Final project

Shalymova Aigerim

Dataset

	Attribute1	Attribute2	Attribute3	Attribute4	Attribute5	Attribute6	\
0	80	102	102	79	76	102	
1	76	102	102	79	76	102	
2	80	98	106	79	76	94	
3	76	94	102	76	76	94	
4	76	94	102	76	76	94	
...	...	...	...	...	...	...	
6430	56	64	108	96	64	71	
6431	64	71	108	96	68	75	
6432	68	75	108	96	71	87	
6433	71	87	108	88	71	91	
6434	71	91	100	81	76	95	

	Attribute7	Attribute8	Attribute9	Attribute10	...	Attribute28	\
0	102	79	76	102	...	87	
1	106	83	76	102	...	87	
2	102	76	76	94	...	79	
3	102	76	76	94	...	79	
4	102	76	76	89	...	75	
...	...	...	...	...	...	...	
6430	108	96	68	75	...	92	
6431	108	96	71	87	...	96	
6432	108	88	71	91	...	89	
6433	100	81	76	95	...	89	
6434	108	88	80	95	...	85	

Dataset contains of 6435 rows and 37 columns. There are 36 features and 6 classes.

Data

```
import pandas as pd
import numpy as np

df = pd.read_csv('/Users/asik/Downloads/satellite.csv')
# check NA values
df.isnull().sum()
```

```
Attribute1     0
Attribute2     0
Attribute3     0
Attribute4     0
Attribute5     0
Attribute6     0
Attribute7     0
Attribute8     0
Attribute9     0
Attribute10    0
Attribute11    0
Attribute12    0
Attribute13    0
Attribute14    0
Attribute15    0
Attribute16    0
Attribute17    0
Attribute18    0
Attribute19    0
Attribute20    0
Attribute21    0
Attribute22    0
Attribute23    0
Attribute24    0
Attribute25    0
Attribute26    0
Attribute27    0
Attribute28    0
Attribute29    0
Attribute30    0
Attribute31    0
Attribute32    0
```

There were no empty values

Neural Network

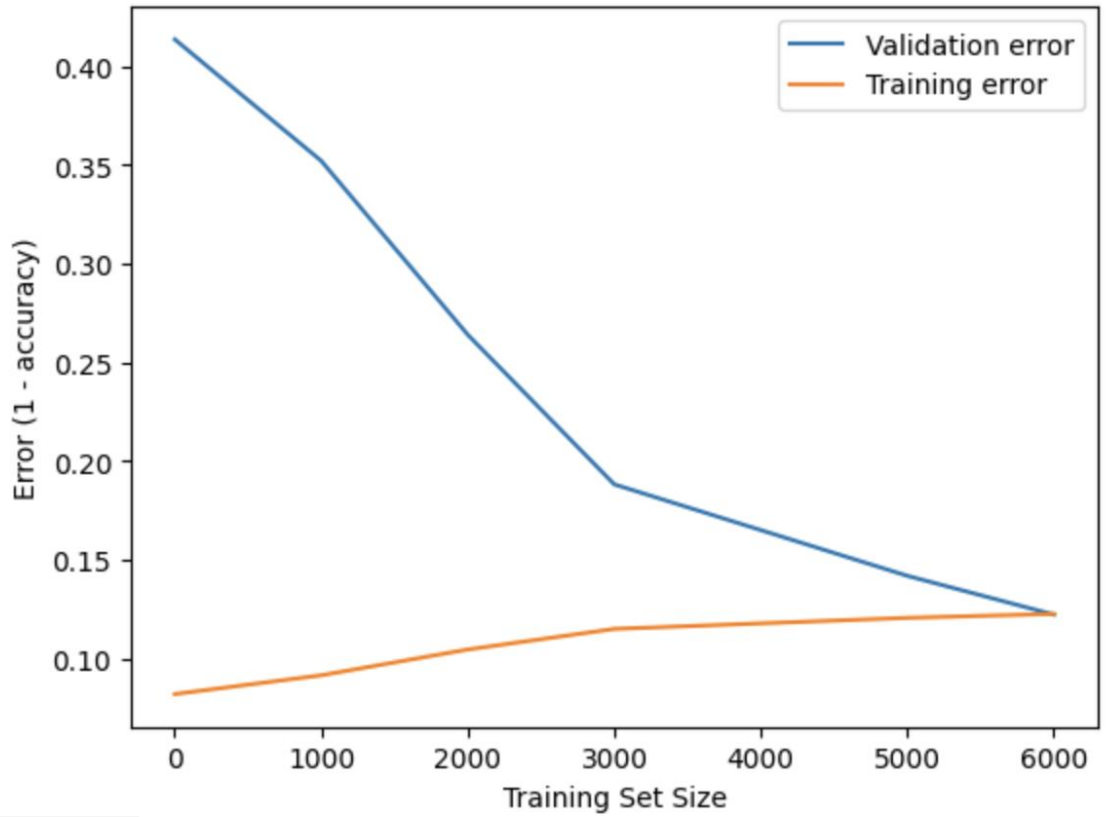
```
X, y = df.drop("Class", axis = 1), df["Class"]

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.1, random_state=1)

X_train, X_valid, y_train, y_valid = train_test_split(X_train, y_train, test_size=0.11, random_state=1)
```

```
#NN without normalization
from sklearn.neural_network import MLPClassifier
from sklearn.metrics import accuracy_score
NN_model = MLPClassifier(random_state=0, max_iter=1000)
NN_model.fit(X_train,y_train)

y_pred1=NN_model.predict(X_valid)
y_pred2=NN_model.predict(X_test)

accuracy1=accuracy_score(y_pred1,y_valid)
accuracy2=accuracy_score(y_pred2,y_test)
print('Accuracy of validation set:',accuracy1)
print('Accuracy of test set:',accuracy2)
```

```
Accuracy of validation set: 0.8275862068965517
Accuracy of test set: 0.8090062111801242
```

```
train_error=[]
valid_error=[]
m=[1,1000,2000,3000,5000,6000]
for j in m:
    NN_model = MLPClassifier(random_state=42, max_iter=1000)
    NN_model.fit(X_train[:j],y_train[:j])
    y_pred1=NN_model.predict(X_valid)
    y_pred2=NN_model.predict(X_train[:j])
    accuracy1=accuracy_score(y_pred1,y_valid)
    accuracy2=accuracy_score(y_pred2,y_train[:j])

    train_error.append(1-accuracy2)
    valid_error.append(1-accuracy1)
```

```
import matplotlib.pyplot as plt
from scipy.ndimage import gaussian_filter1d
plt.plot(m,gaussian_filter1d(valid_error, sigma=2),label='Validation error')
plt.plot(m,gaussian_filter1d(train_error,sigma=2),label='Training error')
plt.legend()
plt.xlabel('Training Set Size')
plt.ylabel('Error (1 - accuracy)')
```



Neural Network

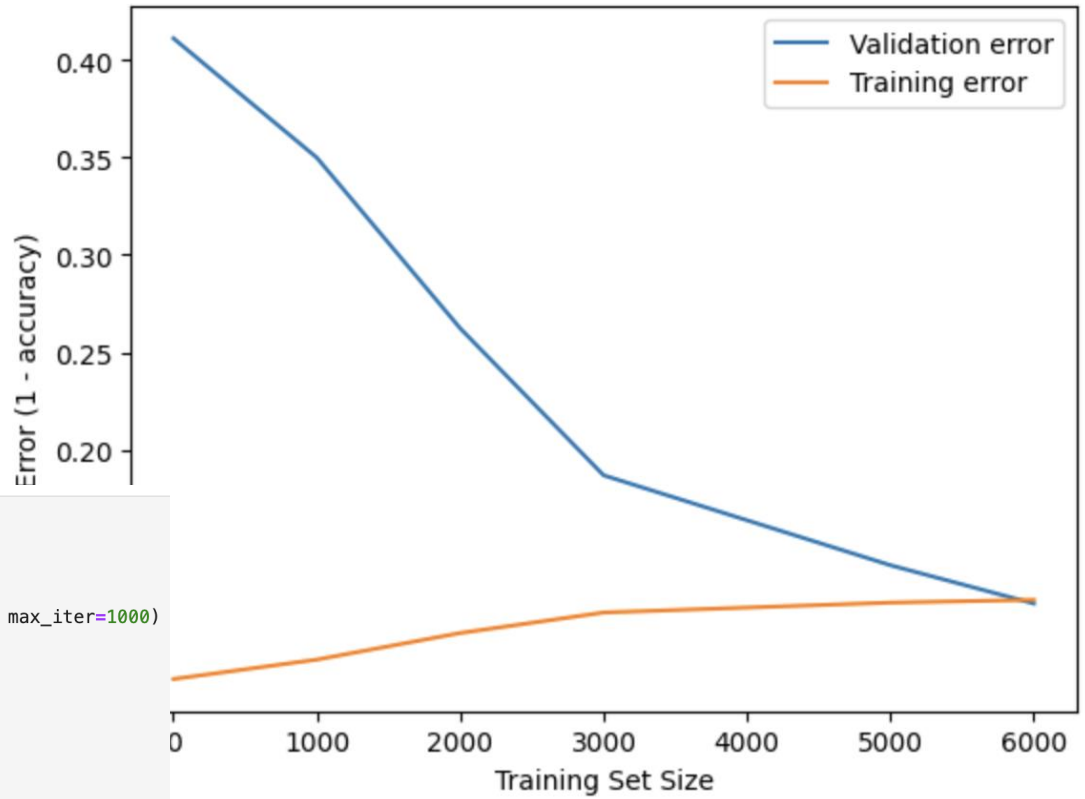
```
X, y = df.drop("Class", axis = 1), df["Class"]
#X, y = df.drop(["quality","Id"], axis = 1), df["quality"]
#normalization
for column in X.columns:
    X[column] = X[column]  / X[column].abs().max()

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.1, random_state=1)

X_train, X_valid, y_train, y_valid = train_test_split(X_train, y_train, test_size=0.11, random_state=1)
```

```
# NN with normalization and dif parameters
from sklearn.neural_network import MLPClassifier
from sklearn.metrics import accuracy_score

NN_model = MLPClassifier(hidden_layer_sizes=(57,),random_state=0,max_iter=2000)
NN_model.fit(X_train,y_train)

y_pred1=NN_model.predict(X_valid)
y_pred2=NN_model.predict(X_test)

accuracy1=accuracy_score(y_pred1,y_valid)
accuracy2=accuracy_score(y_pred2,y_test)
print('Accuracy of validation set:',accuracy1)
print('Accuracy of test set:',accuracy2)
```

```
Accuracy of validation set: 0.8981191222570533
Accuracy of test set: 0.906832298136646
```

```
train_error=[]
valid_error=[]
m=[1,1000,2000,3000,5000,6000]
for j in m:
    NN_model = MLPClassifier(hidden_layer_sizes=(57,),random_state=0, max_iter=1000)
    NN_model.fit(X_train[:j],y_train[:j])
    y_pred1=NN_model.predict(X_valid)
    y_pred2=NN_model.predict(X_train[:j])
    accuracy1=accuracy_score(y_pred1,y_valid)
    accuracy2=accuracy_score(y_pred2,y_train[:j])

    train_error.append(1-accuracy2)
    valid_error.append(1-accuracy1)
```

```
import matplotlib.pyplot as plt
from scipy.ndimage import gaussian_filter1d
plt.plot(m,gaussian_filter1d(valid_error, sigma=2),label='Validation error')
plt.plot(m,gaussian_filter1d(train_error,sigma=2),label='Training error')
plt.legend()
plt.xlabel('Training Set Size')
plt.ylabel('Error (1 - accuracy)')
```



Here I added normalization and changed the parameter hidden layer to 57. It helped me increase accuracy from 0.82 to 0.90

Decision tree

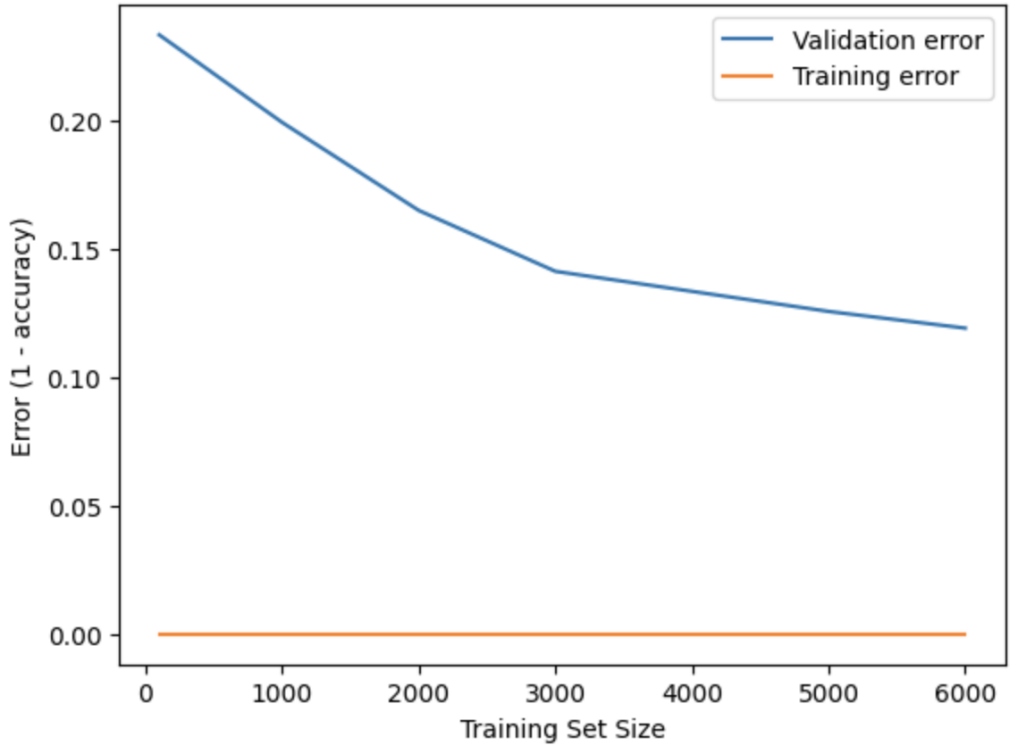
```
df = pd.read_csv('/Users/asik/Downloads/satellite.csv')
df['Class'] = df['Class'].replace({7 : 6})
X, y = df.drop("Class", axis = 1), df["Class"]

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.1, random_state=1)

X_train, X_valid, y_train, y_valid = train_test_split(X_train, y_train, test_size=0.11, random_state=1)
```

```
from sklearn.tree import DecisionTreeClassifier

# Modeling
clf = DecisionTreeClassifier(random_state=0)
clf.fit(X_train,y_train)
y_pred1 = clf.predict(X_valid)
y_pred2 = clf.predict(X_test)
accuracy1=accuracy_score(y_pred1,y_valid)
accuracy2=accuracy_score(y_pred2,y_test)
print('Accuracy of validation set:',accuracy1)
print('Accuracy of test set:',accuracy2)
```

```
Accuracy of validation set: 0.8824451410658307
Accuracy of test set: 0.8788819875776398
```

```
]: train_error=[]
valid_error=[]
m=[100,1000,2000,3000,5000,6000]
for j in m:
    clf = DecisionTreeClassifier(random_state=0)
    clf.fit(X_train[:j],y_train[:j])

    y_pred1=clf.predict(X_valid)
    y_pred2=clf.predict(X_train[:j])

    accuracy1=accuracy_score(y_pred1,y_valid)
    accuracy2=accuracy_score(y_pred2,y_train[:j])

    train_error.append(1-accuracy2)
    valid_error.append(1-accuracy1)
```

```
]: import matplotlib.pyplot as plt
from scipy.ndimage import gaussian_filter1d
plt.plot(m,gaussian_filter1d(valid_error, sigma=1),label='Validation error')
plt.plot(m,gaussian_filter1d(train_error,sigma=1),label='Training error')
plt.legend()
plt.xlabel('Training Set Size')
plt.ylabel('Error (1 - accuracy)')
```



Decision tree

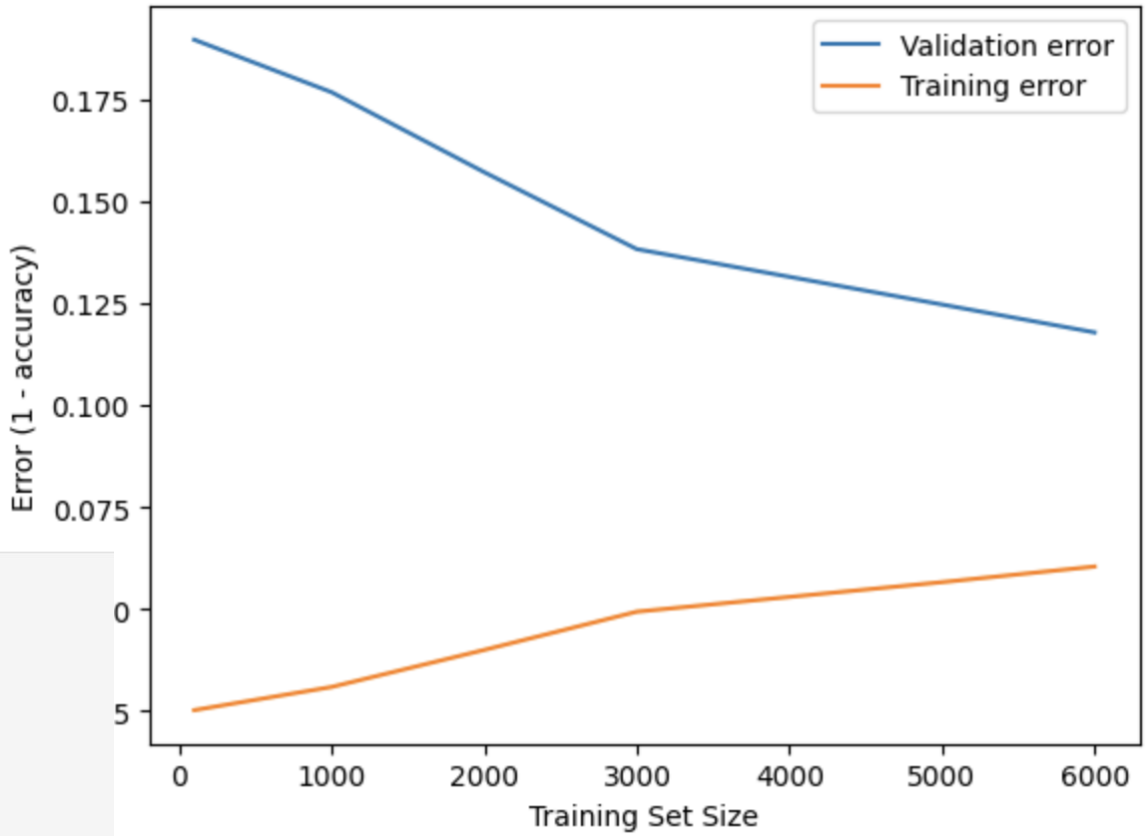
```
X, y = df.drop("Class", axis = 1), df["Class"]

#normalization
for column in X.columns:
    X[column] = X[column]  / X[column].abs().max()

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.1, random_state=1)

X_train, X_valid, y_train, y_valid = train_test_split(X_train, y_train, test_size=0.11, random_state=1)
```

```
#DEcision tree with norm
from sklearn.tree import DecisionTreeClassifier

clf = DecisionTreeClassifier(random_state=0,max_depth=9)
clf.fit(X_train,y_train)

y_pred1 = clf.predict(X_valid)
y_pred2 = clf.predict(X_test)

accuracy1=accuracy_score(y_pred1,y_valid)
accuracy2=accuracy_score(y_pred2,y_test)
print('Accuracy of validation set:',accuracy1)
print('Accuracy of test set:',accuracy2)
```

```
Accuracy of validation set: 0.8996865203761756
Accuracy of test set: 0.8726708074534162
```

```
train_error=[]
valid_error=[]
m=[100,1000,2000,3000,5000,6000]
for j in m:
    clf = DecisionTreeClassifier(random_state=0,max_depth=9)
    clf.fit(X_train[:j],y_train[:j])
    y_pred1=clf.predict(X_valid)
    y_pred2=clf.predict(X_train[:j])
    accuracy1=accuracy_score(y_pred1,y_valid)
    accuracy2=accuracy_score(y_pred2,y_train[:j])

    train_error.append(1-accuracy2)
    valid_error.append(1-accuracy1)
```

```
import matplotlib.pyplot as plt
from scipy.ndimage import gaussian_filter1d
plt.plot(m,gaussian_filter1d(valid_error, sigma=2),label='Validation error')
plt.plot(m,gaussian_filter1d(train_error,sigma=2),label='Training error')
plt.legend()
plt.xlabel('Training Set Size')
plt.ylabel('Error (1 - accuracy)')
```



Here I added normalization and changed parameter max_depth to 9 and it helped to improve learning curve and increased accuracy from 0.88 to 0.90.

Random Forest

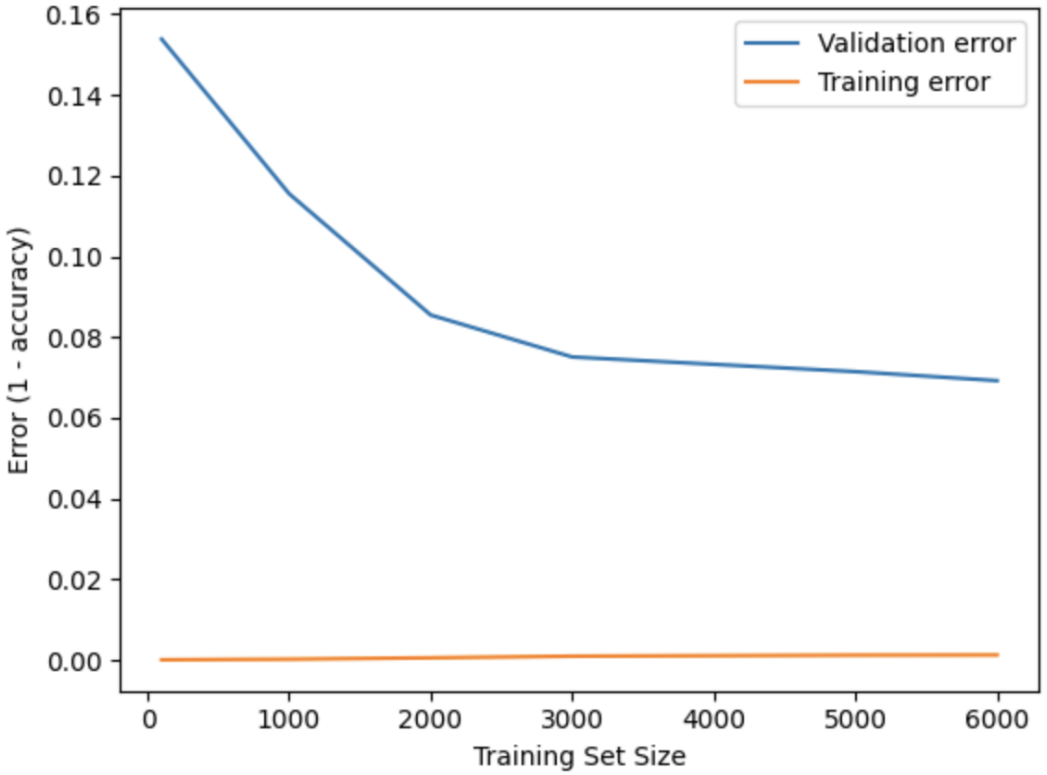
```
df = pd.read_csv('/Users/asik/Downloads/satellite.csv')
df['Class'] = df['Class'].replace({7 : 6})
X, y = df.drop("Class", axis = 1), df["Class"]

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.1, random_state=1)

X_train, X_valid, y_train, y_valid = train_test_split(X_train, y_train, test_size=0.11, random_state=1)
```

```
#Random Forest without normalization
from sklearn.ensemble import RandomForestClassifier

clf = RandomForestClassifier(max_depth=15, random_state=0)
clf.fit(X_train, y_train)

y_pred1 = clf.predict(X_valid)
y_pred2 = clf.predict(X_test)

accuracy1=accuracy_score(y_pred1,y_valid)
accuracy2=accuracy_score(y_pred2,y_test)
print('Accuracy of validation set:',accuracy1)
print('Accuracy of test set:',accuracy2)
```

```
Accuracy of validation set: 0.932601880877743
Accuracy of test set: 0.9332298136645962
```

```
train_error=[]
valid_error=[]
m=[100,1000,2000,3000,5000,6000]
for j in m:
    clf = RandomForestClassifier(max_depth=15, random_state=0)
    clf.fit(X_train[:j],y_train[:j])
    y_pred1=clf.predict(X_valid)
    y_pred2=clf.predict(X_train[:j])
    accuracy1=accuracy_score(y_pred1,y_valid)
    accuracy2=accuracy_score(y_pred2,y_train[:j])

    train_error.append(1-accuracy2)
    valid_error.append(1-accuracy1)
```

```
import matplotlib.pyplot as plt
from scipy.ndimage import gaussian_filter1d
plt.plot(m,gaussian_filter1d(valid_error, sigma=1),label='Validation error')
plt.plot(m,gaussian_filter1d(train_error,sigma=1),label='Training error')
plt.legend()
plt.xlabel('Training Set Size')
plt.ylabel('Error (1 - accuracy)')
```



Random Forest

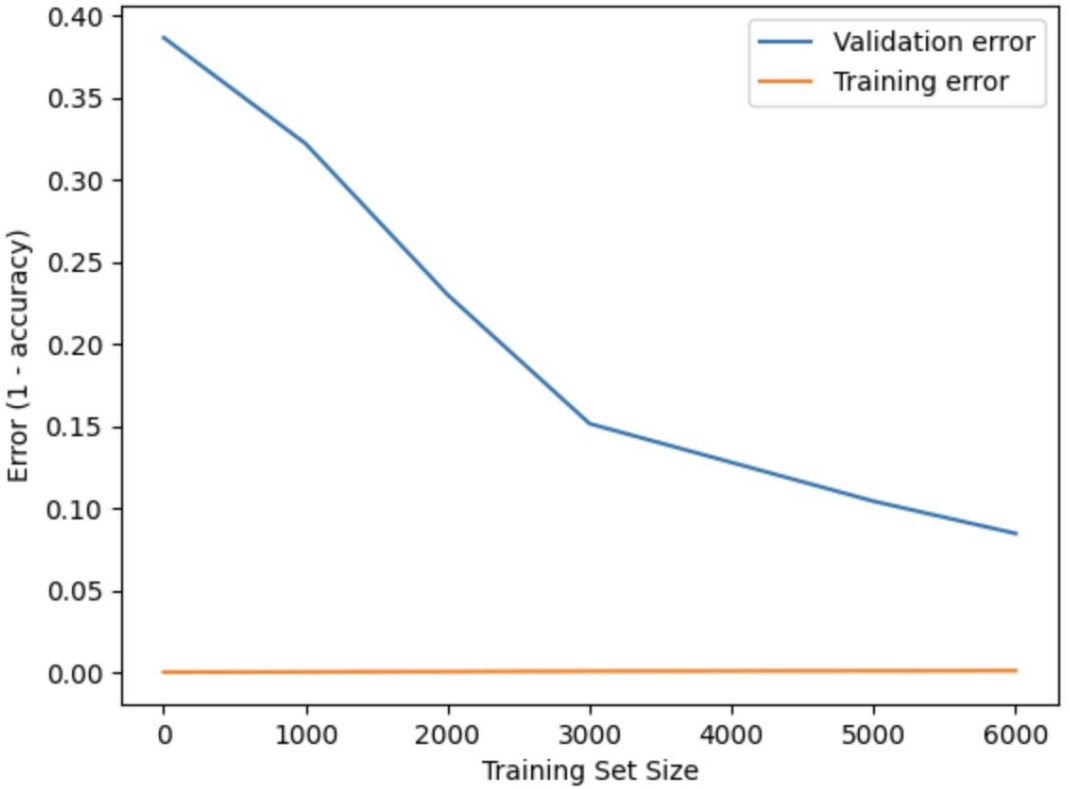
```
X, y = df.drop("Class", axis = 1), df["Class"]
#normalization
for column in X.columns:
    X[column] = X[column]  / X[column].abs().max()

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.1, random_state=1)

X_train, X_valid, y_train, y_valid = train_test_split(X_train, y_train, test_size=0.11, random_state=1)
```

```
#Random Forest with normalization
from sklearn.ensemble import RandomForestClassifier

clf = RandomForestClassifier(max_depth=15, random_state=0)
clf.fit(X_train, y_train)

y_pred1 = clf.predict(X_valid)
y_pred2 = clf.predict(X_test)

accuracy1=accuracy_score(y_pred1,y_valid)
accuracy2=accuracy_score(y_pred2,y_test)
print('Accuracy of validation set:',accuracy1)
print('Accuracy of test set:',accuracy2)
```

```
Accuracy of validation set: 0.9294670846394985
Accuracy of test set: 0.9332298136645962
```

```
train_error=[]
valid_error=[]
m=[1,1000,2000,3000,5000,6000]
for j in m:
    clf = RandomForestClassifier(max_depth=15, random_state=0)
    clf.fit(X_train[:j],y_train[:j])
    y_pred1=clf.predict(X_valid)
    y_pred2=clf.predict(X_train[:j])
    accuracy1=accuracy_score(y_pred1,y_valid)
    accuracy2=accuracy_score(y_pred2,y_train[:j])

    train_error.append(1-accuracy2)
    valid_error.append(1-accuracy1)
```

```
import matplotlib.pyplot as plt
from scipy.ndimage import gaussian_filter1d
plt.plot(m,gaussian_filter1d(valid_error, sigma=2),label='Validation error')
plt.plot(m,gaussian_filter1d(train_error,sigma=2),label='Training error')
plt.legend()
plt.xlabel('Training Set Size')
plt.ylabel('Error (1 - accuracy)')
```



Here I just added normalization and it improved the learning curve.

Gradient Boosting

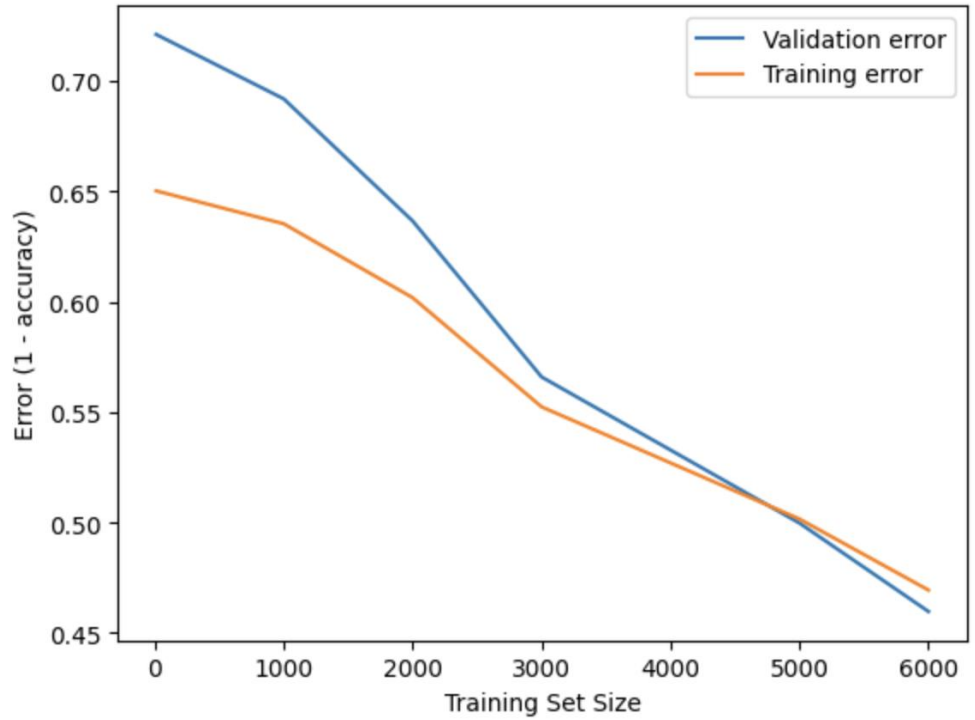
```
df = pd.read_csv('/Users/asik/Downloads/satellite.csv')
df['Class'] = df['Class'].replace({7 : 6})
X, y = df.drop("Class", axis = 1), df["Class"]

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.1, random_state=1)

X_train, X_valid, y_train, y_valid = train_test_split(X_train, y_train, test_size=0.11, random_state=1)
```

```
from sklearn.ensemble import GradientBoostingClassifier
clf = GradientBoostingClassifier(n_estimators=100, learning_rate=0.001,max_depth=1, random_state=0)
clf.fit(X_train, y_train)
y_pred1 = clf.predict(X_valid)
y_pred2 = clf.predict(X_test)

accuracy1=accuracy_score(y_pred1,y_valid)
accuracy2=accuracy_score(y_pred2,y_test)
print('Accuracy of validation set:',accuracy1)
print('Accuracy of test set:',accuracy2)
```

```
Accuracy of validation set: 0.6332288401253918
Accuracy of test set: 0.6304347826086957
```

```
train_error=[]
valid_error=[]
m=[10,1000,2000,3000,5000,6000]
for j in m:
    clf = GradientBoostingClassifier(n_estimators=50, learning_rate=0.001,max_depth=1
    clf.fit(X_train[:j],y_train[:j])
    y_pred1=clf.predict(X_valid)
    y_pred2=clf.predict(X_train[:j])
    accuracy1=accuracy_score(y_pred1,y_valid)
    accuracy2=accuracy_score(y_pred2,y_train[:j])

    train_error.append(1-accuracy2)
    valid_error.append(1-accuracy1)
```

```
import matplotlib.pyplot as plt
from scipy.ndimage import gaussian_filter1d
plt.plot(m,gaussian_filter1d(valid_error, sigma=2),label='Validation error')
plt.plot(m,gaussian_filter1d(train_error,sigma=2),label='Training error')
plt.legend()
plt.xlabel('Training Set Size')
plt.ylabel('Error (1 - accuracy)')
```



Gradient Boosting

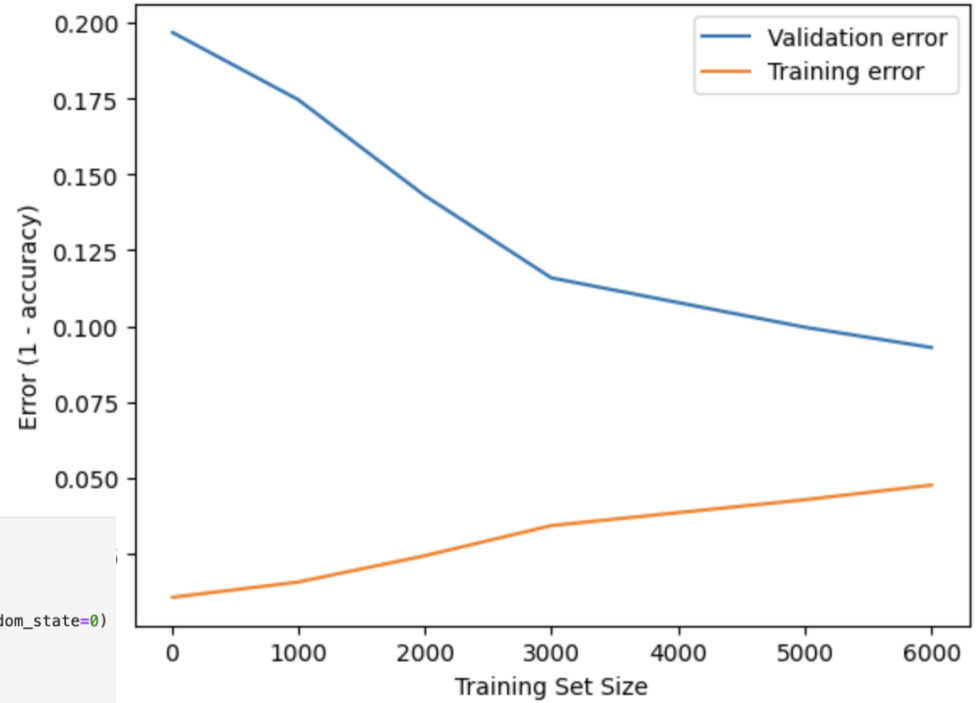
```
X, y = df.drop("Class", axis = 1), df["Class"]
#normalization
for column in X.columns:
    X[column] = X[column]  / X[column].abs().max()

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.1, random_state=1)

X_train, X_valid, y_train, y_valid = train_test_split(X_train, y_train, test_size=0.11, random_state=1)
```

```
from sklearn.ensemble import GradientBoostingClassifier
clf = GradientBoostingClassifier(n_estimators=1000, learning_rate=0.1,max_depth=1, random_state=0)
clf.fit(X_train, y_train)
y_pred1 = clf.predict(X_valid)
y_pred2 = clf.predict(X_test)

accuracy1=accuracy_score(y_pred1,y_valid)
accuracy2=accuracy_score(y_pred2,y_test)
print('Accuracy of validation set:',accuracy1)
print('Accuracy of test set:',accuracy2)
```

```
Accuracy of validation set: 0.9106583072100314
Accuracy of test set: 0.9083850931677019
```

```
train_error=[]
valid_error=[]
m=[10,1000,2000,3000,5000,6000]
for j in m:
    clf=GradientBoostingClassifier(n_estimators=1000, learning_rate=0.1,max_depth=1, random_state=0)
    clf.fit(X_train[:j],y_train[:j])
    y_pred1=clf.predict(X_valid)
    y_pred2=clf.predict(X_train[:j])
    accuracy1=accuracy_score(y_pred1,y_valid)
    accuracy2=accuracy_score(y_pred2,y_train[:j])

    train_error.append(1-accuracy2)
    valid_error.append(1-accuracy1)
```

```
import matplotlib.pyplot as plt
from scipy.ndimage import gaussian_filter1d
plt.plot(m,gaussian_filter1d(valid_error, sigma=2),label='Validation error')
plt.plot(m,gaussian_filter1d(train_error,sigma=2),label='Training error')
plt.legend()
plt.xlabel('Training Set Size')
plt.ylabel('Error (1 - accuracy)')
```



I added normalization and changed parameters like learning rate and number of estimators so it increased accuracy from 0.63 to 0.91 and improved learning curve respectively.

Comparison table

Methods	Accuracy without normalization	Accuracy with normalization + different parameters changed
Neural network validation	0.827	0.898
test	0.809	0.906
Decision tree validation	0.882	0.90
test	0.879	0.872
Random Forest validation	0.933	0.929
test	0.933	0.933
Gradient Boosting validation	0.633	0.91
test	0.630	0.908