbose=False, warm_start=False),
                    fit_params={}, iid=True, n_jobs=1,
                    param_grid={'hidden_layer_sizes': [(1,), (2,), (4,), (8,), (32,), (1
                    pre_dispatch='2*n_jobs', refit=True, return_train_score=True,
                    scoring=None, verbose=0)

In [7]: train_acc = grid_search.cv_results_['mean_train_score'].reshape(6,6)
draw_heatmap_RBF(train_acc, 'train accuracy', momentum_list, hidden_unit_list)

val_acc = grid_search.cv_results_['mean_test_score'].reshape(6,6)
draw_heatmap_RBF(val_acc, 'val accuracy', momentum_list, hidden_unit_list)
print(train_acc.shape, val_acc.shape)

```



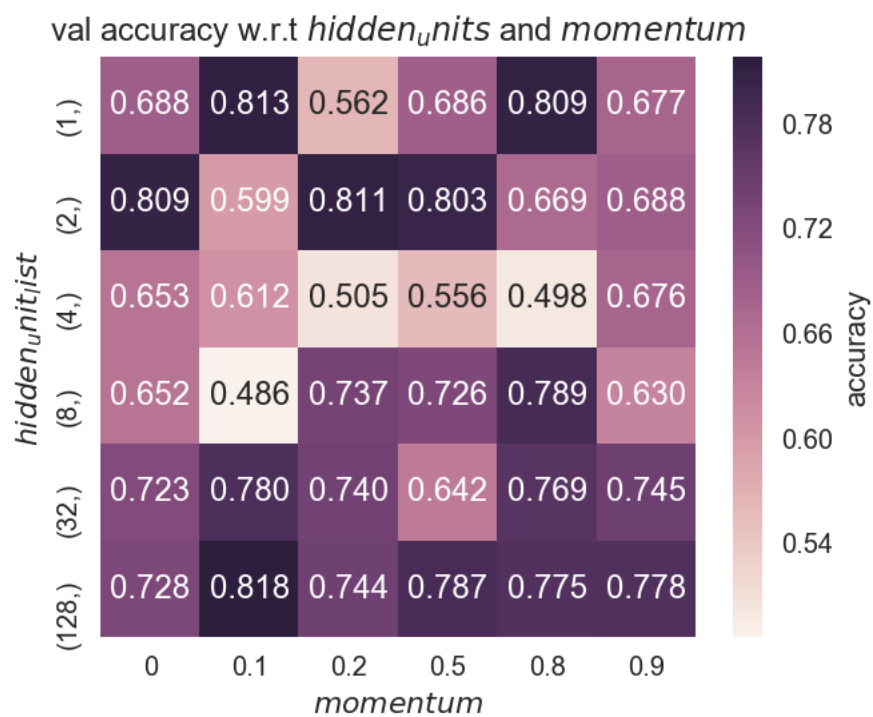
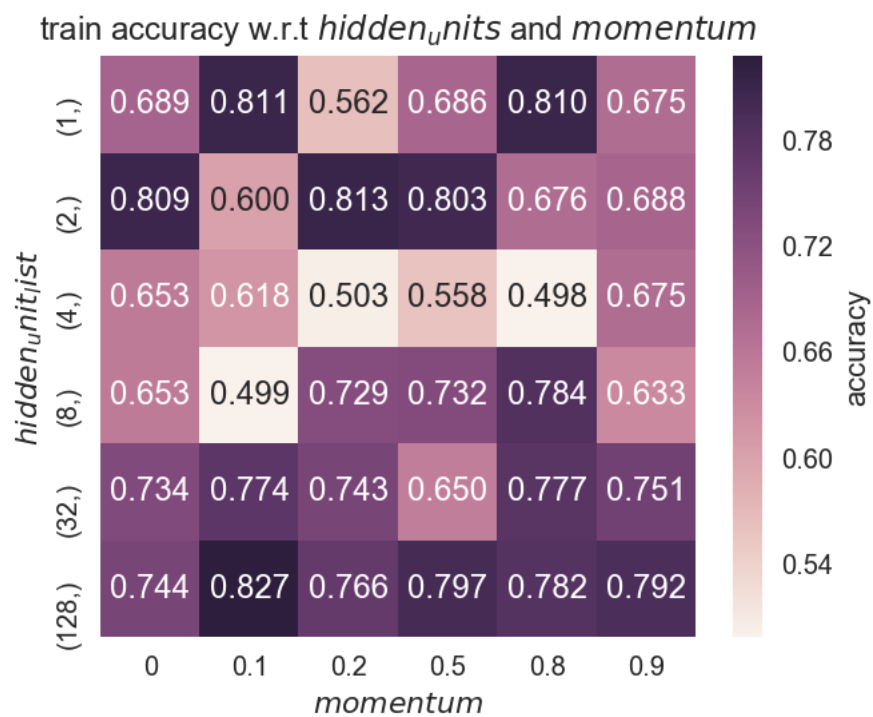
(6, 6) (6, 6)

```
In [8]: test_acc = sum(grid_search.best_estimator_.predict(X_test) == Y_test) / len(X_test)
print(grid_search.best_params_)
print(test_acc)
tot_train = (sum(train_acc)/len(train_acc))
tot_train = (sum(tot_train)/len(tot_train))
print(tot_train)
tot_val = (sum(val_acc)/len(val_acc))
tot_val = (sum(tot_val)/len(tot_val))
print(tot_val)
```

```
{'hidden_layer_sizes': (128,), 'momentum': 0.8}
0.843
0.791872336509
0.789923611111
```

```
In [9]: X_train_val = X[:int(0.5*len(X))] # Get features from train + val set.
X_test = X[int(0.5*len(X)):] # Get features from test set.
Y_train_val = Y[:int(0.5*len(Y))] # Get labels from train + val set.
Y_test = Y[int(0.5*len(Y)):] # Get labels from test set.
grid_search.fit(X_train_val, Y_train_val)
train_acc2 = grid_search.cv_results_['mean_train_score'].reshape(6,6)
draw_heatmap_RBF(train_acc2, 'train accuracy', momentum_list, hidden_unit_list)

val_acc2 = grid_search.cv_results_['mean_test_score'].reshape(6,6)
draw_heatmap_RBF(val_acc2, 'val accuracy', momentum_list, hidden_unit_list)
```



```
In [10]: test_acc2 = sum(grid_search.best_estimator_.predict(X_test) == Y_test) / len(Y_test)
print(grid_search.best_params_)
print(test_acc2)
tot_train2 = (sum(train_acc2))/len(train_acc2)
tot_train2 = (sum(tot_train2))/len(tot_train2)
print(tot_train2)
tot_val2 = (sum(val_acc2))/len(val_acc2)
tot_val2 = (sum(tot_val2))/len(tot_val2)
print(tot_val2)
```

```
{'hidden_layer_sizes': (128,), 'momentum': 0.1}
0.8184
0.702573032145
0.699
```

```
In [11]: X_train_val = X[:int(0.2*len(X))] # Get features from train + val set.
X_test = X[int(0.2*len(X)):] # Get features from test set.
Y_train_val = Y[:int(0.2*len(Y))] # Get labels from train + val set.
Y_test = Y[int(0.2*len(Y)):] # Get labels from test set.
grid_search.fit(X_train_val, Y_train_val)
train_acc3 = grid_search.cv_results_['mean_train_score'].reshape(6,6)
draw_heatmap_RBF(train_acc3, 'train accuracy', momentum_list, hidden_unit_list)

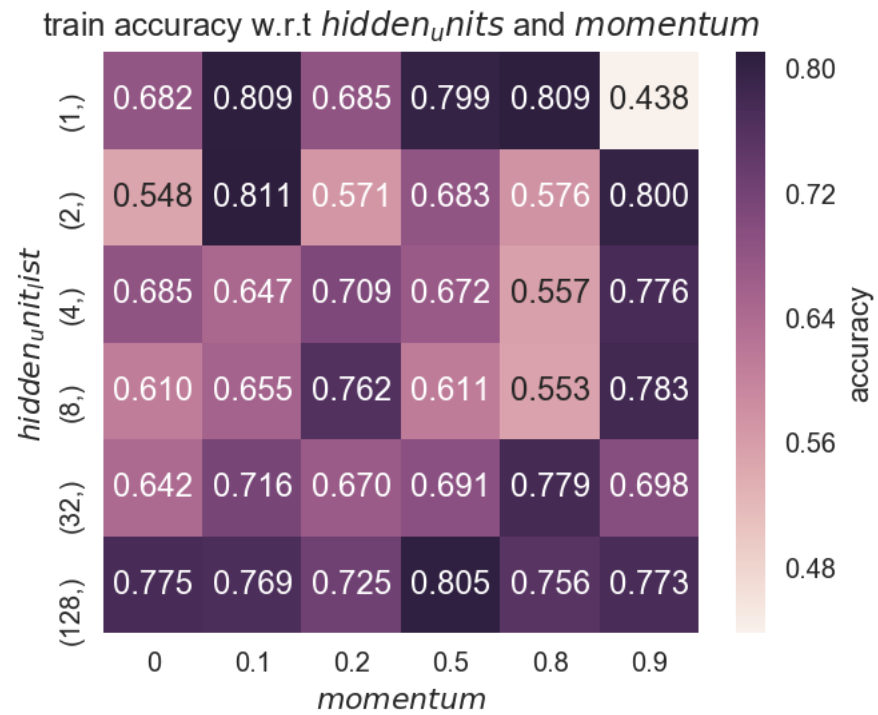
val_acc3 = grid_search.cv_results_['mean_test_score'].reshape(6,6)
draw_heatmap_RBF(val_acc3, 'val accuracy', momentum_list, hidden_unit_list)
```

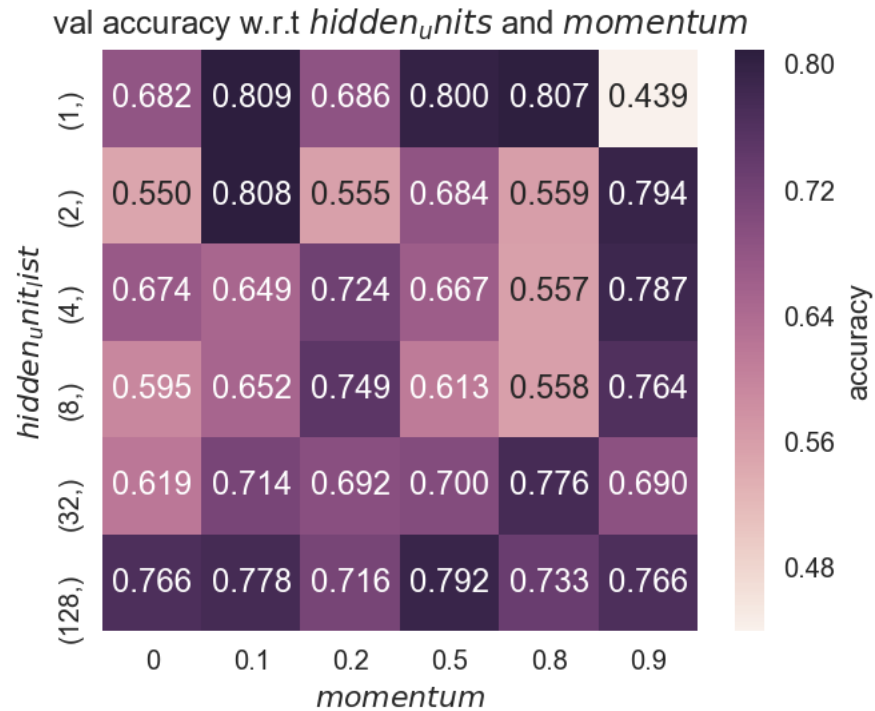
```
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\neural_network\multilayer_perceptron.py:577: ConvergenceWarning:
% (), ConvergenceWarning)
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\neural_network\multilayer_perceptron.py:577: ConvergenceWarning:
% (), ConvergenceWarning)
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\neural_network\multilayer_perceptron.py:577: ConvergenceWarning:
% (), ConvergenceWarning)
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\neural_network\multilayer_perceptron.py:577: ConvergenceWarning:
% (), ConvergenceWarning)
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\neural_network\multilayer_perceptron.py:577: ConvergenceWarning:
% (), ConvergenceWarning)
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\neural_network\multilayer_perceptron.py:577: ConvergenceWarning:
% (), ConvergenceWarning)
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\neural_network\multilayer_perceptron.py:577: ConvergenceWarning:
% (), ConvergenceWarning)
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\neural_network\multilayer_perceptron.py:577: ConvergenceWarning:
% (), ConvergenceWarning)
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\neural_network\multilayer_perceptron.py:577: ConvergenceWarning:
% (), ConvergenceWarning)
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\neural_network\multilayer_perceptron.py:577: ConvergenceWarning:
% (), ConvergenceWarning)
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\neural_network\multilayer_perceptron.py:577: ConvergenceWarning:
% (), ConvergenceWarning)
```

```

% (), ConvergenceWarning)
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\neural_network\multilayer_perceptron.py:577: ConvergenceWarning:
% (), ConvergenceWarning)
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\neural_network\multilayer_perceptron.py:577: ConvergenceWarning)

```





```
In [12]: test_acc3 = sum(grid_search.best_estimator_.predict(X_test) == Y_test) / len(Y_test)
print(grid_search.best_params_)
print(test_acc3)
tot_train3 = (sum(train_acc3)) / len(train_acc3)
tot_train3 = (sum(tot_train3) / len(tot_train3))
print(tot_train3)
tot_val3 = (sum(val_acc3) / len(val_acc3))
tot_val3 = (sum(tot_val3) / len(tot_val3))
print(tot_val3)
```

```
{'hidden_layer_sizes': (1,), 'momentum': 0.1}
0.81075
0.695223363023
0.691777777778
```

```
In [13]: avg_test = (test_acc + test_acc2 + test_acc3) / 3
avg_train = (tot_train + tot_train2 + tot_train3) / 3
avg_val = (tot_val + tot_val2 + tot_val3) / 3
print(avg_test, avg_train, avg_val)
```

```
0.82405 0.729889577226 0.726900462963
```

```
In [ ]:
```



# ANNLETTERS

March 24, 2018

```
In [1]: import scipy.io as sio
import matplotlib.pyplot as plt
import numpy as np
import seaborn as sns
import pandas as pd
from sklearn.neural_network import MLPClassifier
from sklearn.model_selection import GridSearchCV
%config InlineBackend.figure_format = 'retina'

In [2]: df = pd.read_csv('C:/Users/Preston Wong/Downloads/letter-recognition.data',
df2 = df.copy()
df = df.drop([0], axis=1)
df = df.join(df2[0])

In [3]: X_and_Y = df.as_matrix()
np.random.shuffle(X_and_Y)
X_and_Y = X_and_Y[:5000, :17]
X = X_and_Y[:, 0:-1]
Y = X_and_Y[:, -1]
print(X_and_Y.shape, X.shape, Y.shape)

(5000, 17) (5000, 16) (5000,)

In [4]: ordlist = []
for i in range(len(Y)):
    ordlist.append(ord(Y[i]))

In [5]: for i in range(len(Y)):
    if ((ordlist[i]) >= 65 and (ordlist[i] <= 77)):
        Y[i] = 1
    else:
        Y[i] = 0

In [6]: classifier = MLPClassifier(max_iter = 500)
hidden_unit_list = [(1,), (2,), (4,), (8,), (32,), (128,)]
momentum_list = [0,0.1, 0.2, 0.5, 0.8, 0.9]
params = {'hidden_layer_sizes':hidden_unit_list, 'momentum':momentum_list}
grid_search = GridSearchCV(classifier, params, return_train_score = True, c
```

```

In [7]: def draw_heatmap_RBF(acc, acc_desc, momentum_list, hidden_unit_list):
plt.figure(figsize = (5,4))
ax = sns.heatmap(acc, annot=True, fmt='.3f',
                  xticklabels=momentum_list, yticklabels=hidden_unit_list)
ax.collections[0].colorbar.set_label("accuracy")
ax.set(xlabel = '$momentum$', ylabel='$hidden_unit_list$')
plt.title(acc_desc + ' w.r.t $hidden_units$ and $momentum$')
sns.set_style("whitegrid", {'axes.grid' : False})
plt.show()

In [8]: Y = Y.astype('int')

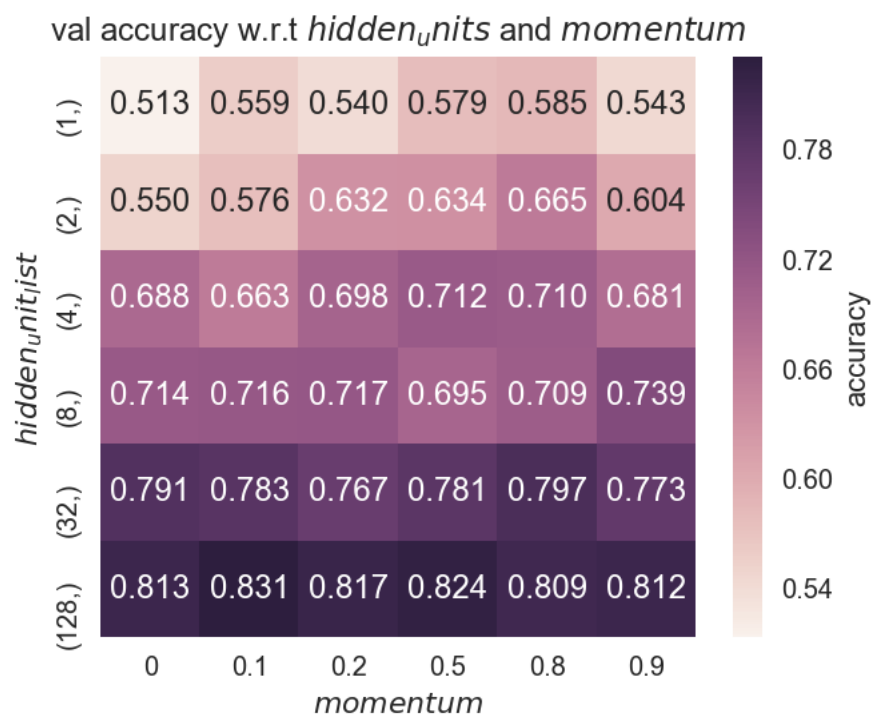
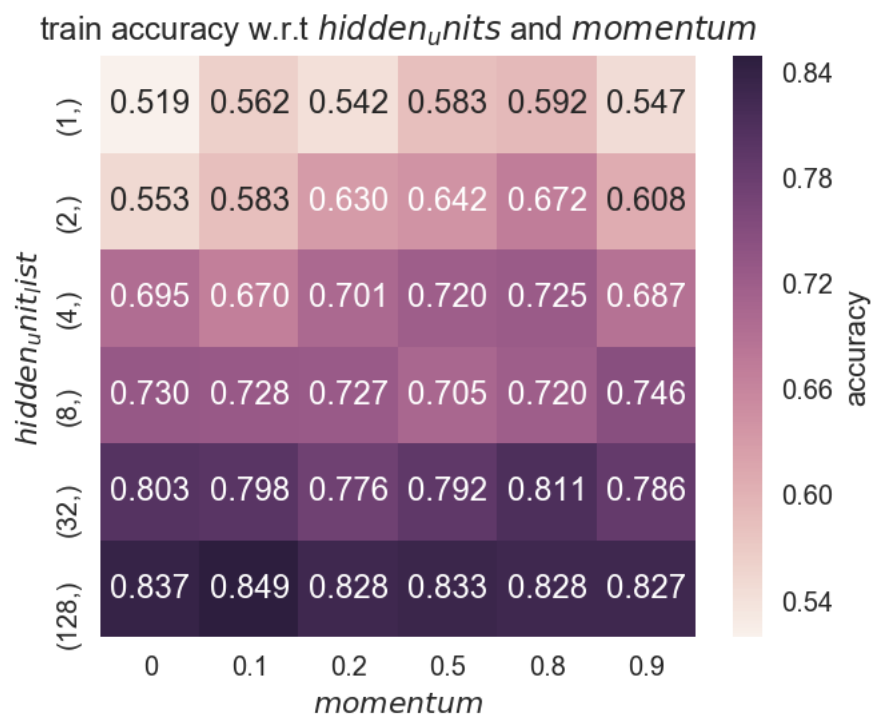
X_train_val = X[:int(0.8*len(X))] # Get features from train + val set.
X_test       = X[int(0.8*len(X)):] # Get features from test set.
Y_train_val = Y[:int(0.8*len(Y))] # Get labels from train + val set.
Y_test       = Y[int(0.8*len(Y)):] # Get labels from test set.
grid_search.fit(X_train_val, Y_train_val)

Out[8]: GridSearchCV(cv=5, error_score='raise',
                    estimator=MLPClassifier(activation='relu', alpha=0.0001, batch_size=
                    beta_2=0.999, early_stopping=False, epsilon=1e-08,
                    hidden_layer_sizes=(100,), learning_rate='constant',
                    learning_rate_init=0.001, max_iter=500, momentum=0.9,
                    nesterovs_momentum=True, power_t=0.5, random_state=None,
                    shuffle=True, solver='adam', tol=0.0001, validation_fraction=0.1,
                    verbose=False, warm_start=False),
                    fit_params={}, iid=True, n_jobs=1,
                    param_grid={'hidden_layer_sizes': [(1,), (2,), (4,), (8,), (32,), (
                    pre_dispatch='2*n_jobs', refit=True, return_train_score=True,
                    scoring=None, verbose=0)

In [9]: train_acc = grid_search.cv_results_['mean_train_score'].reshape(6,6)
draw_heatmap_RBF(train_acc, 'train accuracy', momentum_list, hidden_unit_list)

val_acc = grid_search.cv_results_['mean_test_score'].reshape(6,6)
draw_heatmap_RBF(val_acc, 'val accuracy', momentum_list, hidden_unit_list)
print(train_acc.shape, val_acc.shape)

```



(6, 6) (6, 6)

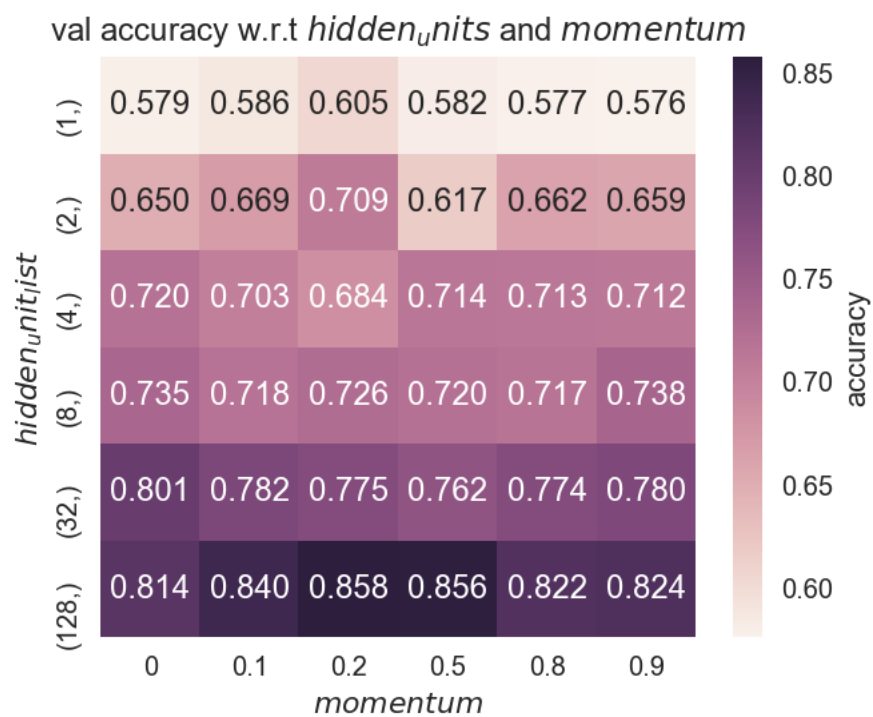
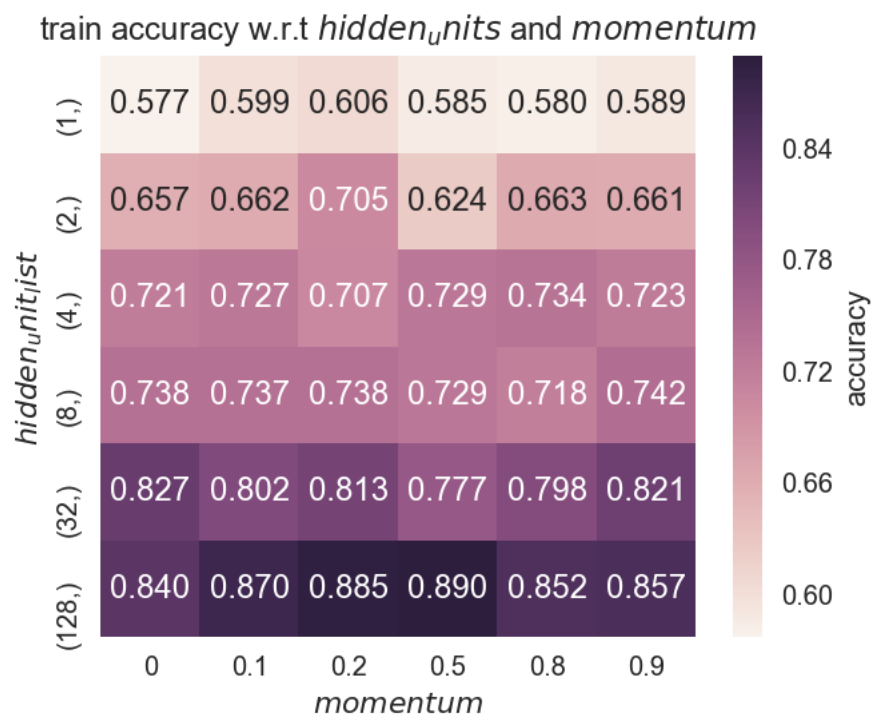
```
In [10]: test_acc = sum(grid_search.best_estimator_.predict(X_test) == Y_test) / len(X_test)
print(grid_search.best_params_)
print(test_acc)
tot_train = (sum(train_acc))/len(train_acc)
tot_train = (sum(tot_train)/len(tot_train))
print(tot_train)
tot_val = (sum(val_acc)/len(val_acc))
tot_val = (sum(tot_val)/len(tot_val))
print(tot_val)
```

```
{'hidden_layer_sizes': (128,), 'momentum': 0.1}
0.88
0.704265253377
0.694979166667
```

```
In [11]: X_train_val = X[:int(0.5*len(X))] # Get features from train + val set.
X_test = X[int(0.5*len(X)):] # Get features from test set.
Y_train_val = Y[:int(0.5*len(Y))] # Get labels from train + val set.
Y_test = Y[int(0.5*len(Y)):] # Get labels from test set.
grid_search.fit(X_train_val, Y_train_val)
train_acc2 = grid_search.cv_results_['mean_train_score'].reshape(6,6)
draw_heatmap_RBF(train_acc2, 'train accuracy', momentum_list, hidden_unit_list)

val_acc2 = grid_search.cv_results_['mean_test_score'].reshape(6,6)
draw_heatmap_RBF(val_acc2, 'val accuracy', momentum_list, hidden_unit_list)
```

```
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\neural_network\multilayer_perceptron.py:577: ConvergenceWarning:
% (), ConvergenceWarning)
```



```

In [12]: test_acc2 = sum(grid_search.best_estimator_.predict(X_test) == Y_test) / len(Y_test)
print(grid_search.best_params_)
print(test_acc2)
tot_train2 = (sum(train_acc2))/len(train_acc2)
tot_train2 = (sum(tot_train2))/len(tot_train2)
print(tot_train2)
tot_val2 = (sum(val_acc2))/len(val_acc2)
tot_val2 = (sum(tot_val2))/len(tot_val2)
print(tot_val2)

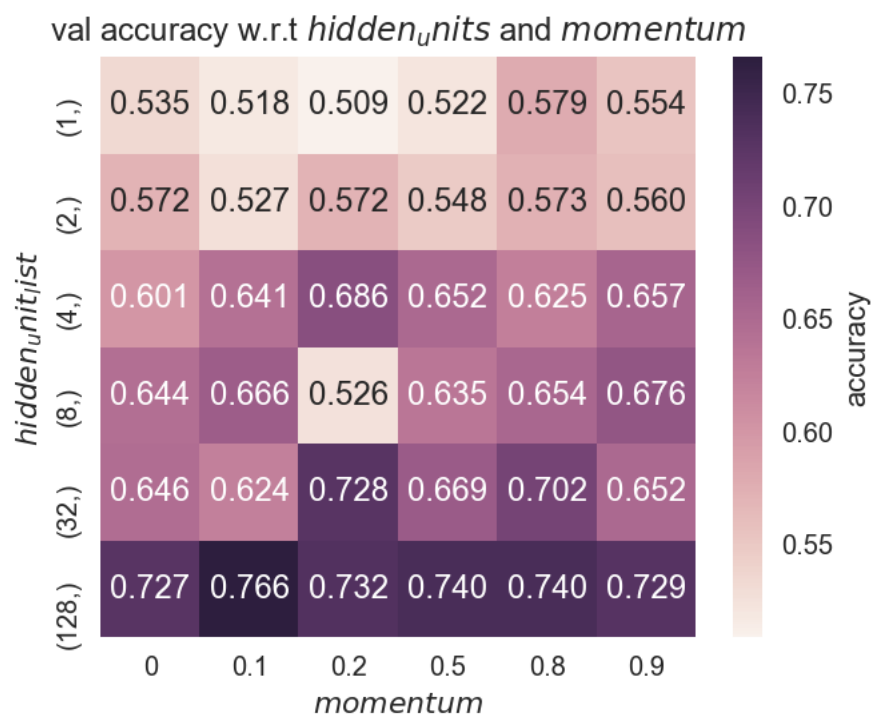
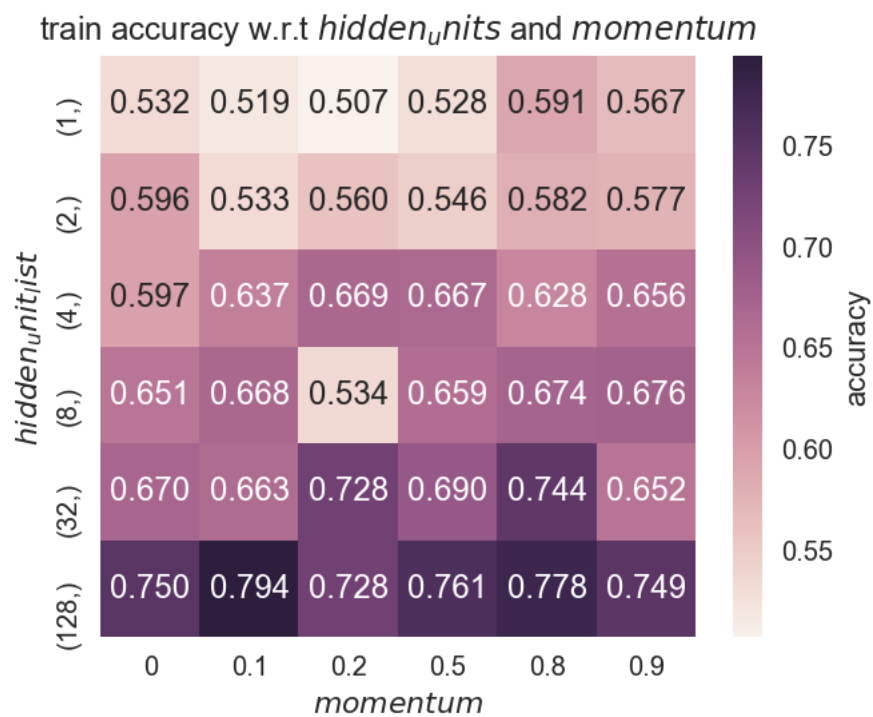
{'hidden_layer_sizes': (128,), 'momentum': 0.2}
0.8572
0.730083333333
0.715588888889

In [13]: X_train_val = X[:int(0.2*len(X))] # Get features from train + val set.
X_test = X[int(0.2*len(X)):] # Get features from test set.
Y_train_val = Y[:int(0.2*len(Y))] # Get labels from train + val set.
Y_test = Y[int(0.2*len(Y)):] # Get labels from test set.
grid_search.fit(X_train_val, Y_train_val)
train_acc3 = grid_search.cv_results_['mean_train_score'].reshape(6,6)
draw_heatmap_RBF(train_acc3, 'train accuracy', momentum_list, hidden_unit_list)

val_acc3 = grid_search.cv_results_['mean_test_score'].reshape(6,6)
draw_heatmap_RBF(val_acc3, 'val accuracy', momentum_list, hidden_unit_list)

C:\ProgramData\Anaconda3\lib\site-packages\sklearn\neural_network\multilayer_perceptron.py:577: ConvergenceWarning:
% (), ConvergenceWarning)
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\neural_network\multilayer_perceptron.py:577: ConvergenceWarning:
% (), ConvergenceWarning)
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\neural_network\multilayer_perceptron.py:577: ConvergenceWarning:
% (), ConvergenceWarning)
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\neural_network\multilayer_perceptron.py:577: ConvergenceWarning:
% (), ConvergenceWarning)
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\neural_network\multilayer_perceptron.py:577: ConvergenceWarning:
% (), ConvergenceWarning)
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\neural_network\multilayer_perceptron.py:577: ConvergenceWarning:
% (), ConvergenceWarning)
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\neural_network\multilayer_perceptron.py:577: ConvergenceWarning:
% (), ConvergenceWarning)
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\neural_network\multilayer_perceptron.py:577: ConvergenceWarning:
% (), ConvergenceWarning)
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\neural_network\multilayer_perceptron.py:577: ConvergenceWarning:
% (), ConvergenceWarning)
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\neural_network\multilayer_perceptron.py:577: ConvergenceWarning:
% (), ConvergenceWarning)
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\neural_network\multilayer_perceptron.py:577: ConvergenceWarning:
% (), ConvergenceWarning)

```



```

In [14]: test_acc3 = sum(grid_search.best_estimator_.predict(X_test) == Y_test) / len(Y_test)
print(grid_search.best_params_)
print(test_acc3)
tot_train3 = (sum(train_acc3))/len(train_acc3)
tot_train3 = (sum(tot_train3))/len(tot_train3)
print(tot_train3)
tot_val3 = (sum(val_acc3))/len(val_acc3)
tot_val3 = (sum(tot_val3))/len(tot_val3)
print(tot_val3)

```

```

{'hidden_layer_sizes': (128,), 'momentum': 0.1}
0.785
0.640501815953
0.630194444444

```

```

In [15]: avg_test = (test_acc + test_acc2 + test_acc3)/3
avg_train = (tot_train + tot_train2 + tot_train3)/3
avg_val = (tot_val + tot_val2 + tot_val3)/3
print(avg_test, avg_train, avg_val)

```

```

0.840733333333 0.691616800888 0.680254166667

```

```

In [ ]:

```



# DecisionTreeADULT

March 24, 2018

```
In [1]: import scipy.io as sio
import matplotlib.pyplot as plt
import numpy as np
import seaborn as sns
import pandas as pd
from sklearn import tree
from sklearn.model_selection import GridSearchCV
%config InlineBackend.figure_format = 'retina'

In [2]: df = pd.read_csv('C:/Users/Preston Wong/Downloads/adult.csv', index_col = 0)
df = df.dropna(axis=0)
onehot = pd.get_dummies(data=df, columns=['Column2', 'Column4', 'Column6', 'Column8', 'Column9', 'Column10', 'Column11', 'Column12', 'Column13', 'Column14', 'Column15'], drop_first = True)

In [3]: onehot = onehot.drop(['Column15'], axis=1)

In [4]: df = onehot.join(df['Column15'])
df = df.dropna(axis=1)
df = df[df.notnull()]

In [5]: X_and_Y = df.as_matrix()
np.random.shuffle(X_and_Y)
X_and_Y = X_and_Y[:5000, :101]
X = X_and_Y[:, 0:-1]
Y = X_and_Y[:, -1]
print(X_and_Y.shape, X.shape, Y.shape)

(5000, 101) (5000, 100) (5000,)

In [6]: #change to binary classification
for i in range(len(Y)):
    if Y[i] == '>50K':
        Y[i] = 1
    else:
        Y[i] = 0

In [7]: print(Y[:200])
```

```
[0 0 0 0 0 0 1 0 0 1 0 1 0 1 0 1 0 0 0 0 1 0 0 0 0 0 0 0 1 0 1 0 1 0 0 0
0 1 0 0 0 1 0 0 0 1 0 1 0 0 0 0 0 0 0 0 1 0 0 0 0 1 0 0 0 0 0 1 0 0 1 1 0
0 1 1 0 0 0 0 0 0 1 0 0 0 0 1 0 0 0 1 0 0 1 0 0 0 0 0 0 0 0 1 0 0 1 0 0 0
0 0 0 0 0 0 0 1 0 0 1 0 0 1 0 0 0 0 0 0 0 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0
1 1 1 0 0 0 0 1 1 0 1 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 1 0 0 1 0 1 0 1 0 0 0
1 0 0 0 0 0 0 1 0 0 0 0 1 1 0]
```

```
In [8]: depth_list = []
        for i in range(1, 21, 1):
            depth_list.append(i)
        print(depth_list)
```

```
[1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20]
```

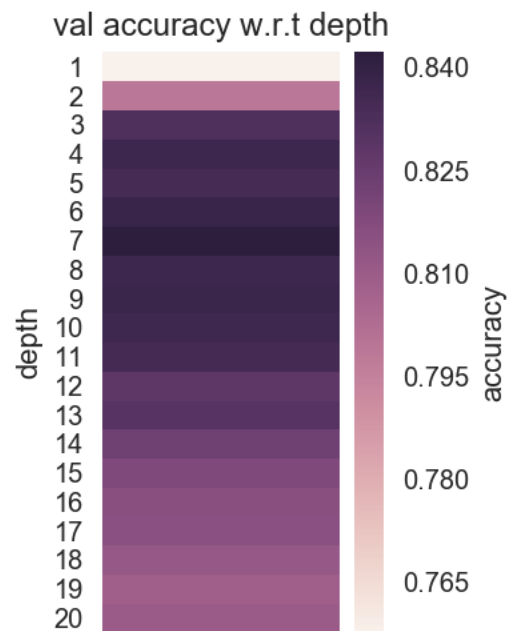
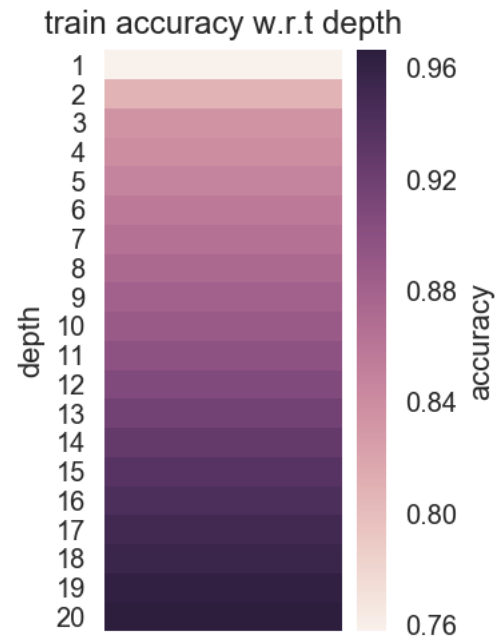
```
In [9]: classifier = tree.DecisionTreeClassifier(criterion = 'entropy')
        params = {'max_depth':depth_list}
        grid_search = GridSearchCV(classifier, params, return_train_score = True, cv=5)
```

```
In [10]: def draw_heatmap_linear(acc, acc_desc, depth_list):
          plt.figure(figsize = (2,4))
          ax = sns.heatmap(acc, annot=True, fmt='.3f', yticklabels=depth_list, xticklabels=acc_desc)
          ax.collections[0].colorbar.set_label("accuracy")
          ax.set(ylabel='depth')
          plt.title(acc_desc + ' w.r.t depth')
          sns.set_style("whitegrid", {'axes.grid' : False})
          plt.show()
```

```
In [11]: Y = Y.astype('int')
          X_train_val = X[:int(0.8*len(X))] # Get features from train + val set.
          X_test       = X[int(0.8*len(X)):] # Get features from test set.
          Y_train_val  = Y[:int(0.8*len(Y))] # Get labels from train + val set.
          Y_test       = Y[int(0.8*len(Y)):] # Get labels from test set.
```

```
In [12]: grid_search.fit(X_train_val, Y_train_val)
          train_acc = grid_search.cv_results_['mean_train_score'].reshape(-1,1)
          draw_heatmap_linear(train_acc, 'train accuracy', depth_list)

          val_acc = grid_search.cv_results_['mean_test_score'].reshape(-1,1)
          draw_heatmap_linear(val_acc, 'val accuracy', depth_list)
```



```
In [13]: test_acc = sum(grid_search.best_estimator_.predict(X_test) == Y_test) / len(Y_test)
          print(grid_search.best_params_)
          print(test_acc)
          tot_train = (sum(train_acc)/len(train_acc))
```

```

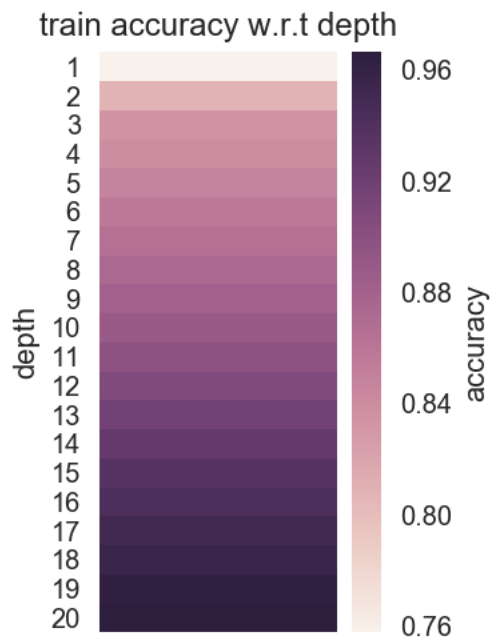
print(tot_train)
tot_val = (sum(val_acc)/len(val_acc))
print(tot_val)

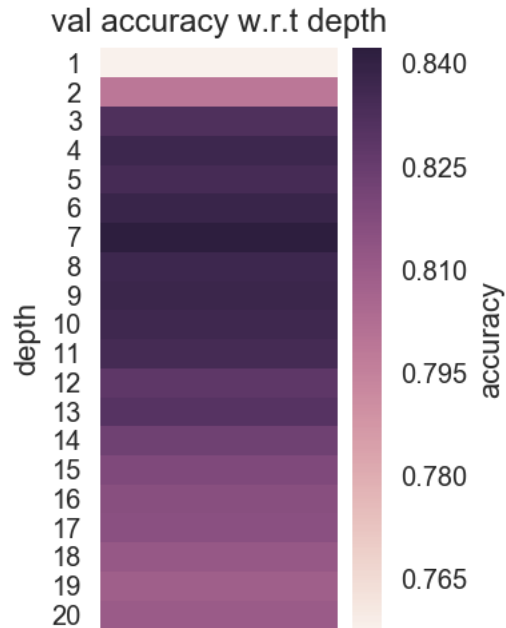
{'max_depth': 7}
0.84
[ 0.8909721]
[ 0.822575]

In [14]: X_train_val = X[:int(0.5*len(X))] # Get features from train + val set.
X_test   = X[int(0.5*len(X)):] # Get features from test set.
Y_train_val = Y[:int(0.5*len(Y))] # Get labels from train + val set.
Y_test    = Y[int(0.5*len(Y)):] # Get labels from test set.
grid_search.fit(X_train_val, Y_train_val)
train_acc2 = grid_search.cv_results_['mean_train_score'].reshape(-1,1)
draw_heatmap_linear(train_acc, 'train accuracy', depth_list)

val_acc2 = grid_search.cv_results_['mean_test_score'].reshape(-1,1)
draw_heatmap_linear(val_acc, 'val accuracy', depth_list)

```



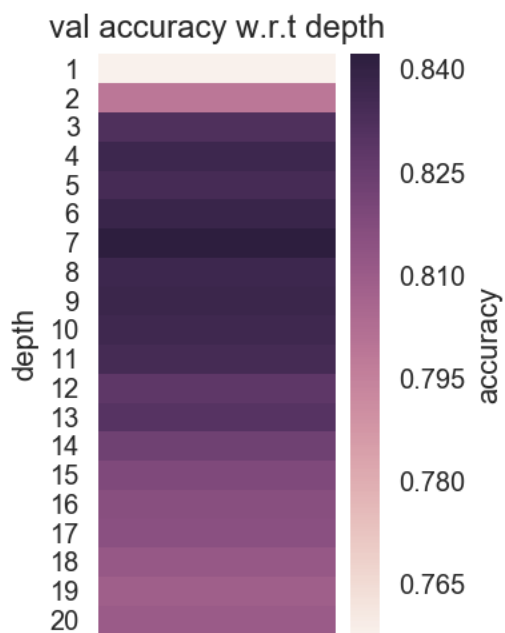
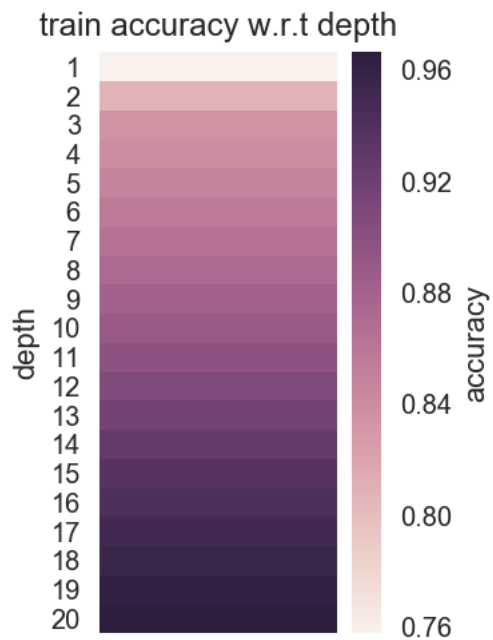


```
In [15]: test_acc2 = sum(grid_search.best_estimator_.predict(X_test) == Y_test) / len(Y_test)
print(grid_search.best_params_)
print(test_acc2)
tot_train2 = (sum(train_acc2)/len(train_acc2))
print(tot_train2)
tot_val2 = (sum(val_acc2)/len(val_acc2))
print(tot_val2)
```

```
{'max_depth': 8}
0.8472
[ 0.8934459]
[ 0.81122]
```

```
In [16]: X_train_val = X[:int(0.2*len(X))] # Get features from train + val set.
X_test = X[int(0.2*len(X)):] # Get features from test set.
Y_train_val = Y[:int(0.2*len(Y))] # Get labels from train + val set.
Y_test = Y[int(0.2*len(Y)):] # Get labels from test set.
grid_search.fit(X_train_val, Y_train_val)
train_acc3 = grid_search.cv_results_['mean_train_score'].reshape(-1,1)
draw_heatmap_linear(train_acc, 'train accuracy', depth_list)

val_acc3 = grid_search.cv_results_['mean_test_score'].reshape(-1,1)
draw_heatmap_linear(val_acc, 'val accuracy', depth_list)
```



```
In [17]: test_acc3 = sum(grid_search.best_estimator_.predict(X_test) == Y_test) / len(Y_test)
          print(grid_search.best_params_)
          print(test_acc3)
          tot_train3 = (sum(train_acc3)/len(train_acc3))
```

```

        print(tot_train3)
        tot_val3 = (sum(val_acc3)/len(val_acc3))
        print(tot_val3)

{'max_depth': 9}
0.81575
[ 0.89980536]
[ 0.7917]

In [18]: avg_test = (test_acc + test_acc2 + test_acc3)/3
        avg_train = sum(tot_train + tot_train2 + tot_train3)/3
        avg_val = sum(tot_val + tot_val2 + tot_val3)/3
        print(avg_test, avg_train, avg_val)

0.834316666667 0.894741119271 0.808498333333

In [ ]:

```

# DecisionTreeCOV

March 24, 2018

```
In [1]: import scipy.io as sio
import matplotlib.pyplot as plt
import numpy as np
import seaborn as sns
import pandas as pd
from sklearn import tree
from sklearn.model_selection import GridSearchCV
%config InlineBackend.figure_format = 'retina'
```

```
In [2]: df = pd.read_csv('C:/Users/Preston Wong/Downloads/covtype.data.gz', compression='gzip')
df.shape
```

```
Out[2]: (581011, 55)
```

```
In [3]: X_and_Y = df.as_matrix()
X_and_Y = X_and_Y[:5000, :55]
X = X_and_Y[:, 0:-1]
Y = X_and_Y[:, -1]
print(X_and_Y.shape, X.shape, Y.shape)
```

```
(5000, 55) (5000, 54) (5000,)
```

```
In [4]: #change to binary classification
for i in range(len(Y)):
    if Y[i] == 2:
        Y[i] = 1
    else:
        Y[i] = 0
np.random.shuffle(X_and_Y)
```

```
In [5]: depth_list = []
for i in range(1, 21, 1):
    depth_list.append(i)
print(depth_list)
```

```
[1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20]
```



```

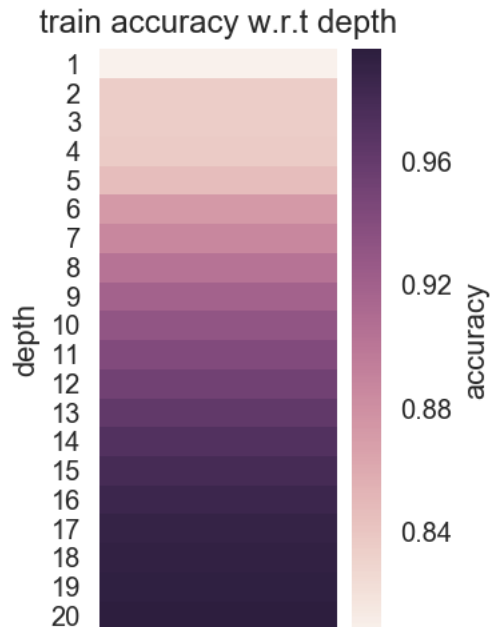
In [6]: classifier = tree.DecisionTreeClassifier(criterion = 'entropy')
        params = {'max_depth':depth_list}
        grid_search = GridSearchCV(classifier, params, return_train_score = True, c

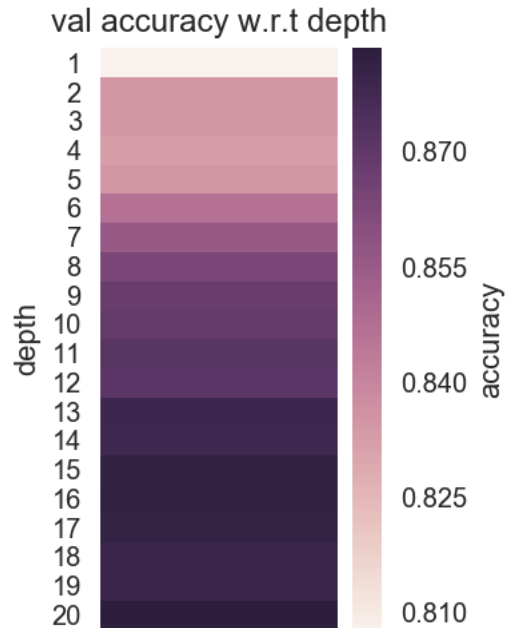
In [7]: def draw_heatmap_linear(acc, acc_desc, depth_list):
        plt.figure(figsize = (2,4))
        ax = sns.heatmap(acc, annot=True, fmt='.3f', yticklabels=depth_list, xt
        ax.collections[0].colorbar.set_label("accuracy")
        ax.set(ylabel='depth')
        plt.title(acc_desc + ' w.r.t depth')
        sns.set_style("whitegrid", {'axes.grid' : False})
        plt.show()

In [8]: X_train_val = X[:int(0.8*len(X))] # Get features from train + val set.
        X_test      = X[int(0.8*len(X)):] # Get features from test set.
        Y_train_val = Y[:int(0.8*len(Y))] # Get labels from train + val set.
        Y_test      = Y[int(0.8*len(Y)):] # Get labels from test set.
        grid_search.fit(X_train_val, Y_train_val)
        train_acc = grid_search.cv_results_['mean_train_score'].reshape(-1,1)
        draw_heatmap_linear(train_acc, 'train accuracy', depth_list)

        val_acc = grid_search.cv_results_['mean_test_score'].reshape(-1,1)
        draw_heatmap_linear(val_acc, 'val accuracy', depth_list)

```



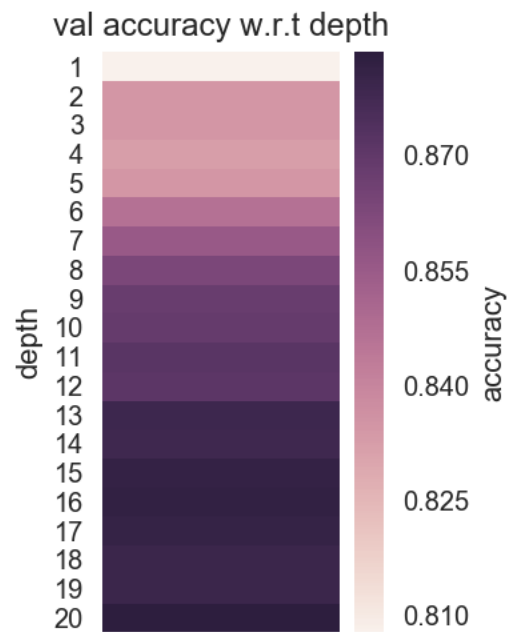
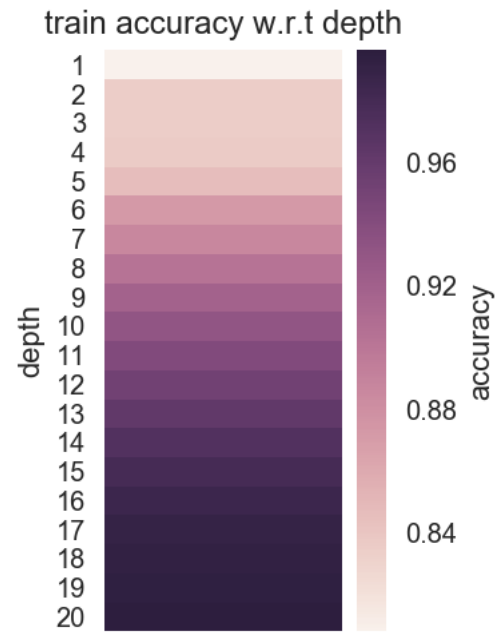


```
In [9]: test_acc = sum(grid_search.best_estimator_.predict(X_test) == Y_test) / len
print(grid_search.best_params_)
print(test_acc)
tot_train = (sum(train_acc)/len(train_acc))
print(tot_train)
tot_val = (sum(val_acc)/len(val_acc))
print(tot_val)
```

```
{'max_depth': 20}
0.893
[ 0.92231545]
[ 0.861775]
```

```
In [10]: X_train_val = X[:int(0.5*len(X))] # Get features from train + val set.
X_test    = X[int(0.5*len(X)):] # Get features from test set.
Y_train_val = Y[:int(0.5*len(Y))] # Get labels from train + val set.
Y_test     = Y[int(0.5*len(Y)):] # Get labels from test set.
grid_search.fit(X_train_val, Y_train_val)
train_acc2 = grid_search.cv_results_['mean_train_score'].reshape(-1,1)
draw_heatmap_linear(train_acc, 'train accuracy', depth_list)

val_acc2 = grid_search.cv_results_['mean_test_score'].reshape(-1,1)
draw_heatmap_linear(val_acc, 'val accuracy', depth_list)
```



```
In [11]: test_acc2 = sum(grid_search.best_estimator_.predict(X_test) == Y_test) / len(Y_test)
          print(grid_search.best_params_)
          print(test_acc2)
          tot_train2 = (sum(train_acc2)/len(train_acc2))
```

```

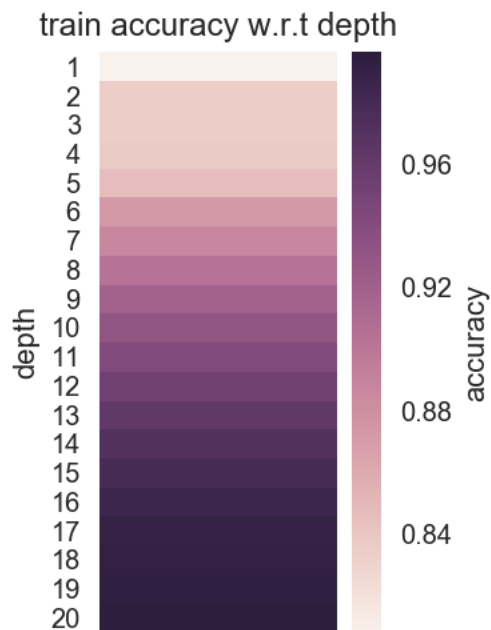
print(tot_train2)
tot_val2 = (sum(val_acc2)/len(val_acc2))
print(tot_val2)

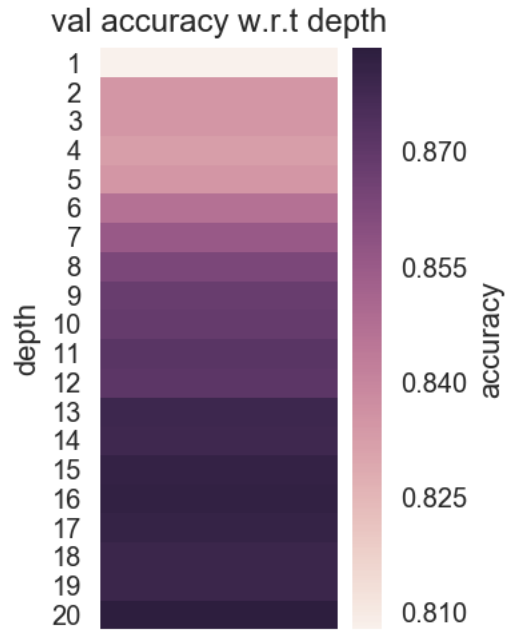
{'max_depth': 13}
0.8864
[ 0.92602057]
[ 0.86132]

In [12]: X_train_val = X[:int(0.2*len(X))] # Get features from train + val set.
X_test    = X[int(0.2*len(X)):] # Get features from test set.
Y_train_val = Y[:int(0.2*len(Y))] # Get labels from train + val set.
Y_test     = Y[int(0.2*len(Y)):] # Get labels from test set.
grid_search.fit(X_train_val, Y_train_val)
train_acc3 = grid_search.cv_results_['mean_train_score'].reshape(-1,1)
draw_heatmap_linear(train_acc, 'train accuracy', depth_list)

val_acc3 = grid_search.cv_results_['mean_test_score'].reshape(-1,1)
draw_heatmap_linear(val_acc, 'val accuracy', depth_list)

```





```
In [13]: test_acc3 = sum(grid_search.best_estimator_.predict(X_test) == Y_test) / len(Y_test)
          print(grid_search.best_params_)
          print(test_acc3)
          tot_train3 = (sum(train_acc3)/len(train_acc3))
          print(tot_train3)
          tot_val3 = (sum(val_acc3)/len(val_acc3))
          print(tot_val3)
```

```
{'max_depth': 18}
0.86325
[ 0.93723405]
[ 0.8538]
```

```
In [14]: avg_test = (test_acc + test_acc2 + test_acc3)/3
          avg_train = sum(tot_train + tot_train2 + tot_train3)/3
          avg_val = sum(tot_val + tot_val2 + tot_val3)/3
          print(avg_test, avg_train, avg_val)
```

```
0.880883333333 0.928523357211 0.858965
```

# DecisionTreeLETTERS

March 24, 2018

```
In [1]: import scipy.io as sio
import matplotlib.pyplot as plt
import numpy as np
import seaborn as sns
import pandas as pd
from sklearn import tree
from sklearn.model_selection import GridSearchCV
from string import ascii_uppercase
%config InlineBackend.figure_format = 'retina'
```

```
In [2]: df = pd.read_csv('C:/Users/Preston Wong/Downloads/letter-recognition.data',
```

```
In [3]: df2 = df.copy()
df = df.drop([0], axis=1)
df = df.join(df2[0])
```

```
In [4]: df.shape
```

```
Out[4]: (20000, 17)
```

```
In [5]: X_and_Y = df.as_matrix()
#np.random.shuffle(X_and_Y)
X_and_Y = X_and_Y[:5000, :17]
X = X_and_Y[:, 0:-1]
Y = X_and_Y[:, -1]
print(X_and_Y.shape, X.shape, Y.shape)
```

```
(5000, 17) (5000, 16) (5000,)
```

```
In [6]: #change to binary classification
#alphabet_list = []
ordlist = []
for i in range(len(Y)):
    #alphabet_list.append(Y[i])
    ordlist.append(ord(Y[i]))
```

```
In [7]: #print(alphabet_list)
#print(ordlist)
```

```

In [8]: for i in range(len(Y)):
        if ((ordlist[i]) >= 65 and (ordlist[i] <= 77)):
            Y[i] = 1
        else:
            Y[i] = 0

In [9]: depth_list = []
        for i in range(1, 21, 1):
            depth_list.append(i)
        print(depth_list)

[1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20]

In [10]: classifier = tree.DecisionTreeClassifier(criterion = 'entropy')
        params = {'max_depth':depth_list}
        grid_search = GridSearchCV(classifier, params, return_train_score = True,

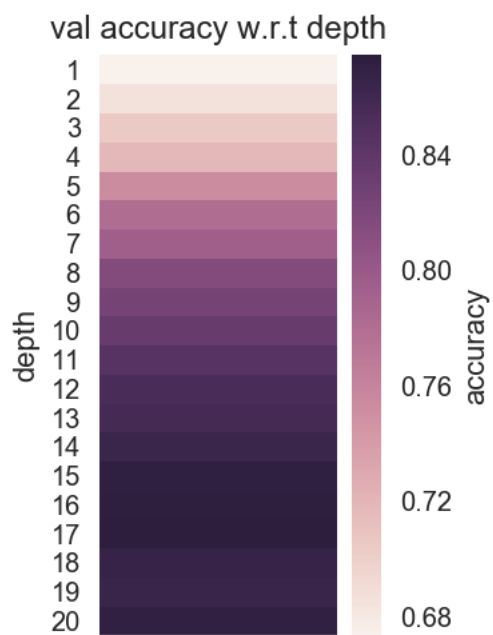
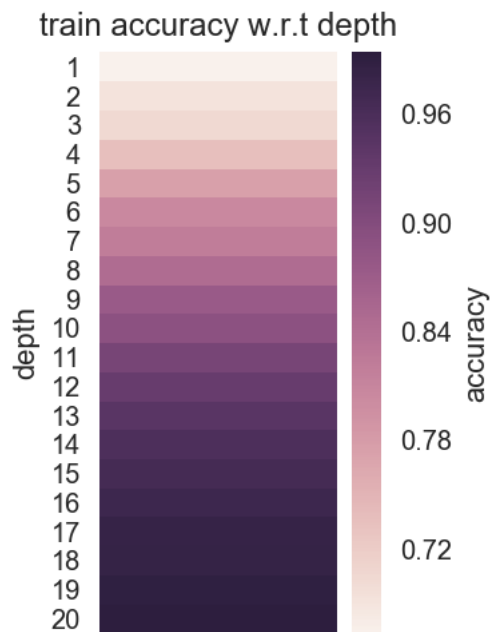
In [11]: def draw_heatmap_linear(acc, acc_desc, depth_list):
        plt.figure(figsize = (2,4))
        ax = sns.heatmap(acc, annot=True, fmt='.3f', yticklabels=depth_list, x
        ax.collections[0].colorbar.set_label("accuracy")
        ax.set(ylabel='depth')
        plt.title(acc_desc + ' w.r.t depth')
        sns.set_style("whitegrid", {'axes.grid' : False})
        plt.show()

In [12]: Y = Y.astype('int')
        X_train_val = X[:int(0.8*len(X))] # Get features from train + val set.
        X_test      = X[int(0.8*len(X)):] # Get features from test set.
        Y_train_val = Y[:int(0.8*len(Y))] # Get labels from train + val set.
        Y_test      = Y[int(0.8*len(Y)):] # Get labels from test set.

In [13]: grid_search.fit(X_train_val, Y_train_val)
        train_acc = grid_search.cv_results_['mean_train_score'].reshape(-1,1)
        draw_heatmap_linear(train_acc, 'train accuracy', depth_list)

        val_acc = grid_search.cv_results_['mean_test_score'].reshape(-1,1)
        draw_heatmap_linear(val_acc, 'val accuracy', depth_list)

```



```
In [14]: test_acc = sum(grid_search.best_estimator_.predict(X_test) == Y_test) / len(Y_test)
          print(grid_search.best_params_)
          print(test_acc)
          tot_train = (sum(train_acc)/len(train_acc))
```



```

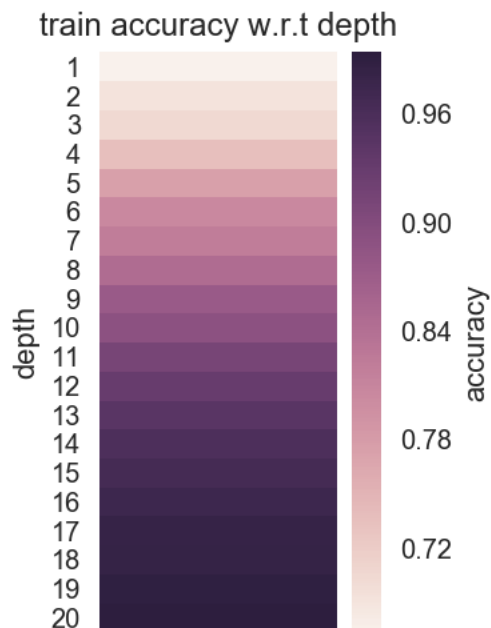
print(tot_train)
tot_val = (sum(val_acc)/len(val_acc))
print(tot_val)

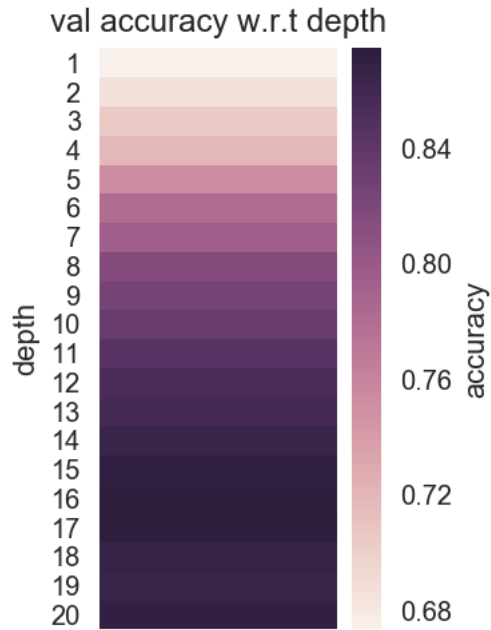
{'max_depth': 17}
0.88
[ 0.87318125]
[ 0.8118375]

In [15]: X_train_val = X[:int(0.5*len(X))] # Get features from train + val set.
X_test    = X[int(0.5*len(X)):] # Get features from test set.
Y_train_val = Y[:int(0.5*len(Y))] # Get labels from train + val set.
Y_test     = Y[int(0.5*len(Y)):] # Get labels from test set.
grid_search.fit(X_train_val, Y_train_val)
train_acc2 = grid_search.cv_results_['mean_train_score'].reshape(-1,1)
draw_heatmap_linear(train_acc, 'train accuracy', depth_list)

val_acc2 = grid_search.cv_results_['mean_test_score'].reshape(-1,1)
draw_heatmap_linear(val_acc, 'val accuracy', depth_list)

```



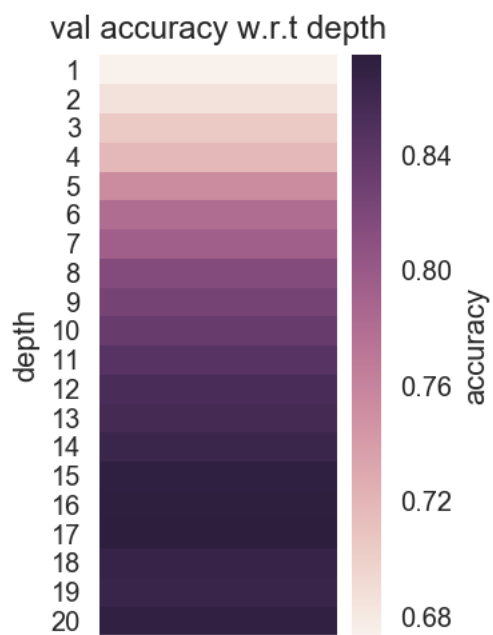
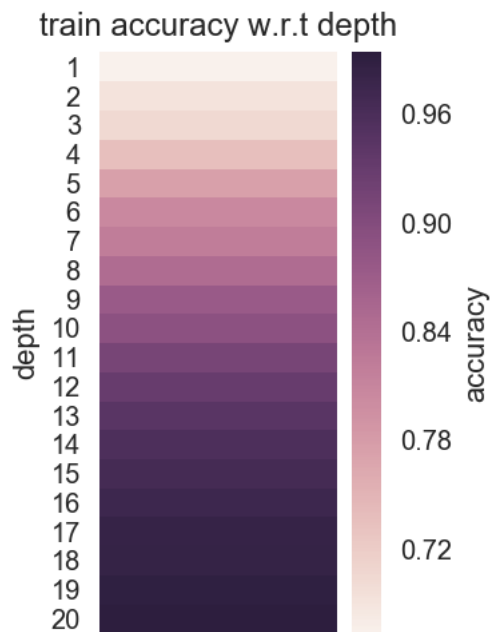


```
In [16]: test_acc2 = sum(grid_search.best_estimator_.predict(X_test) == Y_test) / len(Y_test)
print(grid_search.best_params_)
print(test_acc2)
tot_train2 = (sum(train_acc2)/len(train_acc2))
print(tot_train2)
tot_val2 = (sum(val_acc2)/len(val_acc2))
print(tot_val2)
```

```
{'max_depth': 17}
0.8468
[ 0.8841622]
[ 0.80396]
```

```
In [17]: X_train_val = X[:int(0.2*len(X))] # Get features from train + val set.
X_test    = X[int(0.2*len(X)):] # Get features from test set.
Y_train_val = Y[:int(0.2*len(Y))] # Get labels from train + val set.
Y_test     = Y[int(0.2*len(Y)):] # Get labels from test set.
grid_search.fit(X_train_val, Y_train_val)
train_acc3 = grid_search.cv_results_['mean_train_score'].reshape(-1,1)
draw_heatmap_linear(train_acc, 'train accuracy', depth_list)

val_acc3 = grid_search.cv_results_['mean_test_score'].reshape(-1,1)
draw_heatmap_linear(val_acc, 'val accuracy', depth_list)
```



```
In [18]: test_acc3 = sum(grid_search.best_estimator_.predict(X_test) == Y_test) / len(Y_test)
          print(grid_search.best_params_)
          print(test_acc3)
          tot_train3 = (sum(train_acc3)/len(train_acc3))
```

```

    print(tot_train3)
    tot_val3 = (sum(val_acc3)/len(val_acc3))
    print(tot_val3)

{'max_depth': 18}
0.8175
[ 0.88418269]
[ 0.75975]

In [19]: avg_test = (test_acc + test_acc2 + test_acc3)/3
          avg_train = sum(tot_train + tot_train2 + tot_train3)/3
          avg_val = sum(tot_val + tot_val2 + tot_val3)/3
          print(avg_test, avg_train, avg_val)

0.8481 0.880508712379 0.791849166667

In [ ]:

```

## RandomForestADULT

March 24, 2018

```
In [1]: import scipy.io as sio
import matplotlib.pyplot as plt
import numpy as np
import seaborn as sns
import pandas as pd
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import GridSearchCV
%config InlineBackend.figure_format = 'retina'

In [2]: df = pd.read_csv('C:/Users/Preston Wong/Downloads/adult.csv', index_col = 0)
df = df.dropna(axis=0)
onehot = pd.get_dummies(data=df, columns=['Column2', 'Column4', 'Column6', 'Column8', 'Column10', 'Column12', 'Column14', 'Column16', 'Column18', 'Column20', 'Column22', 'Column24', 'Column26', 'Column28', 'Column30', 'Column32', 'Column34', 'Column36', 'Column38', 'Column40', 'Column42', 'Column44', 'Column46', 'Column48', 'Column50', 'Column52', 'Column54', 'Column56', 'Column58', 'Column60', 'Column62', 'Column64', 'Column66', 'Column68', 'Column70', 'Column72', 'Column74', 'Column76', 'Column78', 'Column80', 'Column82', 'Column84', 'Column86', 'Column88', 'Column90', 'Column92', 'Column94', 'Column96', 'Column98', 'Column100', 'Column102', 'Column104', 'Column106', 'Column108', 'Column110', 'Column112', 'Column114', 'Column116', 'Column118', 'Column120', 'Column122', 'Column124', 'Column126', 'Column128', 'Column130', 'Column132', 'Column134', 'Column136', 'Column138', 'Column140', 'Column142', 'Column144', 'Column146', 'Column148', 'Column150', 'Column152', 'Column154', 'Column156', 'Column158', 'Column160', 'Column162', 'Column164', 'Column166', 'Column168', 'Column170', 'Column172', 'Column174', 'Column176', 'Column178', 'Column180', 'Column182', 'Column184', 'Column186', 'Column188', 'Column190', 'Column192', 'Column194', 'Column196', 'Column198', 'Column200', 'Column202', 'Column204', 'Column206', 'Column208', 'Column210', 'Column212', 'Column214', 'Column216', 'Column218', 'Column220', 'Column222', 'Column224', 'Column226', 'Column228', 'Column230', 'Column232', 'Column234', 'Column236', 'Column238', 'Column240', 'Column242', 'Column244', 'Column246', 'Column248', 'Column250', 'Column252', 'Column254', 'Column256', 'Column258', 'Column260', 'Column262', 'Column264', 'Column266', 'Column268', 'Column270', 'Column272', 'Column274', 'Column276', 'Column278', 'Column280', 'Column282', 'Column284', 'Column286', 'Column288', 'Column290', 'Column292', 'Column294', 'Column296', 'Column298', 'Column300', 'Column302', 'Column304', 'Column306', 'Column308', 'Column310', 'Column312', 'Column314', 'Column316', 'Column318', 'Column320', 'Column322', 'Column324', 'Column326', 'Column328', 'Column330', 'Column332', 'Column334', 'Column336', 'Column338', 'Column340', 'Column342', 'Column344', 'Column346', 'Column348', 'Column350', 'Column352', 'Column354', 'Column356', 'Column358', 'Column360', 'Column362', 'Column364', 'Column366', 'Column368', 'Column370', 'Column372', 'Column374', 'Column376', 'Column378', 'Column380', 'Column382', 'Column384', 'Column386', 'Column388', 'Column390', 'Column392', 'Column394', 'Column396', 'Column398', 'Column400', 'Column402', 'Column404', 'Column406', 'Column408', 'Column410', 'Column412', 'Column414', 'Column416', 'Column418', 'Column420', 'Column422', 'Column424', 'Column426', 'Column428', 'Column430', 'Column432', 'Column434', 'Column436', 'Column438', 'Column440', 'Column442', 'Column444', 'Column446', 'Column448', 'Column450', 'Column452', 'Column454', 'Column456', 'Column458', 'Column460', 'Column462', 'Column464', 'Column466', 'Column468', 'Column470', 'Column472', 'Column474', 'Column476', 'Column478', 'Column480', 'Column482', 'Column484', 'Column486', 'Column488', 'Column490', 'Column492', 'Column494', 'Column496', 'Column498', 'Column500', 'Column502', 'Column504', 'Column506', 'Column508', 'Column510', 'Column512', 'Column514', 'Column516', 'Column518', 'Column520', 'Column522', 'Column524', 'Column526', 'Column528', 'Column530', 'Column532', 'Column534', 'Column536', 'Column538', 'Column540', 'Column542', 'Column544', 'Column546', 'Column548', 'Column550', 'Column552', 'Column554', 'Column556', 'Column558', 'Column560', 'Column562', 'Column564', 'Column566', 'Column568', 'Column570', 'Column572', 'Column574', 'Column576', 'Column578', 'Column580', 'Column582', 'Column584', 'Column586', 'Column588', 'Column590', 'Column592', 'Column594', 'Column596', 'Column598', 'Column600', 'Column602', 'Column604', 'Column606', 'Column608', 'Column610', 'Column612', 'Column614', 'Column616', 'Column618', 'Column620', 'Column622', 'Column624', 'Column626', 'Column628', 'Column630', 'Column632', 'Column634', 'Column636', 'Column638', 'Column640', 'Column642', 'Column644', 'Column646', 'Column648', 'Column650', 'Column652', 'Column654', 'Column656', 'Column658', 'Column660', 'Column662', 'Column664', 'Column666', 'Column668', 'Column670', 'Column672', 'Column674', 'Column676', 'Column678', 'Column680', 'Column682', 'Column684', 'Column686', 'Column688', 'Column690', 'Column692', 'Column694', 'Column696', 'Column698', 'Column700', 'Column702', 'Column704', 'Column706', 'Column708', 'Column710', 'Column712', 'Column714', 'Column716', 'Column718', 'Column720', 'Column722', 'Column724', 'Column726', 'Column728', 'Column730', 'Column732', 'Column734', 'Column736', 'Column738', 'Column740', 'Column742', 'Column744', 'Column746', 'Column748', 'Column750', 'Column752', 'Column754', 'Column756', 'Column758', 'Column760', 'Column762', 'Column764', 'Column766', 'Column768', 'Column770', 'Column772', 'Column774', 'Column776', 'Column778', 'Column780', 'Column782', 'Column784', 'Column786', 'Column788', 'Column790', 'Column792', 'Column794', 'Column796', 'Column798', 'Column800', 'Column802', 'Column804', 'Column806', 'Column808', 'Column810', 'Column812', 'Column814', 'Column816', 'Column818', 'Column820', 'Column822', 'Column824', 'Column826', 'Column828', 'Column830', 'Column832', 'Column834', 'Column836', 'Column838', 'Column840', 'Column842', 'Column844', 'Column846', 'Column848', 'Column850', 'Column852', 'Column854', 'Column856', 'Column858', 'Column860', 'Column862', 'Column864', 'Column866', 'Column868', 'Column870', 'Column872', 'Column874', 'Column876', 'Column878', 'Column880', 'Column882', 'Column884', 'Column886', 'Column888', 'Column890', 'Column892', 'Column894', 'Column896', 'Column898', 'Column900', 'Column902', 'Column904', 'Column906', 'Column908', 'Column910', 'Column912', 'Column914', 'Column916', 'Column918', 'Column920', 'Column922', 'Column924', 'Column926', 'Column928', 'Column930', 'Column932', 'Column934', 'Column936', 'Column938', 'Column940', 'Column942', 'Column944', 'Column946', 'Column948', 'Column950', 'Column952', 'Column954', 'Column956', 'Column958', 'Column960', 'Column962', 'Column964', 'Column966', 'Column968', 'Column970', 'Column972', 'Column974', 'Column976', 'Column978', 'Column980', 'Column982', 'Column984', 'Column986', 'Column988', 'Column990', 'Column992', 'Column994', 'Column996', 'Column998', 'Column1000', 'Column1002', 'Column1004', 'Column1006', 'Column1008', 'Column1010', 'Column1012', 'Column1014', 'Column1016', 'Column1018', 'Column1020', 'Column1022', 'Column1024', 'Column1026', 'Column1028', 'Column1030', 'Column1032', 'Column1034', 'Column1036', 'Column1038', 'Column1040', 'Column1042', 'Column1044', 'Column1046', 'Column1048', 'Column1050', 'Column1052', 'Column1054', 'Column1056', 'Column1058', 'Column1060', 'Column1062', 'Column1064', 'Column1066', 'Column1068', 'Column1070', 'Column1072', 'Column1074', 'Column1076', 'Column1078', 'Column1080', 'Column1082', 'Column1084', 'Column1086', 'Column1088', 'Column1090', 'Column1092', 'Column1094', 'Column1096', 'Column1098', 'Column1100', 'Column1102', 'Column1104', 'Column1106', 'Column1108', 'Column1110', 'Column1112', 'Column1114', 'Column1116', 'Column1118', 'Column1120', 'Column1122', 'Column1124', 'Column1126', 'Column1128', 'Column1130', 'Column1132', 'Column1134', 'Column1136', 'Column1138', 'Column1140', 'Column1142', 'Column1144', 'Column1146', 'Column1148', 'Column1150', 'Column1152', 'Column1154', 'Column1156', 'Column1158', 'Column1160', 'Column1162', 'Column1164', 'Column1166', 'Column1168', 'Column1170', 'Column1172', 'Column1174', 'Column1176', 'Column1178', 'Column1180', 'Column1182', 'Column1184', 'Column1186', 'Column1188', 'Column1190', 'Column1192', 'Column1194', 'Column1196', 'Column1198', 'Column1200', 'Column1202', 'Column1204', 'Column1206', 'Column1208', 'Column1210', 'Column1212', 'Column1214', 'Column1216', 'Column1218', 'Column1220', 'Column1222', 'Column1224', 'Column1226', 'Column1228', 'Column1230', 'Column1232', 'Column1234', 'Column1236', 'Column1238', 'Column1240', 'Column1242', 'Column1244', 'Column1246', 'Column1248', 'Column1250', 'Column1252', 'Column1254', 'Column1256', 'Column1258', 'Column1260', 'Column1262', 'Column1264', 'Column1266', 'Column1268', 'Column1270', 'Column1272', 'Column1274', 'Column1276', 'Column1278', 'Column1280', 'Column1282', 'Column1284', 'Column1286', 'Column1288', 'Column1290', 'Column1292', 'Column1294', 'Column1296
```

```

0 0 1 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 1 0 0 0 0 0 0 0 0 1
0 0 0 0 0 1 0 0 0 0 0 1 0 1 0 1 0 1 0 0 0 0 0 0 0 0 0 1 0 1 0 0 0 0
1 0 0 0 1 0 0 0 1 0 0 0 1 0 0 1 1 0 1 0 0 0 0 0 1 0 0 0 1 0 0 0 0 0 1
0 0 0 0 1 0 0 0 0 1 1 0 1 0 1]

```

```
In [5]: print(Y[:200])
```

```

[0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 1 1 0 1 0 0 0 1 0 1 1 0 0 0
0 0 0 0 0 0 1 1 0 0 1 0 0 1 0 0 1 0 1 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 1 0
0 0 1 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 1 0 0 0 0 0 0 0 0 1
0 0 0 0 0 1 0 0 0 0 0 1 0 1 0 1 0 1 1 0 0 0 0 0 0 0 0 0 1 0 1 0 0 0 0 0
1 0 0 0 1 0 0 0 1 0 0 0 1 0 0 1 1 0 1 0 0 0 0 0 0 1 0 0 0 1 0 0 0 0 0 0 1
0 0 0 0 1 0 0 0 0 1 1 0 1 0 1]

```

```
In [6]: classifier = RandomForestClassifier()
split_list = [2,4,6,8,12,16,20]
params = {'min_samples_split':split_list}
grid_search = GridSearchCV(classifier, params, return_train_score = True, cv=5)
```

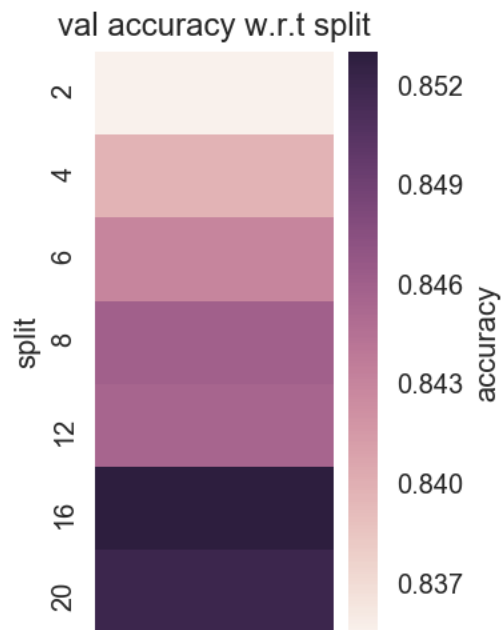
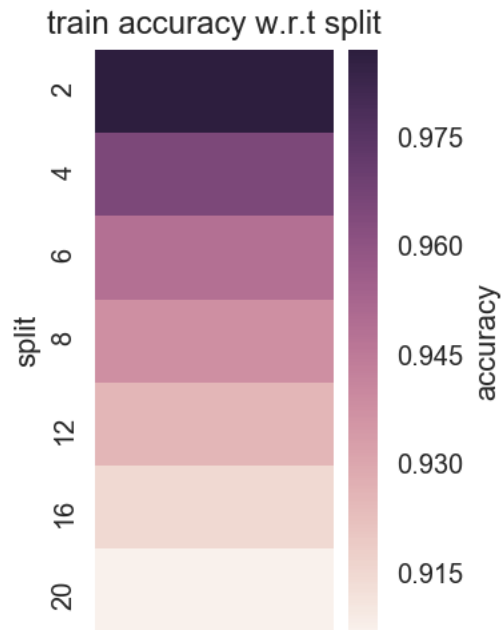
```
In [7]: def draw_heatmap_linear(acc, acc_desc, split_list):
plt.figure(figsize = (2,4))
ax = sns.heatmap(acc, annot=True, fmt='.3f', yticklabels=split_list, xticklabels=split_list)
ax.collections[0].colorbar.set_label("accuracy")
ax.set(ylabel='split')
plt.title(acc_desc + ' w.r.t split')
sns.set_style("whitegrid", {'axes.grid' : False})
plt.show()
```

```
In [8]: Y = Y.astype('int')
X_train_val = X[:int(0.8*len(X))] # Get features from train + val set.
X_test = X[int(0.8*len(X)):] # Get features from test set.
Y_train_val = Y[:int(0.8*len(Y))] # Get labels from train + val set.
Y_test = Y[int(0.8*len(Y)):] # Get labels from test set.
print(X_train_val.shape, X_test.shape, Y_train_val.shape, Y_test.shape)
grid_search.fit(X_train_val, Y_train_val)

train_acc = grid_search.cv_results_['mean_train_score'].reshape(-1,1)
draw_heatmap_linear(train_acc, 'train accuracy', split_list)

val_acc = grid_search.cv_results_['mean_test_score'].reshape(-1,1)
draw_heatmap_linear(val_acc, 'val accuracy', split_list)
```

```
(4000, 100) (1000, 100) (4000,) (1000,)
```



```
In [9]: test_acc = sum(grid_search.best_estimator_.predict(X_test) == Y_test) / len(Y_test)
print(grid_search.best_params_)
print(test_acc)
tot_train = (sum(train_acc)/len(train_acc))
```

```

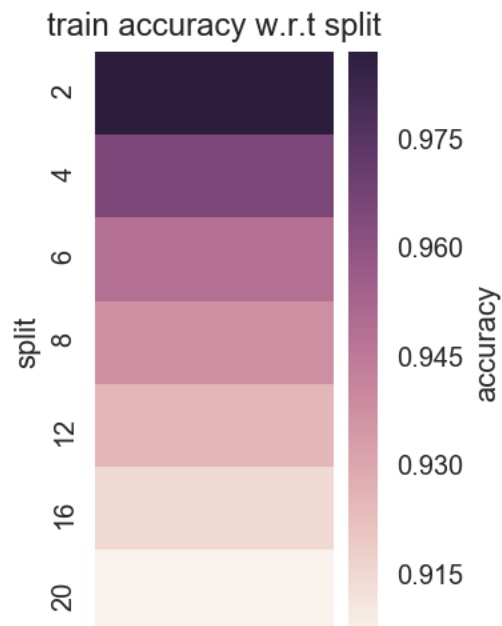
print(tot_train)
tot_val = (sum(val_acc)/len(val_acc))
print(tot_val)

{'min_samples_split': 16}
0.854
[ 0.94078526]
[ 0.84496429]

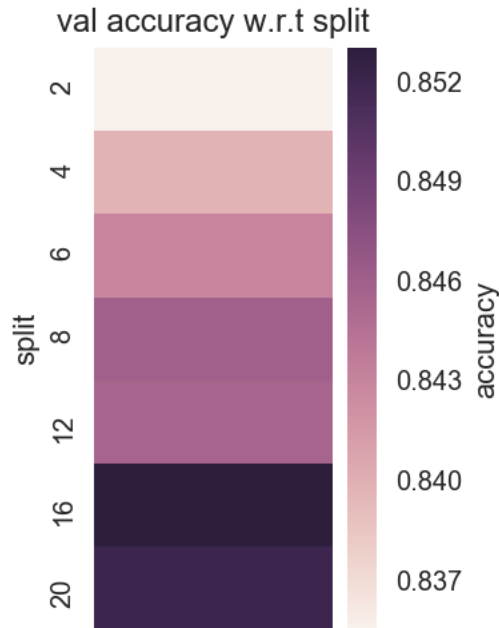
In [10]: X_train_val = X[:int(0.5*len(X))] # Get features from train + val set.
X_test    = X[int(0.5*len(X)):] # Get features from test set.
Y_train_val = Y[:int(0.5*len(Y))] # Get labels from train + val set.
Y_test     = Y[int(0.5*len(Y)):] # Get labels from test set.
grid_search.fit(X_train_val, Y_train_val)
train_acc2 = grid_search.cv_results_['mean_train_score'].reshape(-1,1)
draw_heatmap_linear(train_acc, 'train accuracy', split_list)

val_acc2 = grid_search.cv_results_['mean_test_score'].reshape(-1,1)
draw_heatmap_linear(val_acc, 'val accuracy', split_list)

```





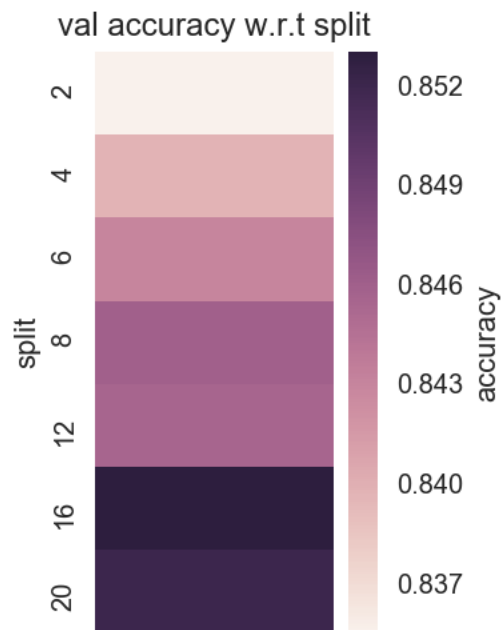
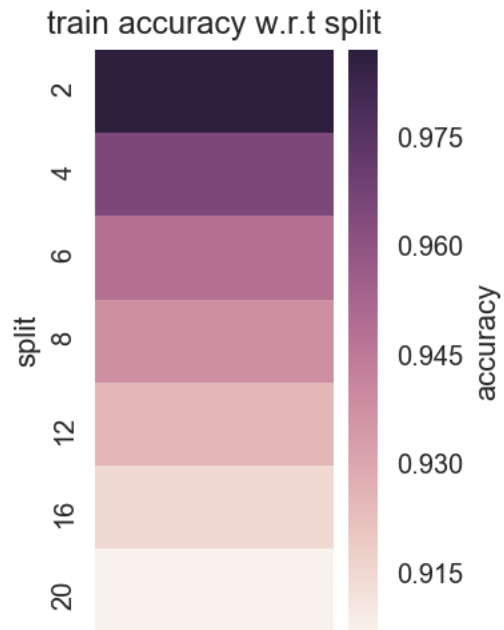


```
In [11]: test_acc2 = sum(grid_search.best_estimator_.predict(X_test) == Y_test) / len(Y_test)
print(grid_search.best_params_)
print(test_acc2)
tot_train2 = (sum(train_acc2)/len(train_acc2))
print(tot_train2)
tot_val2 = (sum(val_acc2)/len(val_acc2))
print(tot_val2)
```

```
{'min_samples_split': 8}
0.8444
[ 0.94187119]
[ 0.84205714]
```

```
In [12]: X_train_val = X[:int(0.2*len(X))] # Get features from train + val set.
X_test = X[int(0.2*len(X)):] # Get features from test set.
Y_train_val = Y[:int(0.2*len(Y))] # Get labels from train + val set.
Y_test = Y[int(0.2*len(Y)):] # Get labels from test set.
grid_search.fit(X_train_val, Y_train_val)
train_acc3 = grid_search.cv_results_['mean_train_score'].reshape(-1,1)
draw_heatmap_linear(train_acc, 'train accuracy', split_list)

val_acc3 = grid_search.cv_results_['mean_test_score'].reshape(-1,1)
draw_heatmap_linear(val_acc, 'val accuracy', split_list)
```



```
In [13]: test_acc3 = sum(grid_search.best_estimator_.predict(X_test) == Y_test) / len(Y_test)
print(grid_search.best_params_)
print(test_acc3)
tot_train3 = (sum(train_acc3)/len(train_acc3))
```

```

    print(tot_train3)
    tot_val3 = (sum(val_acc3)/len(val_acc3))
    print(tot_val3)

{'min_samples_split': 16}
0.84125
[ 0.94835649]
[ 0.855]

In [14]: avg_test = (test_acc + test_acc2 + test_acc3)/3
          avg_train = sum(tot_train + tot_train2 + tot_train3)/3
          avg_val = sum(tot_val + tot_val2 + tot_val3)/3
          print(avg_test, avg_train, avg_val)

0.84655 0.943670979299 0.84734047619

In [ ]:

```

# RandomForestCOV

March 24, 2018

```
In [1]: import scipy.io as sio
import matplotlib.pyplot as plt
import numpy as np
import seaborn as sns
import pandas as pd
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import GridSearchCV
%config InlineBackend.figure_format = 'retina'

In [2]: df = pd.read_csv('C:/Users/Preston Wong/Downloads/covtype.data.gz', compression='gzip')
X_and_Y = df.as_matrix()
X_and_Y = X_and_Y[:5000, :55]
X = X_and_Y[:, 0:-1]
Y = X_and_Y[:, -1]
print(X_and_Y.shape, X.shape, Y.shape)

(5000, 55) (5000, 54) (5000,)

In [3]: #change to binary classification
for i in range(len(Y)):
    if Y[i] == 2:
        Y[i] = 1
    else:
        Y[i] = 0
np.random.shuffle(X_and_Y)

In [4]: classifier = RandomForestClassifier()
split_list = [2,4,6,8,12,16,20]
params = {'min_samples_split':split_list}
grid_search = GridSearchCV(classifier, params, return_train_score = True, cv=5)

In [5]: def draw_heatmap_linear(acc, acc_desc, split_list):
plt.figure(figsize = (2,4))
ax = sns.heatmap(acc, annot=True, fmt='.3f', yticklabels=split_list, xticklabels=acc_desc)
ax.collections[0].colorbar.set_label("accuracy")
ax.set(ylabel='split')
plt.title(acc_desc + ' w.r.t split')
sns.set_style("whitegrid", {'axes.grid' : False})
plt.show()
```

```

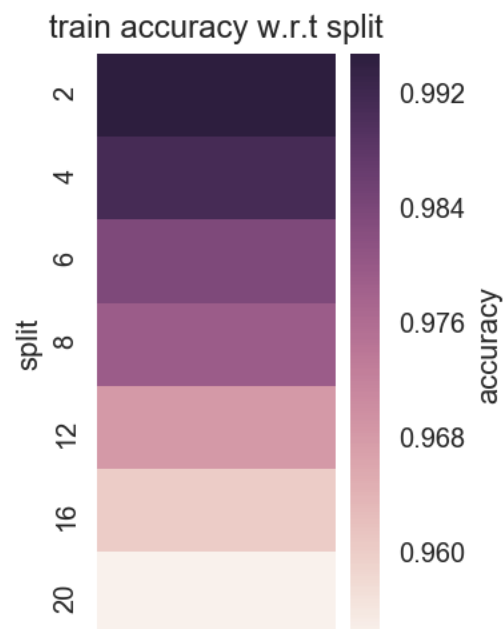
In [6]: X_train_val = X[:int(0.8*len(X))] # Get features from train + val set.
X_test   = X[int(0.8*len(X)):] # Get features from test set.
Y_train_val = Y[:int(0.8*len(Y))] # Get labels from train + val set.
Y_test    = Y[int(0.8*len(Y)):] # Get labels from test set.
print(X_train_val.shape, X_test.shape, Y_train_val.shape, Y_test.shape)
grid_search.fit(X_train_val, Y_train_val)

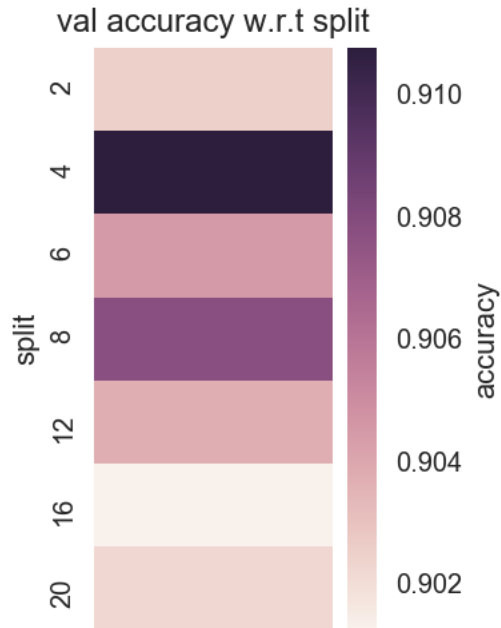
train_acc = grid_search.cv_results_['mean_train_score'].reshape(-1,1)
draw_heatmap_linear(train_acc, 'train accuracy', split_list)

val_acc = grid_search.cv_results_['mean_test_score'].reshape(-1,1)
draw_heatmap_linear(val_acc, 'val accuracy', split_list)

(4000, 54) (1000, 54) (4000,) (1000,)

```



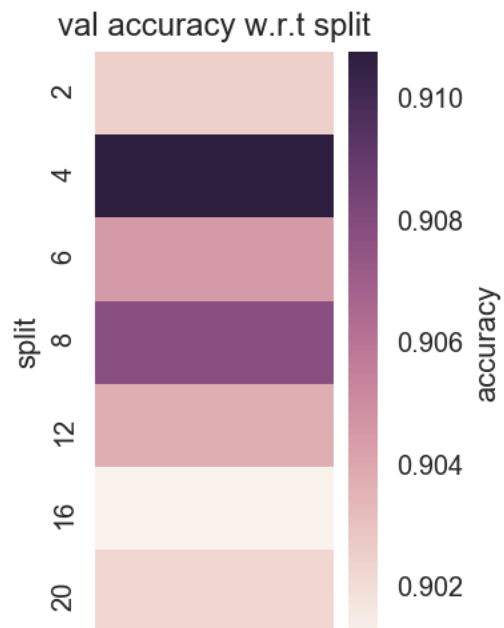
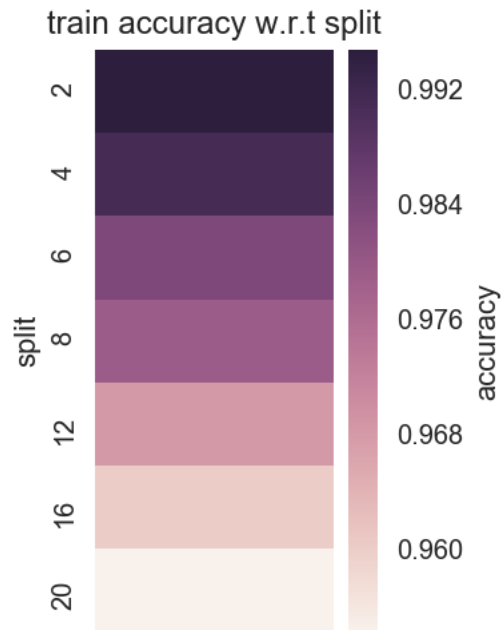


```
In [7]: test_acc = sum(grid_search.best_estimator_.predict(X_test) == Y_test) / len
print(grid_search.best_params_)
print(test_acc)
tot_train = (sum(train_acc)/len(train_acc))
print(tot_train)
tot_val = (sum(val_acc)/len(val_acc))
print(tot_val)
```

```
{'min_samples_split': 4}
0.901
[ 0.97588404]
[ 0.90467857]
```

```
In [8]: X_train_val = X[:int(0.5*len(X))] # Get features from train + val set.
X_test = X[int(0.5*len(X)):] # Get features from test set.
Y_train_val = Y[:int(0.5*len(Y))] # Get labels from train + val set.
Y_test = Y[int(0.5*len(Y)):] # Get labels from test set.
grid_search.fit(X_train_val, Y_train_val)
train_acc2 = grid_search.cv_results_['mean_train_score'].reshape(-1,1)
draw_heatmap_linear(train_acc, 'train accuracy', split_list)

val_acc2 = grid_search.cv_results_['mean_test_score'].reshape(-1,1)
draw_heatmap_linear(val_acc, 'val accuracy', split_list)
```



```
In [9]: test_acc2 = sum(grid_search.best_estimator_.predict(X_test) == Y_test) / len(Y_test)
print(grid_search.best_params_)
print(test_acc2)
tot_train2 = (sum(train_acc2)/len(train_acc2))
```

```

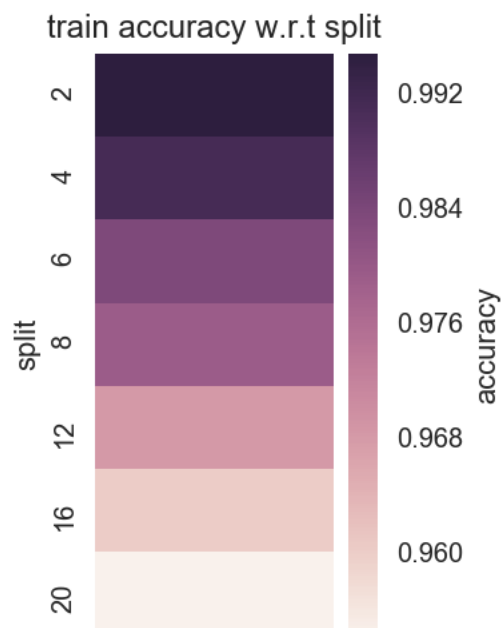
print(tot_train2)
tot_val2 = (sum(val_acc2)/len(val_acc2))
print(tot_val2)

{'min_samples_split': 2}
0.904
[ 0.97372873]
[ 0.89211429]

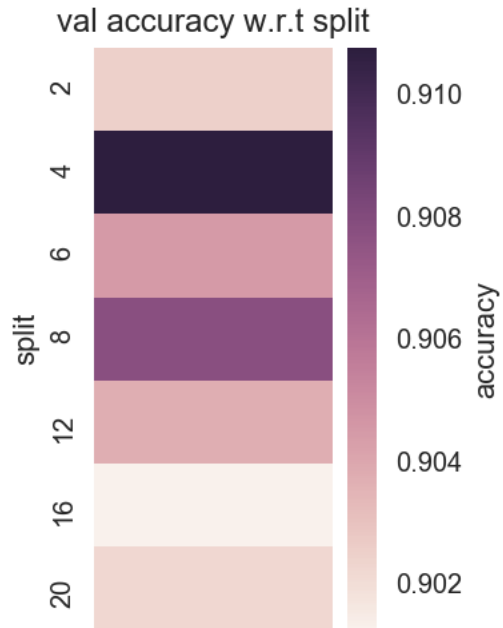
In [10]: X_train_val = X[:int(0.2*len(X))] # Get features from train + val set.
X_test    = X[int(0.2*len(X)):] # Get features from test set.
Y_train_val = Y[:int(0.2*len(Y))] # Get labels from train + val set.
Y_test     = Y[int(0.2*len(Y)):] # Get labels from test set.
grid_search.fit(X_train_val, Y_train_val)
train_acc3 = grid_search.cv_results_['mean_train_score'].reshape(-1,1)
draw_heatmap_linear(train_acc, 'train accuracy', split_list)

val_acc3 = grid_search.cv_results_['mean_test_score'].reshape(-1,1)
draw_heatmap_linear(val_acc, 'val accuracy', split_list)

```







```
In [11]: test_acc3 = sum(grid_search.best_estimator_.predict(X_test) == Y_test) / len(Y_test)
print(grid_search.best_params_)
print(test_acc3)
tot_train3 = (sum(train_acc3)/len(train_acc3))
print(tot_train3)
tot_val3 = (sum(val_acc3)/len(val_acc3))
print(tot_val3)
```

```
{'min_samples_split': 16}
0.8705
[ 0.96846418]
[ 0.87942857]
```

```
In [12]: avg_test = (test_acc + test_acc2 + test_acc3)/3
avg_train = sum(tot_train + tot_train2 + tot_train3)/3
avg_val = sum(tot_val + tot_val2 + tot_val3)/3
print(avg_test, avg_train, avg_val)
```

```
0.891833333333 0.972692313174 0.892073809524
```

# RandomForestLETTERS

March 24, 2018

```
In [1]: import scipy.io as sio
import matplotlib.pyplot as plt
import numpy as np
import seaborn as sns
import pandas as pd
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import GridSearchCV
%config InlineBackend.figure_format = 'retina'

In [2]: df = pd.read_csv('C:/Users/Preston Wong/Downloads/letter-recognition.data',
df2 = df.copy()
df = df.drop([0], axis=1)
df = df.join(df2[0])

In [3]: X_and_Y = df.as_matrix()
np.random.shuffle(X_and_Y)
X_and_Y = X_and_Y[:5000, :17]
X = X_and_Y[:, 0:-1]
Y = X_and_Y[:, -1]
print(X_and_Y.shape, X.shape, Y.shape)

(5000, 17) (5000, 16) (5000,)

In [4]: ordlist = []
for i in range(len(Y)):
    ordlist.append(ord(Y[i]))

In [5]: for i in range(len(Y)):
    if ((ordlist[i]) >= 65 and (ordlist[i] <= 77)):
        Y[i] = 1
    else:
        Y[i] = 0

In [6]: depth_list = []
for i in range(1, 21, 1):
    depth_list.append(i)
print(depth_list)
```

```
[1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20]
```

```
In [7]: classifier = RandomForestClassifier()
        split_list = [2,4,6,8,12,16,20]
        params = {'min_samples_split':split_list}
        grid_search = GridSearchCV(classifier, params, return_train_score = True, c
```

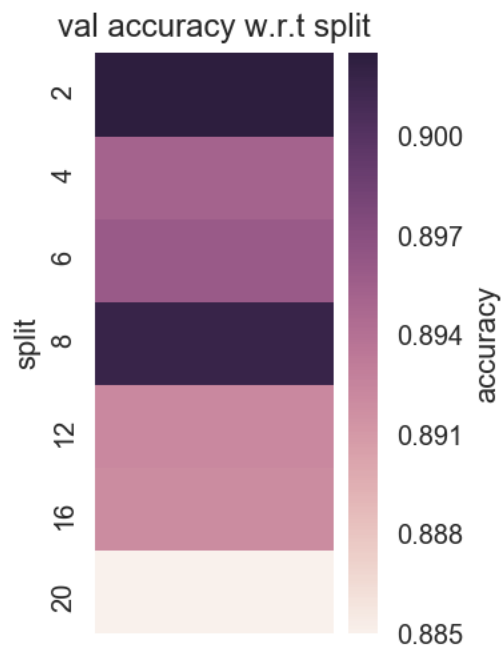
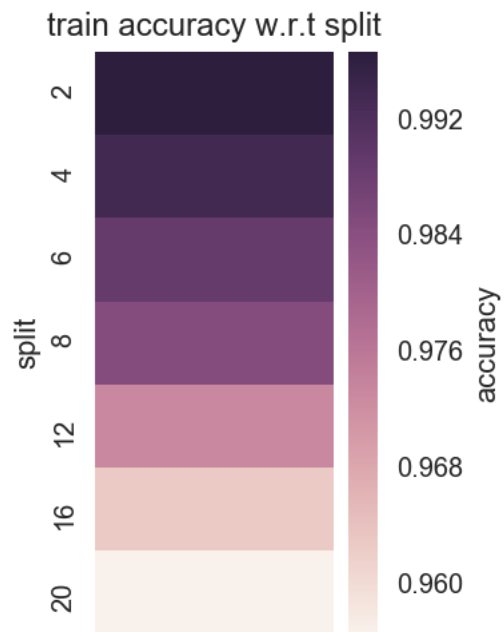
```
In [8]: def draw_heatmap_linear(acc, acc_desc, split_list):
        plt.figure(figsize = (2,4))
        ax = sns.heatmap(acc, annot=True, fmt='.3f', yticklabels=split_list, xt
        ax.collections[0].colorbar.set_label("accuracy")
        ax.set(ylabel='split')
        plt.title(acc_desc + ' w.r.t split')
        sns.set_style("whitegrid", {'axes.grid' : False})
        plt.show()
```

```
In [9]: Y = Y.astype('int')
        X_train_val = X[:int(0.8*len(X))] # Get features from train + val set.
        X_test       = X[int(0.8*len(X)):] # Get features from test set.
        Y_train_val = Y[:int(0.8*len(Y))] # Get labels from train + val set.
        Y_test       = Y[int(0.8*len(Y)):] # Get labels from test set.
        print(X_train_val.shape, X_test.shape, Y_train_val.shape, Y_test.shape)
        grid_search.fit(X_train_val, Y_train_val)

        train_acc = grid_search.cv_results_['mean_train_score'].reshape(-1,1)
        draw_heatmap_linear(train_acc, 'train accuracy', split_list)

        val_acc = grid_search.cv_results_['mean_test_score'].reshape(-1,1)
        draw_heatmap_linear(val_acc, 'val accuracy', split_list)
```

```
(4000, 16) (1000, 16) (4000,) (1000,)
```



```
In [10]: test_acc = sum(grid_search.best_estimator_.predict(X_test) == Y_test) / len(Y_test)
          print(grid_search.best_params_)
          print(test_acc)
```

```

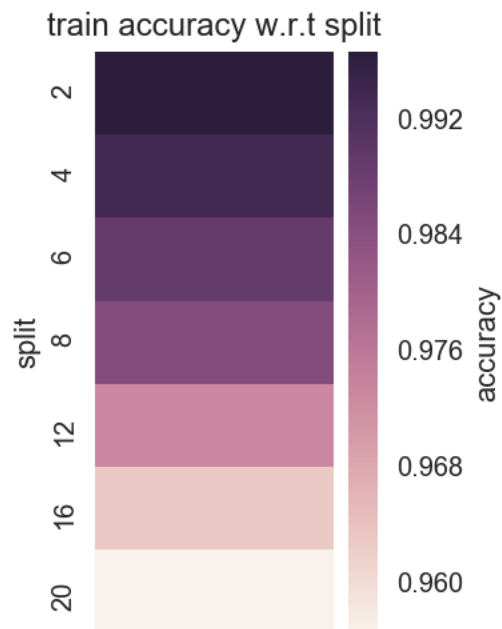
tot_train = (sum(train_acc)/len(train_acc))
print(tot_train)
tot_val = (sum(val_acc)/len(val_acc))
print(tot_val)

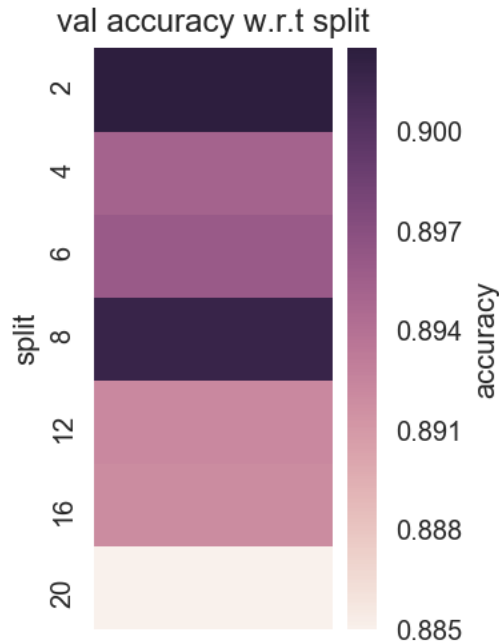
{'min_samples_split': 2}
0.919
[ 0.97942867]
[ 0.89496429]

In [11]: X_train_val = X[:int(0.5*len(X))] # Get features from train + val set.
X_test   = X[int(0.5*len(X)):] # Get features from test set.
Y_train_val = Y[:int(0.5*len(Y))] # Get labels from train + val set.
Y_test     = Y[int(0.5*len(Y)):] # Get labels from test set.
grid_search.fit(X_train_val, Y_train_val)
train_acc2 = grid_search.cv_results_['mean_train_score'].reshape(-1,1)
draw_heatmap_linear(train_acc, 'train accuracy', split_list)

val_acc2 = grid_search.cv_results_['mean_test_score'].reshape(-1,1)
draw_heatmap_linear(val_acc, 'val accuracy', split_list)

```



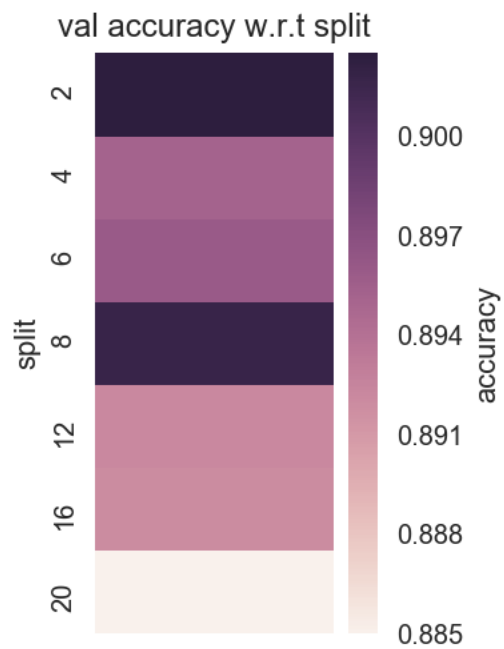
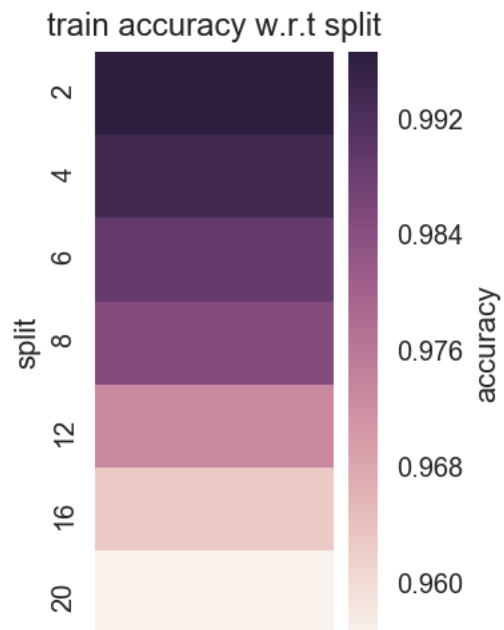


```
In [12]: test_acc2 = sum(grid_search.best_estimator_.predict(X_test) == Y_test) / len(Y_test)
print(grid_search.best_params_)
print(test_acc2)
tot_train2 = (sum(train_acc2)/len(train_acc2))
print(tot_train2)
tot_val2 = (sum(val_acc2)/len(val_acc2))
print(tot_val2)

{'min_samples_split': 2}
0.8924
[ 0.97495714]
[ 0.8812]
```

```
In [13]: X_train_val = X[:int(0.2*len(X))] # Get features from train + val set.
X_test    = X[int(0.2*len(X)):] # Get features from test set.
Y_train_val = Y[:int(0.2*len(Y))] # Get labels from train + val set.
Y_test     = Y[int(0.2*len(Y)):] # Get labels from test set.
grid_search.fit(X_train_val, Y_train_val)
train_acc3 = grid_search.cv_results_['mean_train_score'].reshape(-1,1)
draw_heatmap_linear(train_acc, 'train accuracy', split_list)

val_acc3 = grid_search.cv_results_['mean_test_score'].reshape(-1,1)
draw_heatmap_linear(val_acc, 'val accuracy', split_list)
```



```
In [14]: test_acc3 = sum(grid_search.best_estimator_.predict(X_test) == Y_test) / 1000
          print(grid_search.best_params_)
          print(test_acc3)
```

```

tot_train3 = (sum(train_acc3)/len(train_acc3))
print(tot_train3)
tot_val3 = (sum(val_acc3)/len(val_acc3))
print(tot_val3)

{'min_samples_split': 6}
0.84825
[ 0.96539286]
[ 0.82657143]

In [15]: avg_test = (test_acc + test_acc2 + test_acc3)/3
avg_train = sum(tot_train + tot_train2 + tot_train3)/3
avg_val = sum(tot_val + tot_val2 + tot_val3)/3
print(avg_test, avg_train, avg_val)

0.88655 0.973259556766 0.867578571429

```

```

In [ ]:

```