Machine Learning MIIS (lab 1)

(Due to October 26)

The goal of this task is to familiarize with the Colab environment and with scikit-learn, and to gain some experience on learning simple predictive models.

We will focus on the training error only and consider a training dataset, i.e., we will not look at generalization error on testing data.

Consider the LIBSVM repository of datasets: http://www.csie.ntu.edu.tw/~cjlin/libsvmtools/datasets/

```
# # Example of downloading a dataset
# !wget -t inf https://www.csie.ntu.edu.tw/~cjlin/libsvmtools/datasets/binary/a1a
```

```
# # Example of loading a dataset
# from sklearn.datasets import load_svmlight_file
# X_train, y_train = load_svmlight_file("a1a")
# print(X_train)
```

1. Regression Task

Instructions

Choose a regression dataset and apply linear regression on a random subset of the training set of increasing size. You should select training sets that include more and more data points.

1.1. *Plot the approximation error (square loss) on the training set as a function of the number of samples N (i.e. data points in the training set).* [*You can use matplotlib for plotting. You can average curves over several permutations of the samples in the training dataset to obtain smoother curves, or even use errorbars.*]

1.2. *Plot the cpu-time as a function of N.*

1.3. *Explain in detail the behaviour of both curves and relate this behavior with what you would expect from theory.*

1.4. *Explore how the learned weights change as a function of N. Can you find an interpretation for the learned weights? [You can, for example, plot the value of each weight (in a different colour) as a function of N].*

Solution

We selected the Geographical Analysis Spatial Data (`space_ga`) regression dataset, which can be found here. It consists of just over 3 thousand examples reporting votes for the 1980 US presidential election, by county.

The dependent variable `VOTES` describes the number of votes cast.

The dataset includes the following 6 features:

- `POP`: no. people of voting age (18+)
- `EDUCATION`: no. people with 12th grade or higher education
- `HOUSES`: no. of owner-occupied houses
- `INCOME`: aggregate income
- `XCOORD`: X spatial coordinate of county
- `YCOORD`: Y spatial coordinate of county

```
# Import sklearn stuff
from sklearn.datasets import load_svmlight_file
from sklearn.utils import shuffle
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error
```

```
# Import process_time() for cpu time
from time import process_time
```

```
# Import mean for easy averaging of model coefficients
from statistics import mean
```

```
# Import matplotlib for plotting
import matplotlib.pyplot as plt
%matplotlib inline

# Set figure and font size
plt.rcParams["figure.figsize"] = (15,10)
plt.rcParams.update({'font.size': 18})
```

```
# Download dataset
!wget -t inf https://www.csie.ntu.edu.tw/~cjlin/libsvmtools/datasets/regression/space_ga
```

```
--2021-10-26 09:59:11--  https://www.csie.ntu.edu.tw/~cjlin/libsvmtools/datasets/regression/space_ga
Resolving www.csie.ntu.edu.tw (www.csie.ntu.edu.tw)... 140.112.30.26
Connecting to www.csie.ntu.edu.tw (www.csie.ntu.edu.tw)|140.112.30.26|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 565473 (552K)
Saving to: ‘space_ga.1’

space_ga.1          100%[===================>] 552.22K   620KB/s    in 0.9s

2021-10-26 09:59:13 (620 KB/s) - ‘space_ga.1’ saved [565473/565473]
```

```
# Load dataset
X, y = load_svmlight_file("space_ga")

# Print dataset info
print(f'X shape : {X.shape}')
print(f'X format : {type(y)}')
```

```
X shape : (3107, 6)
X format : <class 'numpy.ndarray'>
```

```
def linreg(X, y, size_increment):
  """
  Selects susbets of increasing size, applies linear regression to each subset
  and returns square loss, cpu times and model coefficients for each subset

  NOTE: cpu time refers to time taken for model fitting
```

```
:param numpy.ndarray X: matrix of features
:param numpy.ndarray y: matrix of labels
:param int size_increment: size increment for each subset

:return dict results: loss cpu time and model coefficients for each subset
indexed by subset size
"""
results = {}

X, y = shuffle(X, y)

curr_subset_size = size_increment

n_subsets = X.shape[0] // size_increment

for n in range(n_subsets):
  # Get the first `curr_subset_size` rows from shuffled dataset
  X_subset, y_subset = X[:curr_subset_size], y[:curr_subset_size]

  start = process_time()
  model = LinearRegression().fit(X_subset,y_subset)
  end = process_time()

  results[curr_subset_size] = {}
  results[curr_subset_size]["cpu_time"] = end - start

  y_pred = model.predict(X)
  results[curr_subset_size]["mse"] = mean_squared_error(y, y_pred)

  results[curr_subset_size]["coef"] = model.coef_

  curr_subset_size += size_increment

  # Last subset should take remaining samples if division wasn't whole
  if n == n_subsets-2:
    curr_subset_size = X.shape[0]

return results
```

```
def avg_results_linreg(X, y, size_increment, n):
  """
  Runs linear regression on random subsets of increasing sizes n times, and
  computes average square loss and cpu time for each subset size

  NOTE: cpu time refers to time taken for model fitting

  :param numpy.ndarray X: matrix of features
  :param numpy.ndarray y: matrix of labels
  :param int size_increment: size increment for each subset
  :param int n: no. of times lin reg should be applied to random subsets

  :return dict avg_results: average loss and cpu time for each subset indexed by
  subset size
  """
  avg_results = {}

  for i in range(n):
    results = linreg(X, y, size_increment)

    # Store results in lists by subset size
    for size in results:
      if size in avg_results:
        avg_results[size]["mse"].append(results[size]["mse"])
        avg_results[size]["cpu_time"].append(results[size]["cpu_time"])
      else:
        avg_results[size] = {}
        avg_results[size]["mse"] = [results[size]["mse"]]
        avg_results[size]["cpu_time"] = [results[size]["cpu_time"]]

  # Average losses and cpu times for each subset size
  for size in avg_results.keys():
    losses = avg_results[size]["mse"]
    avg_results[size]["mse"] = sum(losses) / len(losses)

    cpu_times = avg_results[size]["cpu_time"]
    avg_results[size]["cpu_time"] = sum(cpu_times) / len(cpu_times)
```

```
  return avg_results
```

```
results = avg_results_linreg(X, y, size_increment=500, n=50)
```

```
results
```

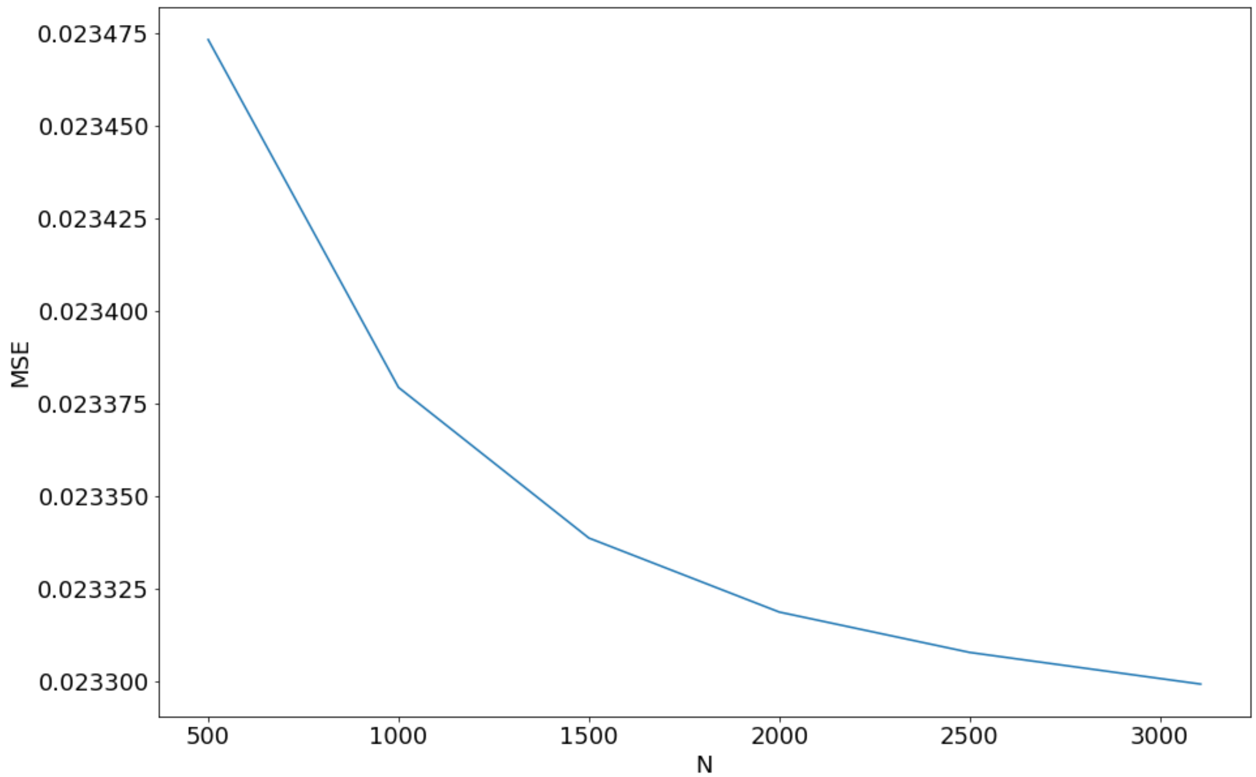
```
{500: {'cpu_time': 0.0016711040799999388, 'mse': 0.023473169762590288},
 1000: {'cpu_time': 0.001753435719999974, 'mse': 0.02337923207647167},
 1500: {'cpu_time': 0.0018810207800001066, 'mse': 0.02333855936282825},
 2000: {'cpu_time': 0.0019144412400000554, 'mse': 0.02331857160154746},
 2500: {'cpu_time': 0.002028778220000049, 'mse': 0.0233076697968729 6},
 3107: {'cpu_time': 0.0021679923799999834, 'mse': 0.023299114264797728}}
```

1.1. Plot the square loss as a function of N

```
# Parse results into lists for plotting
sizes = []
losses = []
for size,result in results.items():
  sizes.append(size)
  losses.append(result["mse"])

# Plot square loss against N
plt.xlabel("N")
plt.ylabel("MSE")
plt.plot(sizes, losses)
```

```
[<matplotlib.lines.Line2D at 0x7f06c6a3cbd0>]
```



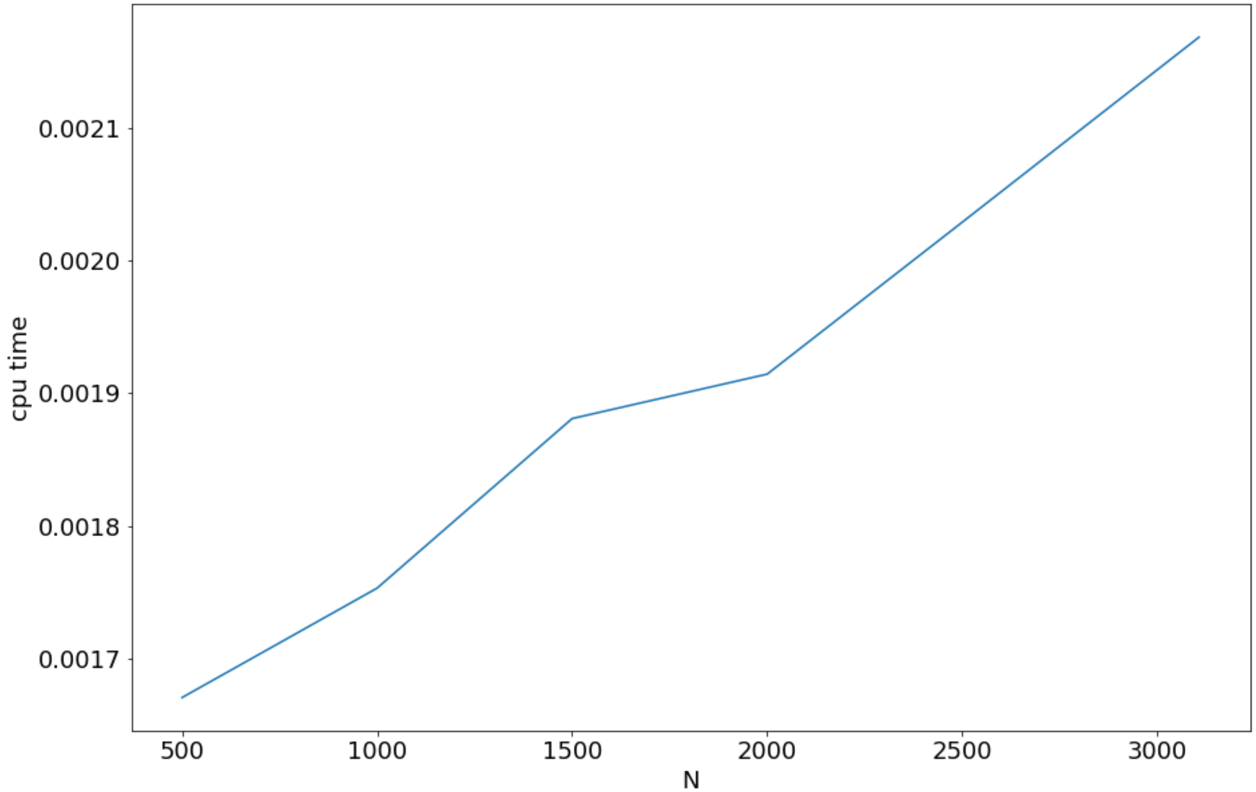
1.2. Plot the cpu-time as a function of N.

```
# Parse results into lists for plotting
sizes = []
times = []
for size,result in results.items():
  sizes.append(size)
  times.append(result["cpu_time"])

# Plot cpu time against N
plt.xlabel("N")
```

```
plt.ylabel("cpu time")
plt.plot(sizes, times)
```

```
[<matplotlib.lines.Line2D at 0x7f06c69a8750>]
```



1.3. Explain in detail the behaviour of both curves and relate this behavior with what you would expect from theory.

We ran linear regression on subsets of increasing sizes where each subset contains 500 more examples than the previous. We repeated this operation 50 times and averaged the results.

The first curve shows the average loss as a function of the number of training samples. As we can observe, the larger training set the lower the loss value. This behaviour is expected, since fitting a model on a larger training set allows us to obtain to better coefficients, which in turn cause the model to generate more accurate predictions. Since the loss function computes the distance between the gold labels and model outputs, the more accurate the predictions are the lower the loss value will be. In this case, the decrease is steeper between subsets of 500 and 1,500 exmaples, after which the loss continues to decrease, albeit at a lower rate. However it should be noted that the numeric change in loss is actually very slight, only ranging betwee around .0234 to around .0233.

The second curve shows the average cpu time for model fitting as a function of the number of training samples. It can be observed that larger training sets correspond to longer cpu time. This is expected, because a larger training set contains more examples, and the model requires more calculations to fit the weights. More expensive computations trivially lead to an increased cpu time.

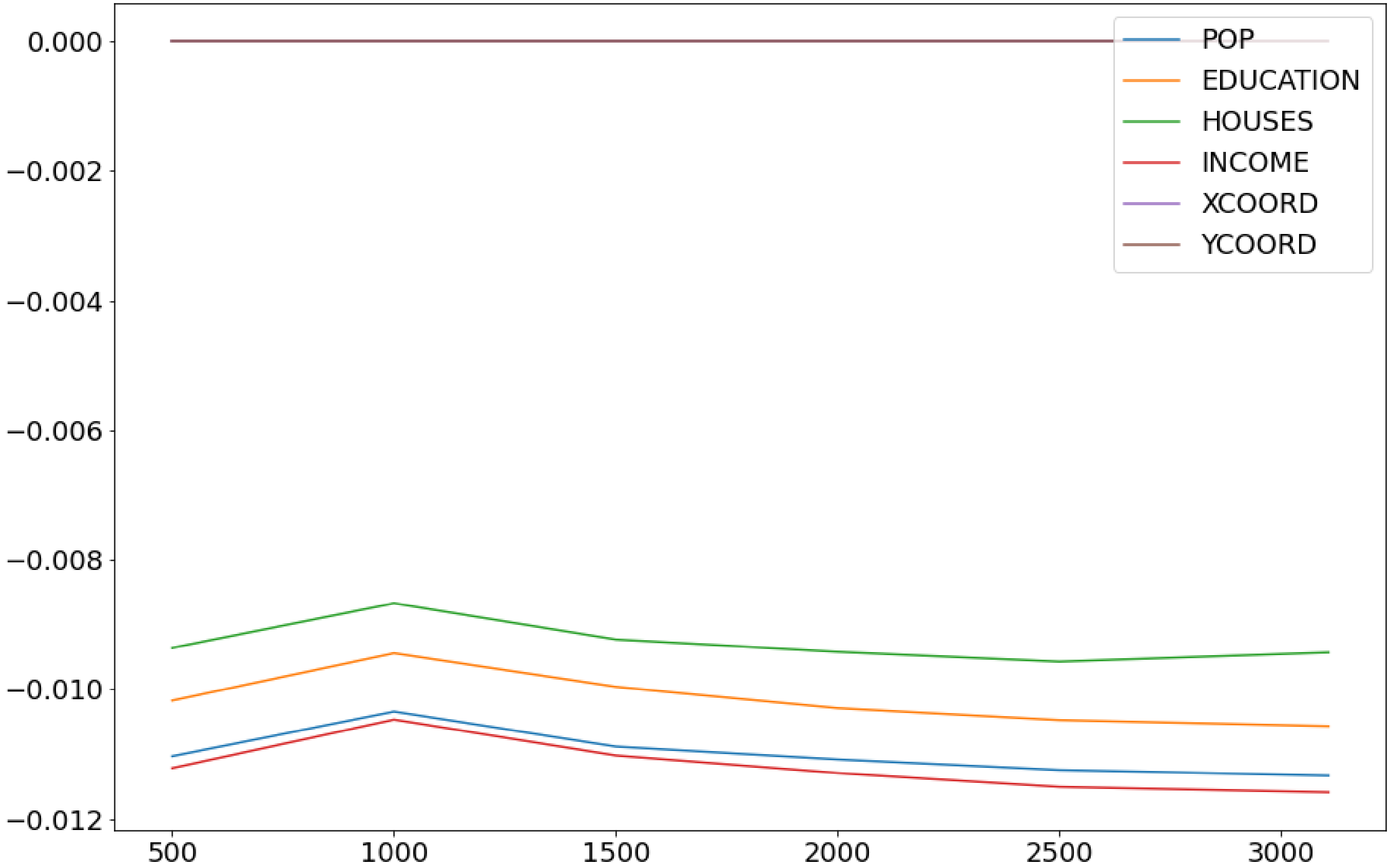
1.4. Explore how the learned weights change as a function of N. Can you find an interpretation for the learned weights?

[*You can, for example, plot the value of each weight (in a different colour) as a function of* N].

```
# Run linreg once
results = linreg(X, y, size_increment=500)
# Parse results into lists for plotting
sizes = []
coefs = []
for size,result in results.items():
  sizes.append(size)
  coefs.append(result["coef"])
```

```
# Plot coefs against N
plot = plt.plot(sizes, coefs)
labels = ["POP", "EDUCATION", "HOUSES", "INCOME", "XCOORD", "YCOORD"]
```

```
plt.legend(plot, labels, loc="upper right")
plt.show()
```



We ran linear regression on subsets of increasing size where each subset contains 500 more examples than the previous. Each line in the figure above shows how the corresponing weight changes when the training size grows, according to the top right legend.

We can observe that the weight associated with `YCOORD` does not significantly change with N, remaining close to $.00018$. This can be interpreted as a weak positive correlation with `VOTES`, corresponding to a $.00018$ (approx) increase in cast votes for every 1-unit increase in `YCOORD`. If we assume that `YCOORD` places the county on a point between two cardinal points on the US map, for example South vs North, then the higher the `YCOORD` the more North the county is located. Then, this correlation could be interpreted as decribing a tendency for Northern counties to cast slightly more votes.

The other weights are noticeably all negative, albeit very small. The changes for these are much more noticeable, and they all follow a similar pattern. Starting from a certain value, there is an increase between 500 and 1000 examples. Then all the weights slowly descend until they reach a value that in most cases is lower than the one from the first model. In general, all of these can be interpreted as describing a slightly negative correlation with `VOTES`, i.e. a decrease in `VOTES` for every 1-unit increase wrt corresponding features.