Semi-Supervised Learning with Support Vector Machine (SVM)

In [1]:
```
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn import svm
from sklearn import datasets
import matplotlib.pyplot as plt
# pd.set_option('display.max_rows', None)
```

Preparing the dataset

In [2]:
```
#Load dataset
dataset = datasets.load_wine()
```

In [15]:
```
df = pd.DataFrame(dataset.data, columns=[dataset.feature_names])
df['label'] = pd.Series(dataset.target)
df = df.sample(frac=1).reset_index(drop=True) # shuffle the dataframe in-place and reset the index
df
```

Out[15]:

	alcohol	malic_acid	ash	alcalinity_of_ash	magnesium	total_phenols	flavanoids	nonflavanoid_
0	13.87	1.90	2.80	19.4	107.0	2.95	2.97	
1	13.34	0.94	2.36	17.0	110.0	2.53	1.30	
2	14.38	3.59	2.28	16.0	102.0	3.25	3.17	
3	13.07	1.50	2.10	15.5	98.0	2.40	2.64	
4	12.33	1.10	2.28	16.0	101.0	2.05	1.09	
...	...	...	...	...	...	...	...	
173	12.58	1.29	2.10	20.0	103.0	1.48	0.58	
174	12.82	3.37	2.30	19.5	88.0	1.48	0.66	
175	12.43	1.53	2.29	21.5	86.0	2.74	3.15	
176	12.07	2.16	2.17	21.0	85.0	2.60	2.65	
177	12.70	3.55	2.36	21.5	106.0	1.70	1.20	

178 rows × 14 columns

Create labeled dataset

In [16]:
```
#taking half of the dataset as labled data
X = df.iloc[0:89,0:13].values
y = df.iloc[0:89,-1].values
df.iloc[0:89,0:13]
```

Out[16]:

	alcohol	malic_acid	ash	alcalinity_of_ash	magnesium	total_phenols	flavanoids	nonflavanoid_p
0	13.87	1.90	2.80	19.4	107.0	2.95	2.97	
1	13.34	0.94	2.36	17.0	110.0	2.53	1.30	
2	14.38	3.59	2.28	16.0	102.0	3.25	3.17	
3	13.07	1.50	2.10	15.5	98.0	2.40	2.64	
4	12.33	1.10	2.28	16.0	101.0	2.05	1.09	
...	...	...	...	...	...	...	...	
84	12.33	0.99	1.95	14.8	136.0	1.90	1.85	
85	12.42	2.55	2.27	22.0	90.0	1.68	1.84	
86	13.16	2.36	2.67	18.6	101.0	2.80	3.24	
87	13.29	1.97	2.68	16.8	102.0	3.00	3.23	
88	13.20	1.78	2.14	11.2	100.0	2.65	2.76	

89 rows × 13 columns

In [17]:
```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.7,
random_state=1)
```

In [18]:
```
X_train.shape
```

Out[18]: (26, 13)

In [19]:
```
X_test.shape
```

Out[19]: (63, 13)

Create unlabeled dataset

In [20]:
```
# taking the other half of the data as unlabeled data
X_unl_df = df.iloc[89:,0:13].reset_index(drop=True)
```

```
X_unl = X_unl_df.values
X_unl_df
```

Out[20]:

	alcohol	malic_acid	ash	alcalinity_of_ash	magnesium	total_phenols	flavanoids	nonflavanoid_p
0	11.64	2.06	2.46	21.6	84.0	1.95	1.69	
1	14.22	3.99	2.51	13.2	128.0	3.00	3.04	
2	11.82	1.72	1.88	19.5	86.0	2.50	1.64	
3	14.19	1.59	2.48	16.5	108.0	3.30	3.93	
4	12.37	1.13	2.16	19.0	87.0	3.50	3.10	
...	...	...	...	...	...	...	...	
84	12.58	1.29	2.10	20.0	103.0	1.48	0.58	
85	12.82	3.37	2.30	19.5	88.0	1.48	0.66	
86	12.43	1.53	2.29	21.5	86.0	2.74	3.15	
87	12.07	2.16	2.17	21.0	85.0	2.60	2.65	
88	12.70	3.55	2.36	21.5	106.0	1.70	1.20	

89 rows × 13 columns

1. Training on the labeled dataset

In [21]:

```
clf = svm.SVC(kernel='linear', probability=True, C=1.0).fit(X_train,
y_train)
clf.score(X_test, y_test)
```

Out[21]: 0.7936507936507936

2. Make a prediction using the unlabeled datset (x_unl)

In [22]:

```
#find the probability of each class
clp= clf.predict_proba(X_unl)
clf_prob = pd.DataFrame(clp, columns = ['class1', 'class2','class3'])
# predict the the label of each class
lab=clf.predict(X_unl)
clf_prob["max"] = clf_prob.max(axis = 1)
clf_prob["lab"] = lab
clf_prob
```

Out[22]:

	class1	class2	class3	max	lab

	class1	class2	class3	max	lab
0	0.154002	0.314076	0.531922	0.531922	1
1	0.292137	0.270598	0.437265	0.437265	2
2	0.023514	0.379967	0.596519	0.596519	1
3	0.998248	0.001364	0.000388	0.998248	0
4	0.024358	0.378537	0.597105	0.597105	1
...	...	...	...	...	...
84	0.119894	0.324574	0.555532	0.555532	2
85	0.160462	0.305302	0.534236	0.534236	2
86	0.015080	0.382361	0.602559	0.602559	1
87	0.018101	0.381986	0.599914	0.599914	1
88	0.090178	0.339134	0.570689	0.570689	2

89 rows × 5 columns

3. Choose the samples in X_unl with high confidence and add them into the labeled dataset

In [23]:

```
th = 0.6
clf_prob[clf_prob["max"] > th]
```

Out[23]:

	class1	class2	class3	max	lab
3	0.998248	0.001364	0.000388	0.998248	0
5	0.014084	0.383017	0.602899	0.602899	1
6	0.970301	0.019355	0.010344	0.970301	0
8	0.994399	0.004148	0.001453	0.994399	0
9	0.937449	0.038556	0.023996	0.937449	0
11	0.994186	0.004298	0.001517	0.994186	0
12	0.991134	0.006408	0.002458	0.991134	0
16	0.009069	0.387745	0.603186	0.603186	1
17	0.819020	0.094391	0.086589	0.819020	0
19	0.611770	0.171319	0.216911	0.611770	0
20	0.842308	0.085591	0.072101	0.842308	0
21	0.715648	0.137932	0.146420	0.715648	0
22	0.034904	0.364447	0.600649	0.600649	2
27	0.968139	0.020916	0.010945	0.968139	0
29	0.018317	0.379884	0.601800	0.601800	1
31	0.913269	0.050632	0.036099	0.913269	0

	class1	class2	class3	max	lab
36	0.017267	0.382011	0.600722	0.600722	1
43	0.012550	0.384893	0.602557	0.602557	1
48	0.962569	0.024327	0.013104	0.962569	0
55	0.984796	0.010579	0.004625	0.984796	0
56	0.748896	0.126724	0.124380	0.748896	0
60	0.880995	0.066579	0.052426	0.880995	0
63	0.009867	0.386900	0.603233	0.603233	1
64	0.976518	0.015492	0.007990	0.976518	0
66	0.903556	0.056237	0.040207	0.903556	0
71	0.037796	0.359127	0.603077	0.603077	2
79	0.937871	0.037880	0.024249	0.937871	0
82	0.742474	0.126368	0.131158	0.742474	0
86	0.015080	0.382361	0.602559	0.602559	1

In [24]:

```
#add the predicted labels to the training dataset
unl_size =len(X_unl[clf_prob["max"] > th])
X_train_new = np.append(X_train, X_unl[clf_prob["max"] > th], axis=0)
y_train_new = np.append(y_train, clf_prob['lab'][clf_prob["max"] >
th].values, axis=0)

X_train = X_train_new
y_train = y_train_new
```

In [25]:

```
#remove the added labels from the unlabled dataset
X_unl_df = X_unl_df.drop(X_unl_df[clf_prob["max"] >
th].index).reset_index(drop=True)
#update the unlabeled set
X_unl = X_unl_df.values
# X_unl_df
```

4. Repeat

In [26]:

```
score_ls = []
while len(X_unl) != 0 and unl_size != 0: # stop when there are no more
unlabeled data or when we are no confident about the data
```

```
    #Step 1
    clf = svm.SVC(kernel='linear', probability=True,C=1).fit(X_train,
y_train)
    score_ls.append(clf.score(X_test, y_test))
    print ('Accuracy: ',clf.score(X_test, y_test))
#     print(len(X_unl))

    #Step2
    #find the probability of each class
    clp= clf.predict_proba(X_unl)
    clf_prob = pd.DataFrame(clp, columns = ['class1', 'class2','class3'])
    # predict the the label of each class
    lab=clf.predict(X_unl)
    clf_prob["max"] = clf_prob.max(axis = 1)
    clf_prob["lab"] = lab

    #Step3
    unl_size =len(X_unl[clf_prob["max"] > th])
    X_train_new = np.append(X_train, X_unl[clf_prob["max"] > th], axis=0)
    y_train_new = np.append(y_train, clf_prob['lab'][clf_prob["max"] >
th].values, axis=0)
    X_train = X_train_new
    y_train = y_train_new



    X_unl_df = X_unl_df.drop(X_unl_df[clf_prob["max"] >
th].index).reset_index(drop=True)
    X_unl = X_unl_df.values
```

```
Accuracy:  0.7936507936507936
Accuracy:  0.8095238095238095
Accuracy:  0.873015873015873
Accuracy:  0.873015873015873
```

In [ ]:

```
```

In [ ]:

```
```

In [1]:

```
import numpy as np
import plotly.express as px
import pandas as pd
import plotly.graph_objects as go
import seaborn as sns
import matplotlib.pyplot as plt
import math
import scipy as sp
from sklearn.model_selection import train_test_split
from sklearn import svm
from sklearn import datasets
from sklearn.metrics import zero_one_loss
```

Generating 2D training dataset with 2 classes¶

In [35]:

```
mu_1=[-2,0]
mu_2 = [0,2]
cov_1 = [[1, -0.25],[-0.25, 1]]
cov_2 = [[1, -0.25],[-0.25, 1]]
num_samples= 500

w1 =  np.random.multivariate_normal(mu_1, cov_1, num_samples)
w2 =  np.random.multivariate_normal(mu_2, cov_2, num_samples)

label1 = np.zeros(num_samples).reshape((num_samples,1))
label2 = np.zeros(num_samples).reshape((num_samples,1)) +1

w1 = np.append(w1, label1, axis=1)
w2 = np.append(w2, label2, axis=1)



#prior probability is the same for each class
prior_p=500/1000
#concatenate whole samples in one array
train_dataset=np.concatenate([w1,w2])
```

```
df = pd.DataFrame(train_dataset, columns=['x1', 'x2', 'label'])
df = df.sample(frac=1).reset_index(drop=True) # shuffle the dataframe in-place and reset the index
df
```

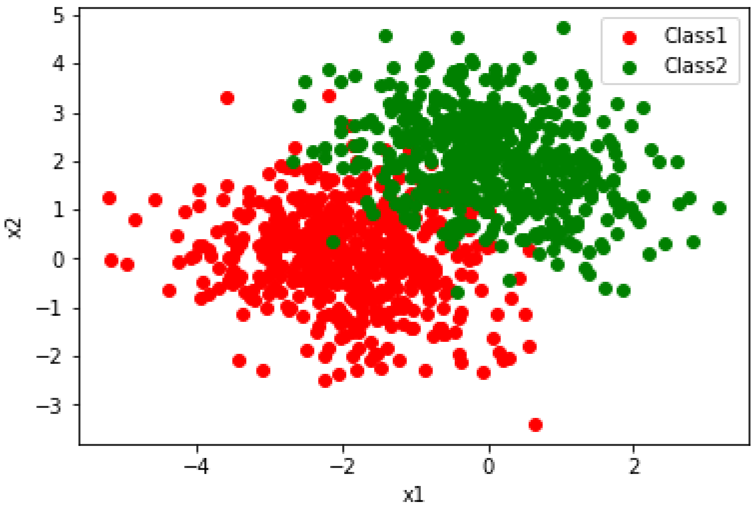
Out[35]:

	x1	x2	label
0	-0.380980	3.884836	1.0
1	0.270876	1.778541	1.0
2	-2.453738	1.025094	0.0
3	-2.438149	-0.028350	0.0
4	0.424334	1.802717	1.0
...	...	...	...
995	-2.945962	0.282230	0.0
996	-0.852222	4.062818	1.0
997	-2.411784	0.366154	0.0
998	-0.780021	1.435980	1.0
999	0.020570	2.387932	1.0

1000 rows × 3 columns

In [36]:

```
figure1 = plt.figure()
plt.scatter(w1[:,0], w1[:,1], color='r', label='Class1')
plt.scatter(w2[:,0], w2[:,1], color='g', label='Class2')
# plt.rcParams['figure.figsize'] = [20, 20]
plt.xlabel('x1')
plt.ylabel('x2')
plt.legend()
# plt.grid()
plt.show()
```



In [37]:
```
df
X = df.iloc[:,0:2].values
y = df.iloc[:,-1].values
```

Creating training and testing datasets with 80:20 split

In [38]:
```
X_train, X_test, y_train, y_test = train_test_split(X, y, train_size=0.8,
random_state=1)
```

In [39]:
```
X_train.shape
```

Out[39]: (800, 2)

Part A: 10% of the training data is labeled

In [40]:
```
#10% of the training data is labeled
X_train, X_unl, y_train, _ = train_test_split(X_train, y_train,
train_size=0.1, random_state=1)
```

In [41]:
```
X_unl.shape
```

Out[41]: (720, 2)

In [42]:
```
X_unl_df = pd.DataFrame(X_unl, columns=['x1', 'x2'])
X_unl_df
```

Out[42]:

	x1	x2

	x1	x2
0	-0.162472	1.843025
1	-2.202430	-1.832835
2	-0.392301	-1.230569
3	-3.410816	-0.358632
4	-0.495970	1.463370
...	...	...
715	1.142499	0.650258
716	-0.509620	1.222140
717	1.309253	3.330639
718	-4.041126	0.053144
719	-1.401420	0.298887

720 rows × 2 columns

1. Training on the labeled dataset

In [43]:

```
clf = svm.SVC(kernel='linear', probability=True, C=1.0).fit(X_train,
y_train)
print ('Accuracy: ',clf.score(X_test, y_test), ' error: ', 1 -
clf.score(X_test, y_test) )
```

```
Accuracy:  0.97  error:  0.030000000000000027
```

2. Make a prediction using the unlabeled datset (x_unl)

- ### Using predict_prob() to find the probability of each sample

In [44]:

```
#find the probability of each class
clp= clf.predict_proba(X_unl)
clf_prob = pd.DataFrame(clp, columns = ['class1', 'class2'])
# predict the the label of each class
lab=clf.predict(X_unl)
#find the max probability
clf_prob["max"] = clf_prob.max(axis = 1)
clf_prob["label"] = lab
clf_prob
```

Out[44]:

	class1	class2	max	label

	class1	class2	max	label
0	6.080655e-02	0.939193	0.939193	1.0
1	9.984301e-01	0.001570	0.998430	0.0
2	9.049883e-01	0.095012	0.904988	0.0
3	9.984551e-01	0.001545	0.998455	0.0
4	1.732832e-01	0.826717	0.826717	1.0
...	...	...	...	...
715	3.397656e-02	0.966023	0.966023	1.0
716	2.353276e-01	0.764672	0.764672	1.0
717	6.985512e-07	0.999999	0.999999	1.0
718	9.990992e-01	0.000901	0.999099	0.0
719	8.610110e-01	0.138989	0.861011	0.0

720 rows × 4 columns

3. Choose the samples in X_unl with high confidence and add them into the labeled dataset

In [45]:

```
th = 0.6
clf_prob[clf_prob["max"] > th]
```

Out[45]:

	class1	class2	max	label
0	6.080655e-02	0.939193	0.939193	1.0
1	9.984301e-01	0.001570	0.998430	0.0
2	9.049883e-01	0.095012	0.904988	0.0
3	9.984551e-01	0.001545	0.998455	0.0
4	1.732832e-01	0.826717	0.826717	1.0
...	...	...	...	...
715	3.397656e-02	0.966023	0.966023	1.0
716	2.353276e-01	0.764672	0.764672	1.0
717	6.985512e-07	0.999999	0.999999	1.0
718	9.990992e-01	0.000901	0.999099	0.0
719	8.610110e-01	0.138989	0.861011	0.0

679 rows × 4 columns

In [46]:

```
pseudo_lab_size =len(X_unl[clf_prob["max"] > th])
pseudo_lab_size
```

Out[46]: 679

In [47]:
```
#add the predicted labels to the training dataset
X_train_new = np.append(X_train, X_unl[clf_prob["max"] > th], axis=0)
y_train_new = np.append(y_train, clf_prob['label'][clf_prob["max"] >
th].values, axis=0)

X_train = X_train_new
y_train = y_train_new
```

In [48]:
```
#remove the added labels from the unlabled dataset
X_unl_df = X_unl_df.drop(X_unl_df[clf_prob["max"] >
th].index).reset_index(drop=True)
#update the unlabeled set
X_unl = X_unl_df.values
# X_unl_df
```

4. Repeat

In [49]:
```
score_ls = []
while len(X_unl) != 0 and pseudo_lab_size != 0: # stop when there are no
more unlabeled data or when we are no confident about the data
    #Step 1
    clf = svm.SVC(kernel='linear', probability=True,C=1).fit(X_train,
y_train)
    score_ls.append(clf.score(X_test, y_test))
    print ('Accuracy: ',clf.score(X_test, y_test), ' error: ', 1 -
clf.score(X_test, y_test) )
#     print(len(X_unl))

    #Step2
    #find the probability of each class
    clp= clf.predict_proba(X_unl)
    clf_prob = pd.DataFrame(clp, columns = ['class1', 'class2'])
    # predict the the label of each class
    lab=clf.predict(X_unl)
```

```
    clf_prob["max"] = clf_prob.max(axis = 1)
    clf_prob["label"] = lab

    #Step3
    pseudo_lab_size =len(X_unl[clf_prob["max"] > th])
    X_train_new = np.append(X_train, X_unl[clf_prob["max"] > th], axis=0)
    y_train_new = np.append(y_train, clf_prob['label'][clf_prob["max"] >
th].values, axis=0)
    X_train = X_train_new
    y_train = y_train_new



    X_unl_df = X_unl_df.drop(X_unl_df[clf_prob["max"] >
th].index).reset_index(drop=True)
    X_unl = X_unl_df.values
```

```
Accuracy:  0.97  error:  0.030000000000000027
Accuracy:  0.975  error:  0.025000000000000022
Accuracy:  0.975  error:  0.025000000000000022
```

When 10% of the training data is labeled, the error of the self-training algorithm on the testing data with SVM classifier is as follows:

1. The first time the classifier is trained using only the labeled -----> accuracy: 0.97 error: 0.030000000000000027
2. At least one time point during the self training process -----> accuracy: 0.97 error: 0.030000000000000027
3. After self-training is completed -----> accuracy: 0.975 error: 0.025000000000000022

From the above results, we can see that there is no significant boost in performance. This confirms the lecture's discussion that SSL is not always going to work or give you a significant boost in performance.

In [ ]:

```

```

Part B: 25% of the training data is labeled

In [50]:

```
X_train, X_test, y_train, y_test = train_test_split(X, y, train_size=0.8,
random_state=1)
#25% of the training data is labeled
```

```
X_train, X_unl, y_train, _ = train_test_split(X_train, y_train,
train_size=0.25, random_state=1)
```

In [53]:

```
X_train.shape
```

Out[53]:

```
(200, 2)
```

In [54]:

```
X_unl_df = pd.DataFrame(X_unl, columns=['x1', 'x2'])
X_unl_df
```

Out[54]:

	x1	x2
0	-0.162472	1.843025
1	-2.202430	-1.832835
2	-0.392301	-1.230569
3	-3.410816	-0.358632
4	-0.495970	1.463370
...	...	...
595	0.320025	-0.795005
596	1.238727	2.093136
597	-0.269999	0.602040
598	-1.398507	0.951005
599	-2.549455	0.151460

600 rows × 2 columns

1. Training on the labeled dataset

In [55]:

```
clf = svm.SVC(kernel='linear', probability=True, C=1.0).fit(X_train,
y_train)
print ('Accuracy: ',clf.score(X_test, y_test), ' error: ', 1 -
clf.score(X_test, y_test) )
```

```
Accuracy:  0.965  error:  0.035000000000000003
```

2. Make a prediction using the unlabeled datset (x_unl)

- ### Using predict_prob() to find the probability of each sample

In [56]:

```
#find the probability of each class
clp= clf.predict_proba(X_unl)
```

```python
clf_prob = pd.DataFrame(clp, columns = ['class1', 'class2'])
# predict the the label of each class
lab=clf.predict(X_unl)
#find the max probability
clf_prob["max"] = clf_prob.max(axis = 1)
clf_prob["label"] = lab
clf_prob
```

Out[56]:

	class1	class2	max	label
0	0.029395	0.970605	0.970605	1.0
1	0.999555	0.000445	0.999555	0.0
2	0.931709	0.068291	0.931709	0.0
3	0.999550	0.000450	0.999550	0.0
4	0.112255	0.887745	0.887745	1.0
...	...	...	...	...
595	0.560227	0.439773	0.560227	0.0
596	0.000002	0.999998	0.999998	1.0
597	0.269522	0.730478	0.730478	1.0
598	0.703817	0.296183	0.703817	0.0
599	0.992334	0.007666	0.992334	0.0

600 rows × 4 columns

3. Choose the samples in X_unl with high confidence and add them into the labeled dataset

In [57]:

```python
th = 0.6
clf_prob[clf_prob["max"] > th]
```

Out[57]:

	class1	class2	max	label
0	0.029395	0.970605	0.970605	1.0
1	0.999555	0.000445	0.999555	0.0
2	0.931709	0.068291	0.931709	0.0
3	0.999550	0.000450	0.999550	0.0
4	0.112255	0.887745	0.887745	1.0
...	...	...	...	...
594	0.005475	0.994525	0.994525	1.0
596	0.000002	0.999998	0.999998	1.0

	class1	class2	max	label
597	0.269522	0.730478	0.730478	1.0
598	0.703817	0.296183	0.703817	0.0
599	0.992334	0.007666	0.992334	0.0

569 rows × 4 columns

In [58]:

```
pseudo_lab_size =len(X_unl[clf_prob["max"] > th])
#add the predicted labels to the training dataset
X_train_new = np.append(X_train, X_unl[clf_prob["max"] > th], axis=0)
y_train_new = np.append(y_train, clf_prob['label'][clf_prob["max"] >
th].values, axis=0)

X_train = X_train_new
y_train = y_train_new
#remove the added labels from the unlabled dataset
X_unl_df = X_unl_df.drop(X_unl_df[clf_prob["max"] >
th].index).reset_index(drop=True)
#update the unlabeled set
X_unl = X_unl_df.values
# X_unl_df
```

4. Repeat

In [59]:

```
score_ls = []
while len(X_unl) != 0 and pseudo_lab_size != 0: # stop when there are no
more unlabeled data or when we are no confident about the data
    #Step 1
    clf = svm.SVC(kernel='linear', probability=True,C=1).fit(X_train,
y_train)
    score_ls.append(clf.score(X_test, y_test))
    print ('Accuracy: ',clf.score(X_test, y_test), ' error: ', 1 -
clf.score(X_test, y_test) )
#     print(len(X_unl))

    #Step2
    #find the probability of each class
```

```
    clp= clf.predict_proba(X_unl)
    clf_prob = pd.DataFrame(clp, columns = ['class1', 'class2'])
    # predict the the label of each class
    lab=clf.predict(X_unl)
    clf_prob["max"] = clf_prob.max(axis = 1)
    clf_prob["label"] = lab

    #Step3
    pseudo_lab_size =len(X_unl[clf_prob["max"] > th])
    X_train_new = np.append(X_train, X_unl[clf_prob["max"] > th], axis=0)
    y_train_new = np.append(y_train, clf_prob['label'][clf_prob["max"] >
th].values, axis=0)
    X_train = X_train_new
    y_train = y_train_new



    X_unl_df = X_unl_df.drop(X_unl_df[clf_prob["max"] >
th].index).reset_index(drop=True)
    X_unl = X_unl_df.values
```

```
Accuracy:  0.97  error:  0.030000000000000027
Accuracy:  0.975  error:  0.025000000000000022
Accuracy:  0.975  error:  0.025000000000000022
```

When 25% of the training data is labeled, the error of the self-training algorithm on the testing data with SVM classifier is as follows:¶

1. The first time the classifier is trained using only the labeled -----> accuracy: 0.965 error: 0.03500000000000003
2. At least one time point during the self training process -----> accuracy: 0.97 error: 0.030000000000000027
3. After self-training is completed -----> accuracy: 0.975 error: 0.025000000000000022

From the above results, we can see similar trend to the previous results. That is there is no significant boost in performance.

In [ ]:

In [ ]:

In [1]:

```
import numpy as np
import plotly.express as px
import pandas as pd
import plotly.graph_objects as go
import seaborn as sns
import matplotlib.pyplot as plt
import math
import scipy as sp
from sklearn.model_selection import train_test_split
from sklearn import svm
from sklearn import datasets
from sklearn.metrics import zero_one_loss
from sklearn.model_selection import KFold
```

In [83]:

```
class SelftrainKfold:
    def __init__(self):
        self.error = []
        pass
    def readfile(self, path):
        self.raw_data = pd.read_csv(path, header=None)
        self.raw_data = self.raw_data.sample(frac=1).reset_index(drop=True)
# shuffle the dataframe in-place and reset the index
        print(self.raw_data)
    def splitdata(self):
        self.X = self.raw_data.iloc[:,:-1] #get all columns except the last
one
        self.y = self.raw_data.iloc[:,-1]
        self.X = self.X.to_numpy()
        self.y = self.y.to_numpy()

    def fitwithKfold(self):
        self.kfold = KFold(n_splits= 5, shuffle= True, random_state = 1)
        i=1
        for train_index, test_index in self.kfold.split(self.X):
        #     print("TRAIN:", train_index, "TEST:", test_index)
```

```
            X_train, X_test =  self.X[train_index],  self.X[test_index]
            y_train, y_test =  self.y[train_index],  self.y[test_index]

            ## 15% of the training data is labeled
            X_train_lab, X_unl, y_train_lab, _ = train_test_split(X_train,
y_train, train_size=0.15, random_state=1)
            # 1. Training on the labeled dataset
            print("Fold_{}".format(i))
            while True:
                #Step1
                clf = svm.SVC(kernel='linear',
probability=True,C=1).fit(X_train_lab, y_train_lab)
                print ('Accuracy: ',clf.score(X_test, y_test), ' error: ',
1 - clf.score(X_test, y_test) )
                if (len(X_unl) == 0): # exit the training if no more
unlabeled data
        #            print("fist break")
                    break
                #Step2
                clp= clf.predict_proba(X_unl)
                clf_prob = pd.DataFrame(clp)
                lab=clf.predict(X_unl)
                clf_prob["max"] = clf_prob.max(axis = 1)
                clf_prob["label"] = lab
            #     print (clf_prob)
                th = 0.8
                if len(X_unl[clf_prob["max"] > th]) == 0: #exit if no more
samples meets the threshold condition
                    break
                #Step3
                #add the predicted labels to the training dataset
                X_train_new = np.append(X_train_lab, X_unl[clf_prob["max"]
> th], axis=0)
                y_train_new = np.append(y_train_lab, clf_prob['label']
[clf_prob["max"] > th].values, axis=0)
                X_train_lab = X_train_new
```

```
                    y_train_lab = y_train_new

                    #remove the added labels from the unlabled dataset
                    X_unl_df = pd.DataFrame(X_unl)
                    X_unl_df = X_unl_df.drop(X_unl_df[clf_prob["max"] >
th].index).reset_index(drop=True)
                    #update the unlabeled set
                    X_unl = X_unl_df.values

               print("Fold_{}_done".format(i))
               self.error.append(clf.score(X_test, y_test))
               i+=1
     def ave_error(self):
          return sum(self.error)/len(self.error)
```

In [84]:

```
obj = SelftrainKfold()
```

In [85]:

```
obj.readfile("/home/safwan/Documents/spring2021/ece523/hw/hw5/breast-
cancer.csv")
```

```
            0         1         2         3         4        5         6  \
0   -0.656576  0.912871  0.531300 -0.455814  0.508429 -1.42093 -0.937281
1    0.331744 -0.912871  2.430700 -0.455814  0.508429 -1.42093  1.063180
2   -1.644900  0.912871  1.481000  0.413271  0.508429  1.28831  1.063180
3    0.331744 -0.912871  0.531300  0.413271  0.508429  1.28831 -0.937281
4    0.331744 -0.912871 -0.893249 -0.455814  0.508429 -0.06631  1.063180
..        ...       ...       ...       ...       ...      ...       ...
281  0.331744 -0.912871 -0.418399 -0.455814  0.508429 -1.42093  1.063180
282 -0.656576  0.912871  1.006150 -0.455814 -1.511170  1.28831  1.063180
283  1.320060 -0.912871 -1.368100 -0.455814  0.508429 -0.06631  1.063180
284 -0.656576  0.912871  0.056451 -0.455814  0.508429 -0.06631 -0.937281
285 -0.656576  0.912871  0.056451 -0.455814  0.508429 -0.06631  1.063180

            7         8  9
0   -0.130877  0.557527  0
1    0.700918  0.557527  0
2    0.700918 -1.787360  0
3   -0.130877  0.557527  1
4    0.700918  0.557527  0
..        ...       ... ..
281 -0.130877  0.557527  0
282 -0.130877 -1.787360  0
283 -0.962672 -1.787360  0
284 -0.962672 -1.787360  0
285 -0.130877  0.557527  1

[286 rows x 10 columns]
```

In [86]:
```
obj.splitdata()
```

In [87]:
```
obj.fitwithKfold()
```

```
Fold_1
Accuracy:  0.7758620689655172  error:  0.22413793103448276
Accuracy:  0.7758620689655172  error:  0.22413793103448276
Accuracy:  0.7758620689655172  error:  0.22413793103448276
Accuracy:  0.7758620689655172  error:  0.22413793103448276
Accuracy:  0.7413793103448276  error:  0.2586206896551724
Accuracy:  0.7758620689655172  error:  0.22413793103448276
Accuracy:  0.7758620689655172  error:  0.22413793103448276
Accuracy:  0.7758620689655172  error:  0.22413793103448276
Accuracy:  0.7586206896551724  error:  0.24137931034482762
Accuracy:  0.7586206896551724  error:  0.24137931034482762
Fold_1_done
Fold_2
Accuracy:  0.5087719298245614  error:  0.49122807017543857
Accuracy:  0.5087719298245614  error:  0.49122807017543857
Accuracy:  0.5087719298245614  error:  0.49122807017543857
Accuracy:  0.5263157894736842  error:  0.4736842105263158
Accuracy:  0.5263157894736842  error:  0.4736842105263158
Accuracy:  0.5263157894736842  error:  0.4736842105263158
Fold_2_done
Fold_3
Accuracy:  0.8070175438596491  error:  0.19298245614035092
Accuracy:  0.7894736842105263  error:  0.21052631578947367
Accuracy:  0.7894736842105263  error:  0.21052631578947367
Fold_3_done
Fold_4
Accuracy:  0.6666666666666666  error:  0.33333333333333337
Accuracy:  0.6666666666666666  error:  0.33333333333333337
Accuracy:  0.6842105263157895  error:  0.3157894736842105
Accuracy:  0.6842105263157895  error:  0.3157894736842105
Accuracy:  0.7192982456140351  error:  0.2807017543859649
Accuracy:  0.7017543859649122  error:  0.29824561403508776
Accuracy:  0.6491228070175439  error:  0.3508771929824561
Accuracy:  0.7368421052631579  error:  0.26315789473684215
Accuracy:  0.7368421052631579  error:  0.26315789473684215
Fold_4_done
Fold_5
Accuracy:  0.47368421052631576  error:  0.5263157894736843
Accuracy:  0.47368421052631576  error:  0.5263157894736843
Accuracy:  0.47368421052631576  error:  0.5263157894736843
Accuracy:  0.47368421052631576  error:  0.5263157894736843
Accuracy:  0.47368421052631576  error:  0.5263157894736843
Accuracy:  0.543859649122807  error:  0.45614035087719296
Accuracy:  0.631578947368421  error:  0.368421052631579
Accuracy:  0.6491228070175439  error:  0.3508771929824561
Accuracy:  0.631578947368421  error:  0.368421052631579
Accuracy:  0.631578947368421  error:  0.368421052631579
Accuracy:  0.6491228070175439  error:  0.3508771929824561
Accuracy:  0.631578947368421  error:  0.368421052631579
Fold_5_done
```

In [ ]:

In [ ]:

In [88]:
```
obj.ave_error()
```

Out[88]:
```
0.6885662431941924
```

For the results above, we can see that the self training did help in 3 out of 5 folds and the boost in performance was not significat. However, we can also observe that in some cases the self training algorithm gave worst accuracy compared to the previous iteration.

In [90]:
```
# Using different dataset
obj1 = SelftrainKfold()
obj1.readfile('/home/safwan/Documents/spring2021/ece523/hw/hw5/abalone.csv')
```

```
             0         1         2         3         4         5         6  \
0     0.053792  0.882716  0.928243  0.489721  0.797852  0.590786  1.089330
1     1.261790 -1.781890 -1.842820 -1.542460 -1.373890 -1.326200 -1.309990
2     1.261790 -1.823520 -1.893200 -1.662000 -1.402440 -1.339720 -1.364730
3     0.053792  0.466371  0.525180  0.250642  0.584756  0.892638  0.409677
4    -1.154210  1.049250  1.079390  1.087420  1.112910  1.759900 -0.000854
...        ...       ...       ...       ...       ...       ...       ...
4172 -1.154210  1.299060  1.331310  1.206960  1.619650  1.608970  1.581970
4173 -1.154210  1.757040  1.784750  1.087420  2.641290  2.669960  3.370060
4174  0.053792  0.882716  0.877860  1.446040  1.216910  1.349920  1.417760
4175  1.261790 -0.324683 -0.230565 -0.227518 -0.531705 -0.490476 -0.023661
4176 -1.154210  1.299060  1.331310  1.326500  1.944900  0.888133  1.303720

             7  8
0     0.762695  2
1    -1.453500  0
2    -1.449910  0
3     0.367587  0
4     0.935105  2
...        ... ..
4172  1.671440  2
4173  1.876180  2
4174  1.014130  2
4175 -0.609405  0
4176  1.362540  2

[4177 rows x 9 columns]
```

In [91]:
```
obj1.splitdata()
obj1.fitwithKfold()
```

```
Fold_1
Accuracy:  0.645933014354067  error:  0.354066985645933
Accuracy:  0.645933014354067  error:  0.354066985645933
Accuracy:  0.645933014354067  error:  0.354066985645933
Accuracy:  0.6423444976076556  error:  0.35765550239234445
Accuracy:  0.6495215311004785  error:  0.3504784688995215
Accuracy:  0.6435406698564593  error:  0.3564593301435407
Accuracy:  0.6399521531100478  error:  0.36004784688995217
Accuracy:  0.638755980861244  error:  0.36124401913875603
Accuracy:  0.6399521531100478  error:  0.36004784688995217
Accuracy:  0.6351674641148325  error:  0.3648325358851675
```

```
Accuracy:  0.6339712918660287  error:  0.36602870813397126
Accuracy:  0.6339712918660287  error:  0.36602870813397126
Accuracy:  0.6351674641148325  error:  0.3648325358851675
Fold_1_done
Fold_2
Accuracy:  0.6435406698564593  error:  0.3564593301435407
Accuracy:  0.6435406698564593  error:  0.3564593301435407
Accuracy:  0.6447368421052632  error:  0.35526315789473684
Accuracy:  0.6507177033492823  error:  0.34928229665071775
Accuracy:  0.6363636363636364  error:  0.36363636363636365
Accuracy:  0.6196172248803827  error:  0.38038277511961727
Accuracy:  0.6196172248803827  error:  0.38038277511961727
Accuracy:  0.6172248803827751  error:  0.3827751196172249
Accuracy:  0.618421052631579  error:  0.381578947368421
Accuracy:  0.6172248803827751  error:  0.3827751196172249
Accuracy:  0.6136363636363636  error:  0.38636363636363635
Accuracy:  0.6100478468899522  error:  0.38995215311004783
Accuracy:  0.6100478468899522  error:  0.38995215311004783
Accuracy:  0.6088516746411483  error:  0.3911483253588517
Accuracy:  0.611244019138756  error:  0.38875598086124397
Accuracy:  0.6064593301435407  error:  0.3935406698564593
Accuracy:  0.6052631578947368  error:  0.39473684210526316
Accuracy:  0.6028708133971292  error:  0.3971291866028708
Accuracy:  0.6016746411483254  error:  0.39832535885167464
Accuracy:  0.6016746411483254  error:  0.39832535885167464
Accuracy:  0.5992822966507177  error:  0.40071770334928225
Accuracy:  0.5992822966507177  error:  0.40071770334928225
Accuracy:  0.5980861244019139  error:  0.4019138755980861
Fold_2_done
Fold_3
Accuracy:  0.6347305389221557  error:  0.3652694610778443
Accuracy:  0.6347305389221557  error:  0.3652694610778443
Accuracy:  0.6215568862275449  error:  0.3784431137724551
Accuracy:  0.6239520958083832  error:  0.3760479041916168
Accuracy:  0.6107784431137725  error:  0.3892215568862275
Accuracy:  0.6023952095808384  error:  0.39760479041916164
Accuracy:  0.6071856287425149  error:  0.3928143712574851
Accuracy:  0.6059880239520958  error:  0.39401197604790417
Accuracy:  0.6059880239520958  error:  0.39401197604790417
Accuracy:  0.6071856287425149  error:  0.3928143712574851
Accuracy:  0.6035928143712574  error:  0.39640718562874255
Accuracy:  0.6047904191616766  error:  0.39520958083832336
Accuracy:  0.6047904191616766  error:  0.39520958083832336
Fold_3_done
Fold_4
Accuracy:  0.629940119760479  error:  0.37005988023952097
Accuracy:  0.629940119760479  error:  0.37005988023952097
Accuracy:  0.6347305389221557  error:  0.3652694610778443
Accuracy:  0.6275449101796408  error:  0.37245508982035924
Accuracy:  0.6179640718562874  error:  0.38203592814371257
Accuracy:  0.6095808383233533  error:  0.3904191616766467
Accuracy:  0.6  error:  0.4
Accuracy:  0.5964071856287425  error:  0.4035928143712575
Accuracy:  0.5976047904191617  error:  0.4023952095808383
Accuracy:  0.5964071856287425  error:  0.4035928143712575
Accuracy:  0.5976047904191617  error:  0.4023952095808383
Accuracy:  0.5952095808383233  error:  0.4047904191616767
Accuracy:  0.5964071856287425  error:  0.4035928143712575
Accuracy:  0.5964071856287425  error:  0.4035928143712575
Accuracy:  0.5976047904191617  error:  0.4023952095808383
Accuracy:  0.5976047904191617  error:  0.4023952095808383
Fold_4_done
Fold_5
Accuracy:  0.6491017964071857  error:  0.3508982035928143
Accuracy:  0.6491017964071857  error:  0.3508982035928143
```

```
Accuracy:  0.644311377245509  error:  0.355688622754491
Accuracy:  0.6431137724550898  error:  0.3568862275449102
Accuracy:  0.6479041916167665  error:  0.3520958083832335
Accuracy:  0.6479041916167665  error:  0.3520958083832335
Accuracy:  0.6419161676646706  error:  0.35808383233532937
Accuracy:  0.6383233532934132  error:  0.36167664670658684
Accuracy:  0.6407185628742516  error:  0.35928143712574845
Accuracy:  0.6383233532934132  error:  0.36167664670658684
Accuracy:  0.6395209580838324  error:  0.36047904191616764
Accuracy:  0.6407185628742516  error:  0.35928143712574845
Accuracy:  0.6407185628742516  error:  0.35928143712574845
Fold_5_done
```

In [93]:

```
obj1.ave_error()
```

Out[93]:

```
0.6152734721943673
```

We see similar observation to the previous resutls

In [ ]:

```

```