Model Selection

Bayesian Optimization for model selection to maximize accuracy.

In [1]:

```
from src.classifier.ThreadedBayesianSearcher impor
from src.classifier.BayesianTabularModelSearch imp
from src.classifier.CustomTabularModel import Cust
from src.recommender.BayesianRecommender import Ba
from bayes_opt import BayesianOptimization
from bayes_opt import UtilityFunction
import numpy as np

import matplotlib.pyplot as plt
from matplotlib import gridspec
%matplotlib inline
```

In [2]:

```
bayesian_optimizer = BayesianSearcher(20)
model = CustomTabularModel(0.5, False, 1000, {'lay
bayesian_optimizer.run_optimization(model, {'laye
bayesian_optimizer.run_optimization(model, {'laye
bayesian_optimizer.run_optimization(model, {'laye
```

Epoch 1: early stopping

20.00% [4/20 00:00<00:02]

epoch	train_loss	valid_loss	accuracy	time
0	0.689145	0.675664	0.822695	00:00
1	0.681097	0.674108	0.822695	00:00
2	0.679540	0.670044	0.822695	00:00
3	0.679220	0.663594	0.822695	00:00

100.00% [14/14 00:00<00:00]

Epoch 4: early stopping

20.00% [4/20

In [3]:

```
def maximization_function(**params: dict):
    """
    The function whose value we want to maximize.

    :param params: The parameters to set to the mo
    :return: The value of the metric used by the r
             Expected to be the validation acc.
    """
    params['early_stopping'] = False
    model.reset_params(params)
    return model.train(epochs=20, k=1)
```

In [4]:

```
x = [{'layer1': i} for i in range(2, 400, 50)]
y = [maximization_function(**layer_size) for layer
```

1	0.696800	0.660868	0.820331	00:00
2	0.688626	0.651283	0.820331	00:00
3	0.677082	0.635440	0.820331	00:00

100.00% [14/14 00:00<00:00]

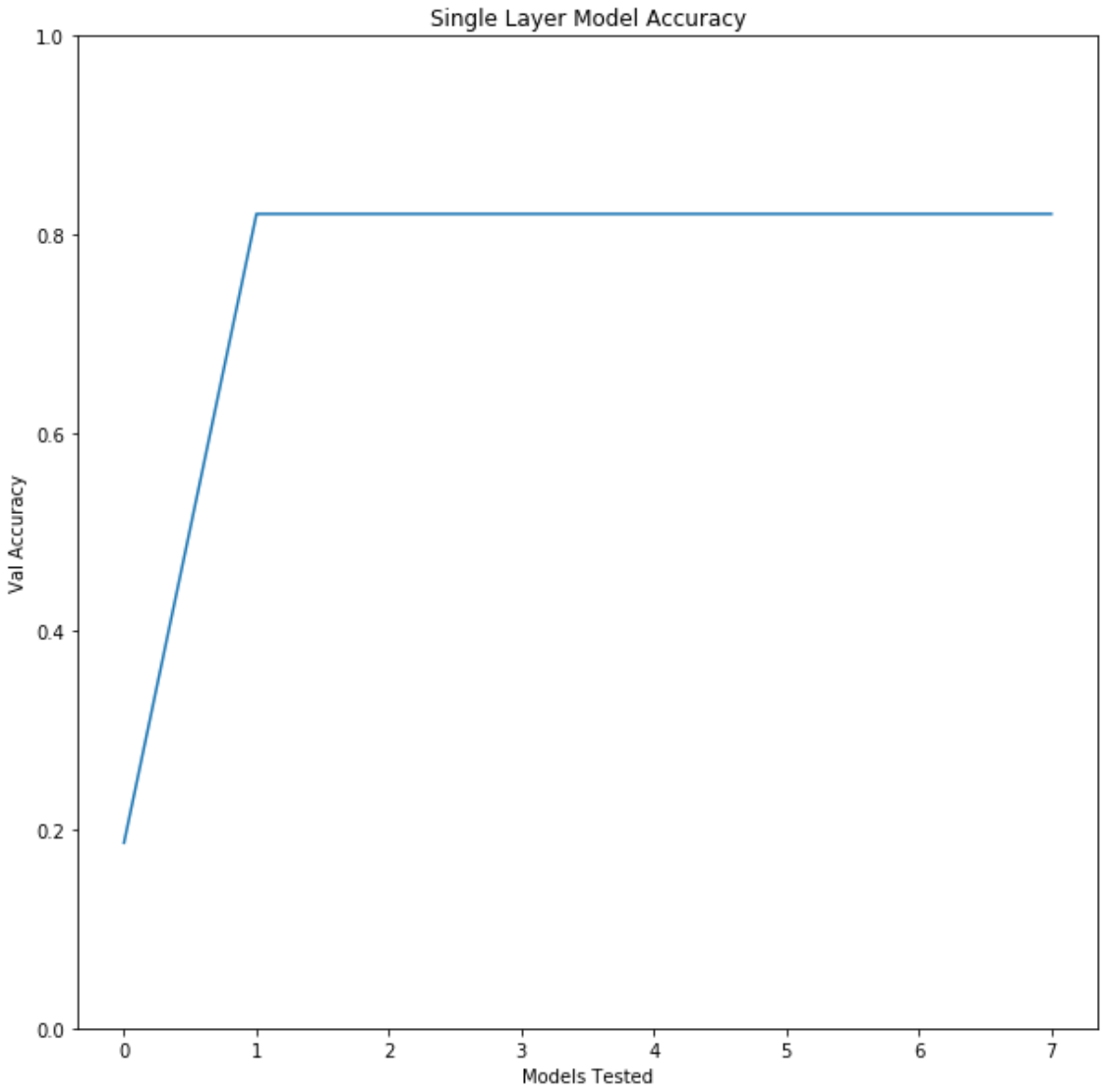
Epoch 4: early stopping

20.00% [4/20 00:00<00:03]

epoch	train_loss	valid_loss	accuracy	time
0	0.670903	0.663224	0.820331	00:00
1	0.647447	0.659451	0.820331	00:00
2	0.644057	0.650343	0.820331	00:00
3	0.641955	0.634390	0.820331	00:00

In [5]:
```
x_i = list(range(0, len(x)))

plt.figure(figsize=(10,10))
plt.title("Single Layer Model Accuracy")
plt.ylim((0, 1))
plt.xlabel("Models Tested")
plt.ylabel("Val Accuracy")
plt.plot(x_i, y)
```

```
[<matplotlib.lines.Line2D at 0x1a2888c828>]
```



In [6]:

```
def posterior(optimizer, x_obs, y_obs, grid):
    optimizer._gp.fit(x_obs, y_obs)

    mu, sigma = optimizer._gp.predict(grid, return
    return mu, sigma


def plot_gp(optimizer, x, y):
    fig = plt.figure(figsize=(10, 10))
    steps = len(optimizer.space)
    fig.suptitle(
        'Gaussian Process and Utility Function Aft
        fontdict={'size': 30}
    )

    gs = gridspec.GridSpec(2, 1, height_ratios=[3,
    axis = plt.subplot(gs[0])
    acq = plt.subplot(gs[1])

    x_obs = np.array([[res["params"]["layer1"]] fo
    y_obs = np.array([res["target"] for res in opt

    mu, sigma = posterior(optimizer, x_obs, y_obs,
    axis.plot(x, y, linewidth=3, label='Target')
    axis.plot(x_obs.flatten(), y_obs, 'D', markers
    axis.plot(x, mu, '--', color='k', label='Predi

    axis.fill(np.concatenate([x, x[::-1]]),
              np.concatenate([mu - 1.9600 * sigma,
              alpha=.6, fc='c', ec='None', label=
```

```
axis.set_xlim((-2, 10))
axis.set_ylim((None, None))
axis.set_ylabel('f( layer1 )', fontdict={'size
axis.set_xlabel('layer1', fontdict={'size': 2(

utility_function = UtilityFunction(kind="ucb",
utility = utility_function.utility(x, optimize
acq.plot(x, utility, label='Utility Function',
acq.plot(x[np.argmax(utility)], np.max(utility
         label=u'Next Best Guess', markerfacec
acq.set_xlim((-2, 10))
acq.set_ylim((0, np.max(utility) + 0.5))
acq.set_ylabel('Utility', fontdict={'size': 2(
acq.set_xlabel('layer1', fontdict={'size': 20}

axis.legend(loc=2, bbox_to_anchor=(1.01, 1), b
acq.legend(loc=2, bbox_to_anchor=(1.01, 1), bo
```

In [7]:
```
yesian_optimizer.optimizer.maximize(init_points=0, r
ot_gp(bayesian_optimizer.optimizer, np.array(x_i).re
```

4	0.698687	0.701711	0.749409	00:00
5	0.691048	0.675212	0.794326	00:00
6	0.675027	0.639234	0.806147	00:00
7	0.660218	0.596379	0.813239	00:00
8	0.654463	0.558682	0.815603	00:00
9	0.637534	0.538710	0.815603	00:00

100.00% [14/14 00:00<00:00]

Epoch 10: early stopping

30.00% [6/20 00:01<00:03]

epoch	train_loss	valid_loss	accuracy	time
0	0.772114	0.681974	0.747045	00:00

Recommendation

Via Baysian Optimization, find the params that need to change to reduce the likelihood someone experience depression.

In [25]:

```
from src.recommender.JSONParamReader import JSONPa
import pprint
import warnings
from pandas.io.json import json
import pandas as pd
from bayes_opt import BayesianOptimization, Events
from fastai.basic_data import DataBunch, Tensor
import numpy as np
from src.classifier.CustomTabularModel import Cust
from src.data.DataCsvInterface import DataCsvInte
from src.recommender.JSONParamReader import JSONPa

warnings.simplefilter(action='ignore', category=Fu

bayesian_optimizer = BayesianRecommender()
# Init the model with the best params
model = CustomTabularModel(0.5, False, 1000, {'lay
best_params = JSONParamReader('classifier/logs').g
model.reset_params(best_params)
pp = pprint.PrettyPrinter(indent=4)
print("        Using Model Architecture")
pp.pprint( best_params)
```

```
        Using Model Architecture
{'dropout': 0.4170220047, 'layer1': 288.409472883
4}
```

In [26]:

```
data = model.input_data.train_ds[0][0]
data_parsed = []
for element in data.data:
    if type(element) is Tensor:
        data_parsed += list(element.numpy())
    else:
        data_parsed += element


data_init = {key: data_parsed[i] for i, key in enu
cr = bayesian_optimizer.get_ranges(data.names, moc
column_range = {key: cr[key] for key in cr if key
model.train(90)
```

4.44% [4/90 00:01<00:24]

epoch	train_loss	valid_loss	accuracy	time
0	0.675767	0.658606	0.846808	00:00
1	0.664987	0.645669	0.846808	00:00
2	0.654010	0.630003	0.846808	00:00
3	0.644451	0.604866	0.846808	00:00

100.00% [8/8 00:00<00:00]

Epoch 4: early stopping

0.8468084931373596

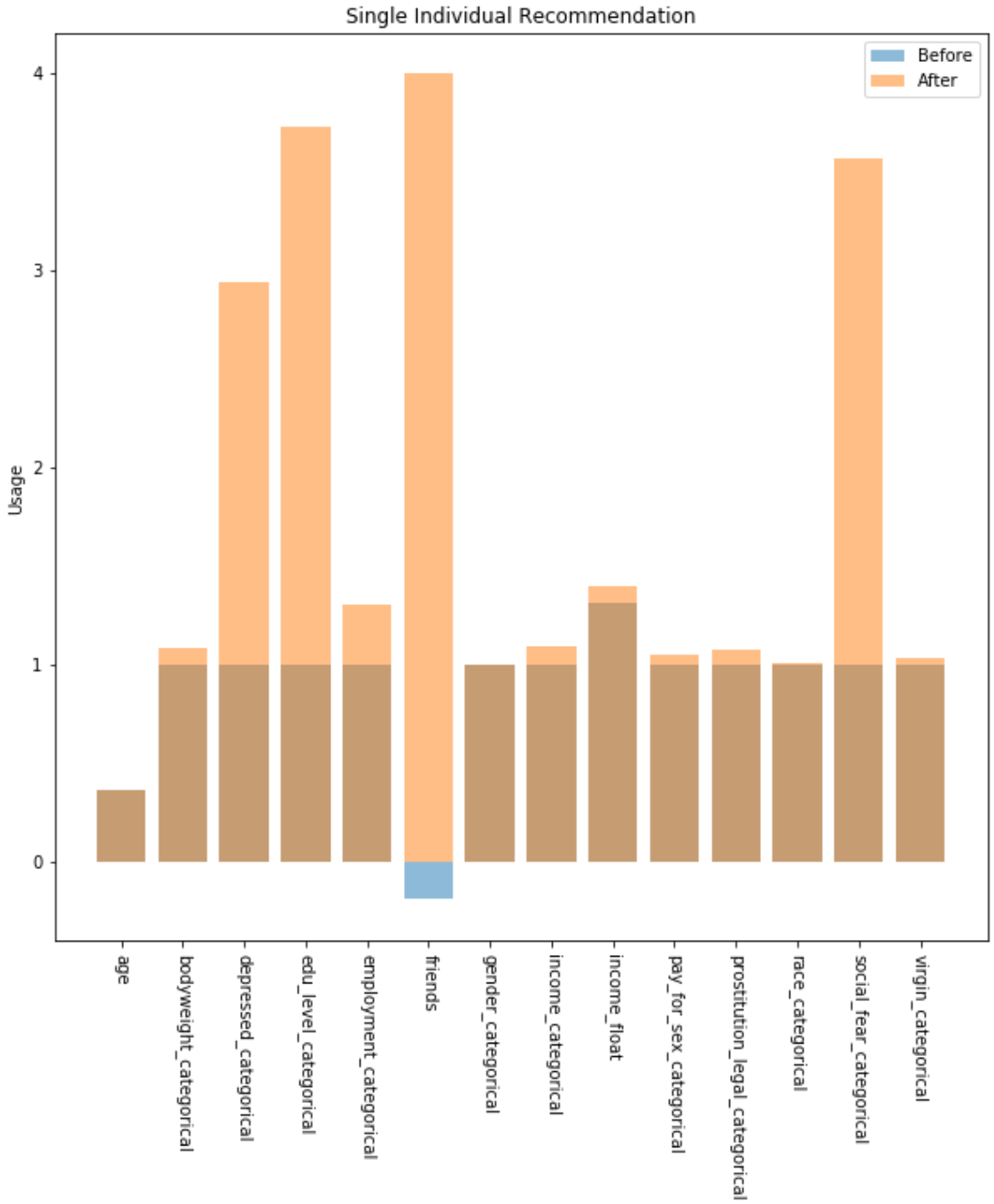
In [27]:
```
esian_optimizer.run_optimization(model, data_init, c


Value to maximize: 0.4999523162841797
Value to maximize: 0.5003073215484619
Value to maximize: 0.500303328037262
Value to maximize: 0.5007879734039307
Value to maximize: 0.5008745789527893
Value to maximize: 0.5008864402770996
Keeping results: {'target': 0.5008864402770996, 'p
arams': {'bodyweight_categorical': 1.0850132308450
688, 'depressed_categorical': 2.936165923199928 4,
'edu_level_categorical': 3.722764174374958, 'emplo
yment_categorical': 1.3035851241440028, 'friends':
3.9945746100373665, 'gender_categorical': 1.0, 'in
come_categorical': 1.0870035421055428, 'income_flo
at': 1.3967942175852635, 'pay_for_sex_categorica
l': 1.0466278587617235, 'prostitution_legal_catego
rical': 1.0768838754090093, 'race_categorical': 1.
0084664602164604, 'sexuallity_categorical': 1.6822
567688459231, 'social_fear_categorical': 3.5620199
16620608, 'virgin_categorical': 1.033908755094694
4}}
[{'target': 0.5008864402770996, 'params': {'bodywe
ight_categorical': 1.0850132308450688, 'depressed_
categorical': 2.936165923199928 4, 'edu_level_categ
orical': 3.722764174374958, 'employment_categorica
l': 1.3035851241440028, 'friends': 3.9945746100373
665, 'gender_categorical': 1.0, 'income_categorica
l': 1.0870035421055428, 'income_float': 1.3967942 1
75852635, 'pay_for_sex_categorical': 1.0466278587 6
17235, 'prostitution_legal_categorical': 1.0768838
754090093, 'race_categorical': 1.0084664602164604,
'sexuallity_categorical': 1.6822567688459231, 'soc
```

```
ial_fear_categorical': 3.562019916620608, 'virgin_
categorical': 1.0339087550946944}}]
```

In [64]:

```
after_data = bayesian_optimizer.results[0]['params

# Sort both dictionaries by keys
data_init = {i:data_init[i] for i in sorted(data_i
after_data = {i:after_data[i] for i in sorted([_ f
              if i != 1}

# Exclude changing your sexuality lol
del data_init['sexuallity_categorical']
del after_data['sexuallity_categorical']



# Before Data
feature_data = [data_init[_] for _ in data_init]
feature_pos = list(range(len(feature_data)))
objects = [_ for _ in data_init]
# Fill in any missing data for running predictions
for key in data_init:
    if key not in after_data:
        print("    "+ str(key))
        after_data[key] = data_init[key]

after_feature_data = [after_data[_] for _ in after
after_feature_pos = list(range(len(after_feature_d
after_objects = [_ for _ in after_data]

plt.figure(figsize=(10, 10))
b1 = plt.bar(feature_pos, feature_data, align='cen
b2 = plt.bar(after_feature_pos, after_feature_data
plt.xticks(feature_pos, objects, rotation=-90)
plt.ylabel('Usage')
```

```
plt.title('Single Individual Recommendation')
plt.legend((b1[0], b2[0]), ('Before', 'After'))
plt.show()
```



In [62]:
```
data_init
```

```
{'age': 0.36457014,
 'bodyweight_categorical': 1,
 'depressed_categorical': 1,
 'edu_level_categorical': 1,
 'employment_categorical': 1,
 'friends': -0.19225857,
 'gender_categorical': 1,
 'income_categorical': 1,
 'income_float': 1.3152382,
 'pay_for_sex_categorical': 1,
 'prostitution_legal_categorical': 1,
 'race_categorical': 1,
 'sexuallity_categorical': 1,
 'social_fear_categorical': 1,
 'virgin_categorical': 1}
```

In [78]:
```
changes = np.abs(np.subtract(feature_data, after_f
directions = np.sign(np.subtract(feature_data, aft
```

In [73]:
```
n = 3
max_categories = np.argsort(changes)[-n:][::-1]
```

In [ ]:
```
model.input_data.train_dl
```

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]: