

**A REPORT OF SIX WEEKS INDUSTRIAL TRAINING**

at

**N.I.E.L.I.T**

SUBMITTED IN PARTIAL FULFILLMENT OF THE REQUIREMENT FOR THE AWARD  
OF THE DEGREE OF

**BACHELOR OF ENGINEERING**

(Computer Science & Engineering)



**CHANDIGARH  
UNIVERSITY**  
*Discover. Learn. Empower.*

JUNE - JULY , 2018

**SUBMITTED BY:**

NAME : Ankit

UNIVERSITY UID : 16BCS1129

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

CHANDIGARH UNIVERSITY GHARUAN, MOHALI

## CONTENTS

Topic	Page No.
<i>Certificate by Company/Industry/Institute</i>	<i>i</i>
<i>Candidate's Declaration</i>	<i>ii</i>
<i>Abstract</i>	<i>iii</i>
<i>Acknowledgement</i>	<i>vi</i>
<i>About the Company/ Industry / Institute</i>	<i>v</i>
<b>CHAPTER 1 INTRODUCTION</b>	
<b>CHAPTER 2 TRAINING WORK UNDERTAKEN</b>	
<b>CHAPTER 3 RESULTS AND DISCUSSION</b>	
<b>CHAPTER 4 CONCLUSION AND FUTURE SCOPE</b>	
<b>REFERENCES</b>	



## **ABSTRACT**

Industrial training is an important phase of a student life. A well planned, properly executed and evaluated industrial training helps a lot in developing a professional attitude. It develop an awareness of industrial approach to problem solving, based on a broad understanding of process and mode of operation of organization. The aim and motivation of this industrial training is to receive discipline, skills, teamwork and technical knowledge through a proper training environment, which will help me, as a student in the field of engineering, to develop a responsiveness of the self-disciplinary nature of problems in Computer Science. During a period of six weeks training at N.I.E.L.I.T (Mohali), I was assigned a project which predicts the number of servers requirement in a given hour based on the given previous observations. As a result I completed my project and on the basis of it, the servers' cost can be minimized and maintained in advance. Throughout this industrial training, I have learned Advanced Python with Machine Learning that was required for the project. I was able to catch up the concept easily based on my experience gained in programming at the University itself.

**CHANDIGARH UNIVERSITY,GHARUAN,MOHALI**

**CANDIDATE'S DECLARATION**

I , **Ankit** hereby declare that I have undertaken six weeks industrial training at **N.I.E.L.I.T** during a period from \_\_\_\_\_ to \_\_\_\_\_ in partial fulfillment of requirements for the award of degree of B.E (COMPUTER SCIENCE & ENGINEERING) at CHANDIGARH UNIVERSITY GHARUAN, MOHALI. The work which is being presented in the training report submitted to Department of Computer Science & Engineering at CHANDIGARH UNIVERSITY GHARUAN, MOHALI is an authentic record of training work.

Signature of the Student

The six weeks industrial training Viva–Voce Examination of \_\_\_\_\_ has been held on \_\_\_\_\_ and accepted.

Signature of Internal Examiner

Signature of External Examiner

## **ACKNOWLEDGEMENTS**

To become a professional in technical industry, industrial training is the foundation for each undergraduate student. It helps students to improve their practical skills related to interpersonal, problems solving, research and reporting as well as soft skills. Also it helps the students get exposure to the industry, apply the gained knowledge throughout the academic program and learn new updated technologies. In addition, it helps students' career development and to prepare for employment after graduation, by engaging in personal and professional development planning. I hereby to extend my sincere appreciation and thankfulness to my helpful internship supervisor Ms. Anita Budhiraja , Faculty at N.I.E.L.I.T for giving me this project as a challenge to enhance my technical skills. Further I would like to be thankful to Mr. Ankit Deswal, Batchmate. The supervision and support that he gave truly helped the progression and smoothness of the internship program. And also I would like to thank Mr. Sarvan Kumar, Faculty at N.I.E.L.I.T for his guidance and teaching me about the requirements of working in an I.T company as well as the ways to use the Machine Learning concepts during the internship. Besides, this internship program made me aware about the value of working together as a team and as a new experience in working environment. Not to forget, great appreciation to other department staff that helped me from time to time during our internship. The whole program really brought us together to appreciate the true value of friendship and respect of each other.

## **ABOUT N.I.E.L.I.T**

National Institute of Electronics & Information Technology (N.I.E.L.I.T), formerly known as the D.O.E.A.C.C Society, is a society that offers information technology training at different levels. It is associated with the Ministry of Electronics and Information Technology of the Government of India.

## **Introduction**

### **Python :**

Python is an interpreted high-level programming language for general-purpose programming. Created by Guido van Rossum and first released in 1991, Python has a design philosophy that emphasizes code readability, notably using significant whitespace. It provides constructs that enable clear programming on both small and large scales.

Python features a dynamic type system and automatic memory management. It supports multiple programming paradigms, including object-oriented, imperative, functional and procedural, and has a large and comprehensive standard library.

Python interpreters are available for many operating systems. CPython, the reference implementation of Python, is open source software and has a community-based development model, as do nearly all of its variant implementations. CPython is managed by the non-profit Python Software Foundation.

Python is a multi-paradigm programming language. Object-oriented programming and structured programming are fully supported, and many of its features support functional programming and aspect-oriented programming (including by metaprogramming and metaobjects (magic methods)). Many other paradigms are supported via extensions, including design by contract and logic programming.

Python uses dynamic typing, and a combination of reference counting and a cycle-detecting garbage collector for memory management. It also features dynamic name resolution (late binding), which binds method and variable names during program execution. Python's design offers some support for functional programming in the Lisp tradition. It has `filter()`, `map()`, and `reduce()` functions; list comprehensions, dictionaries, and sets; and generator expressions. The standard library has two modules (`itertools` and `functools`) that implement functional tools borrowed from Haskell and Standard ML.



## **Python libraries :**

**Numpy :** is a library for the Python programming language, adding support for large, multi-dimensional arrays and matrices, along with a large collection of high-level mathematical functions to operate on these arrays. The ancestor of NumPy, Numeric, was originally created by Jim Hugunin with contributions from several other developers. In 2005, Travis Oliphant created NumPy by incorporating features of the competing Numarray into Numeric, with extensive modifications. NumPy is open-source software and has many contributors. NumPy targets the CPython reference implementation of Python, which is a non-optimizing bytecode interpreter. Mathematical algorithms written for this version of Python often run much slower than compiled equivalents. NumPy addresses the slowness problem partly by providing multidimensional arrays and functions and operators that operate efficiently on arrays, requiring rewriting some code, mostly inner loops using NumPy. Using NumPy in Python gives functionality comparable to MATLAB since they are both interpreted, and they both allow the user to write fast programs as long as most operations work on arrays or matrices instead of scalars.

**Pandas :** is a software library written for the Python programming language for data manipulation and analysis. In particular, it offers data structures and operations for manipulating numerical tables and time series. It is free software released under the three-clause BSD license. The name is derived from the term "panel data", an econometrics term for data sets that include observations over multiple time periods for the same individuals. DataFrame object for data manipulation with integrated indexing. Tools for reading and writing data between in-memory data structures and different file formats. Data alignment and integrated handling of missing data. Reshaping and pivoting of data sets. Label-based slicing, fancy indexing, and subsetting of large data sets. Data structure column insertion and deletion. Group by engine allowing split-apply-combine operations on data sets. Data set merging and joining. Hierarchical axis indexing to work with high-dimensional data in a lower-dimensional data structure.

Time series-functionality: Date range generation and frequency conversion, moving window statistics, moving window linear regressions, date shifting and lagging. The library is highly optimized for performance, with critical code paths written in Cython or C.

**Matplotlib** : is a plotting library for the Python programming language and its numerical mathematics extension NumPy. It provides an object-oriented API for embedding plots into applications using general-purpose GUI toolkits like Tkinter, wxPython, Qt, or GTK+. There is also a procedural "pylab" interface based on a state machine (like OpenGL), designed to closely resemble that of MATLAB, though its use is discouraged. SciPy makes use of matplotlib. Matplotlib was originally written by John D. Hunter, as an active development community, and is distributed under a BSD-style license. Michael Droettboom was nominated as matplotlib's lead developer shortly before John Hunter's death in 2012, and further joined by Thomas Caswell. As of 23 June 2017, matplotlib 2.0.x supports Python versions 2.7 through 3.6. Matplotlib 1.2 is the first version of matplotlib to support Python 3.x.

Matplotlib 1.4 is the last version of matplotlib to support Python 2.6. Matplotlib has pledged to not support Python 2 past 2020 by signing the Python 3 Statement.

Basemap: map plotting with various map projections, coastlines, and political boundaries.

Cartopy: a mapping library featuring object-oriented map projection definitions, and arbitrary point, line, polygon and image transformation capabilities. (matplotlib v1.2 and above) Excel tools: utilities for exchanging data with Microsoft Excel GTK tools: interface to the GTK+ library Qt interface Mplot3d: 3-D plots

Natgrid: interface to the natgrid library for gridding irregularly spaced data.

**Scipy** : is a free and open-source Python library used for scientific computing and technical computing. SciPy contains modules for optimization, linear algebra, integration, interpolation, special functions, FFT, signal and image processing, ODE solvers and other tasks common in science and engineering. SciPy builds on the NumPy array object and is part of the NumPy stack which includes tools like Matplotlib, pandas and SymPy, and an expanding set of scientific computing libraries. This NumPy stack has similar users to other applications such as MATLAB, GNU Octave, and Scilab. The NumPy stack is also sometimes referred to as the SciPy stack. SciPy is also a family of conferences for users and developers of these tools: SciPy (in the United States), EuroSciPy (in Europe) and SciPy.in (in India). Enthought originated the SciPy conference in the United States and continues to sponsor many of the international conferences as well as host the SciPy website. The SciPy library is currently distributed under the BSD license, and its development is sponsored and supported by an open community of developers. It is also supported by Numfocus which is a community foundation for supporting reproducible and accessible science.

**Scikit Learn** : is a free software machine learning library for the Python programming language.[3] It features various classification, regression and clustering algorithms including support vector machines, random forests, gradient boosting, k-means and DBSCAN, and is designed to interoperate with the Python numerical and scientific libraries NumPy and SciPy. The scikit-learn project started as scikits.learn, a Google Summer of Code project by David Cournapeau. Its name stems from the notion that it is a "SciKit" (SciPy Toolkit), a separately-developed and distributed third-party extension to SciPy. The original codebase was later rewritten by other developers. In 2010 Fabian Pedregosa, Gael Varoquaux, Alexandre Gramfort and Vincent Michel, all from INRIA took leadership of the project and made the first public release on February the 1st 2010.[5] Of the various scikits, scikit-learn as well as scikit-image were described as "well-maintained and popular" in November 2012. As of 2018, scikit-learn is under active development.

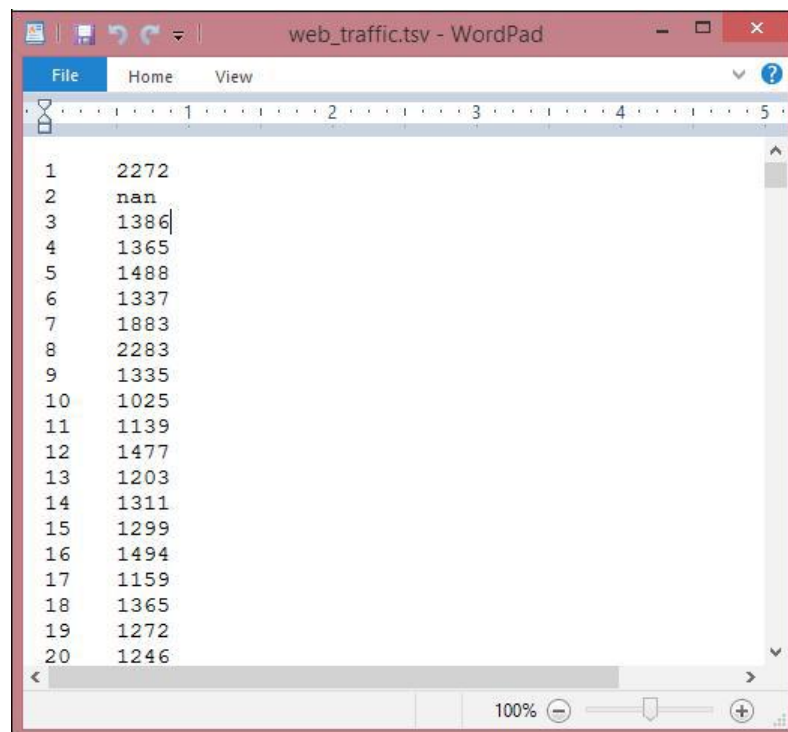
## TRAINING WORK UNDERTAKEN

**Project's Aim :** To determine the number of servers required by a particular website in the future at a given hour of the day.

**Given :** Number of website hits per hour observation data.

### Reading in the data :

We have collected the web stats for the last month and aggregated them in web\_traffic.tsv (.tsv because it contains tab-separated values). They are stored as the number of hits per hour. Each line contains the hour consecutively and the number of web hits in that hour.



1	2272
2	nan
3	1386
4	1365
5	1488
6	1337
7	1883
8	2283
9	1335
10	1025
11	1139
12	1477
13	1203
14	1311
15	1299
16	1494
17	1159
18	1365
19	1272
20	1246

Using SciPy's `genfromtxt()`, we can easily read in the data using the following code:

- **`import scipy as sp`**
- **`data = sp.genfromtxt("web_traffic.tsv", delimiter="\t")`**

We have to specify tab as the delimiter so that the columns are correctly determined.

## Pre-processing and cleaning the data

It is more convenient for SciPy to separate the dimensions into two vectors, each of size 743. The first vector, `x`, will contain the hours, and the other, `y`, will contain the Web hits in that particular hour. This splitting is done using the special index notation of SciPy, by which we can choose the columns individually:

```
x = data[:,0]
```

```
y = data[:,1]
```

Remember that we can index a SciPy array with another array. `Sp.isnan(y)` returns an array of Booleans indicating whether an entry is a number or not. Using `~`, we logically negate that array so that we choose only those elements from `x` and `y` where `y` contains valid numbers:

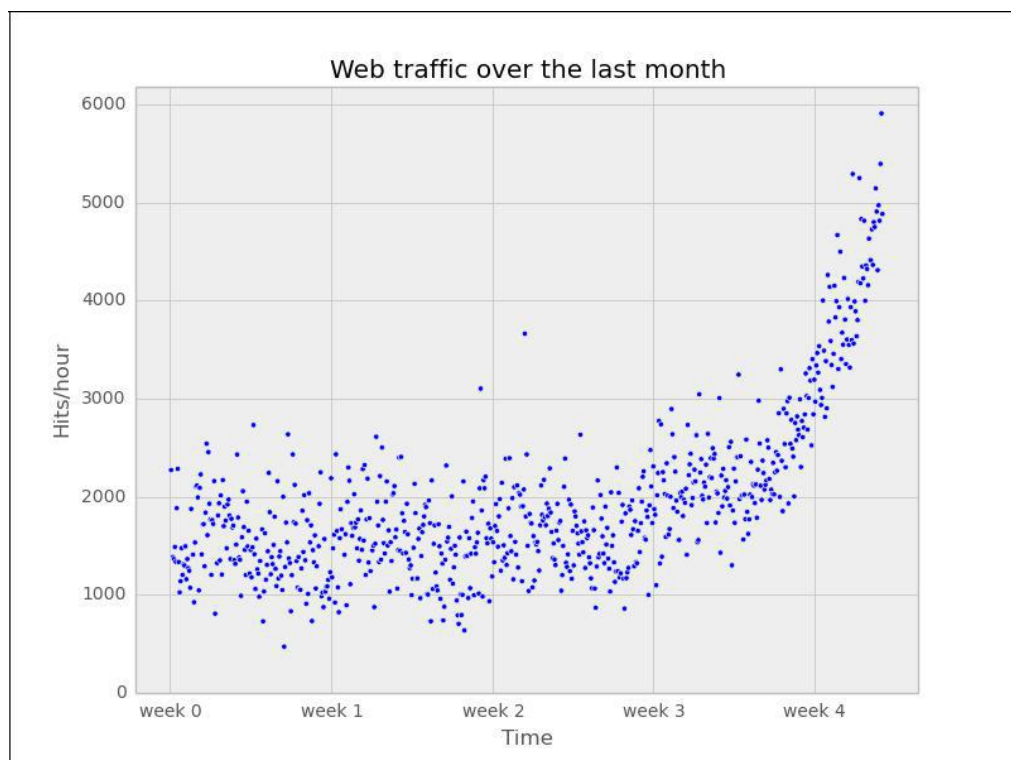
- **`x = x[~sp.isnan(y)]`**
- **`y = y[~sp.isnan(y)]`**

To get the first impression of our data, let's plot the data in a scatter plot using matplotlib. matplotlib contains the pyplot package, which tries to mimic MATLAB's interface, which is a very convenient and easy to use one as you can see in the following code:

- **`import matplotlib.pyplot as plt`**
- **`# plot the (x,y) points with dots of size 10`**
- **`plt.scatter(x, y, s=10)`**

- `plt.title("Web traffic over the last month")`
- `plt.xlabel("Time")`
- `plt.ylabel("Hits/hour")`
- `plt.xticks([w*7*24 for w in range(10)], ['week %i' % w  
for w in range(10)])`
- `plt.autoscale(tight=True)`
- `# draw a slightly opaque, dashed grid`
- `plt.grid(True, linestyle='-', color='0.75')`
- `plt.show()`

In the resulting chart, we can see that while in the first weeks the traffic stayed more or less the same, the last week shows a steep increase:



# Choosing the right model and learning algorithm

Now that we have a first impression of the data, we return to the initial question: How long will our server handle the incoming web traffic? To answer this we have to do the following:

- Find the real model behind the noisy data points.
- Following this, use the model to extrapolate into the future to find the point in time where our infrastructure has to be extended.

## Before building our first model...

When we talk about models, you can think of them as simplified theoretical approximations of complex reality. As such there is always some inferiority involved, also called the approximation error. This error will guide us in choosing the right model among the myriad of choices we have. And this error will be calculated as the squared distance of the model's prediction to the real data; for example, for a learned model function  $f$ , the error is calculated as follows:

```
def error(f, x, y):  
  
    return sp.sum((f(x)-y)**2)
```

The vectors  $x$  and  $y$  contain the web stats data that we have extracted earlier. It is the beauty of SciPy's vectorized functions that we exploit here with  $f(x)$ . The trained model is assumed to take a vector and return the results again as a vector of the same size so that we can use it to calculate the difference to  $y$ .

## Starting with a simple straight line

Let's assume for a second that the underlying model is a straight line. Then the challenge is how to best put that line into the chart so that it results in the smallest approximation error. SciPy's `polyfit()`

function does exactly that. Given data  $x$  and  $y$  and the desired order of the polynomial (a straight line has order 1), it finds the model function that minimizes the error function defined earlier:

```
fp1, residuals, rank, sv, rcond = sp.polyfit(x, y, 1, full=True)
```

The `polyfit()` function returns the parameters of the fitted model function, `fp1`. And by setting `full=True`, we also get additional background information on the fitting process. Of this, only residuals are of interest, which is exactly the error of the approximation:

```
>>> print("Model parameters: %s" % fp1)
```

```
Model parameters: [ 2.59619213 , 989.02487106 ]
```

- **`print(residuals)`**

```
[ 3.17389767e+08]
```

This means the best straight line fit is the following function

$$f(x) = 2.59619213 * x + 989.02487106.$$

We then use `poly1d()` to create a model function from the model parameters:

- **`f1 = sp.poly1d(fp1)`**

- **`print(error(f1, x, y))`**

```
317389767.34
```

We have used `full=True` to retrieve more details on the fitting process. Normally, we would not need it, in which case only the model parameters would be returned.

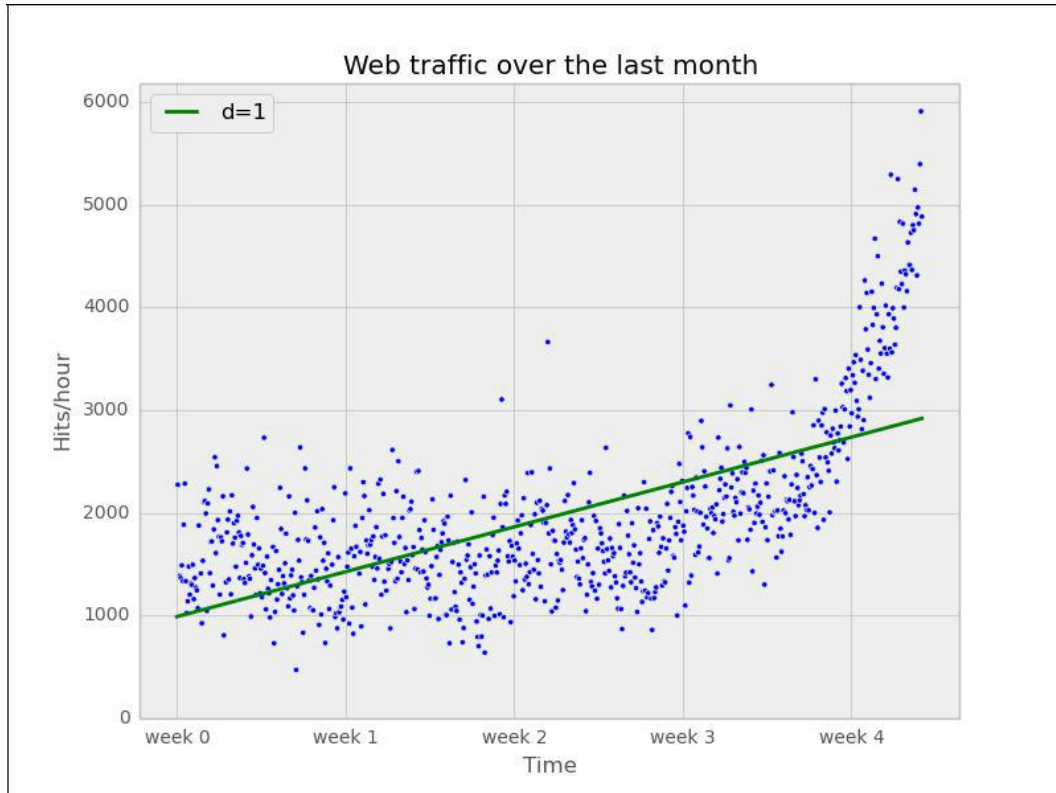
We can now use `f1()` to plot our first trained model. In addition to the preceding plotting instructions, we simply add the following code:

```
fx = sp.linspace(0,x[-1], 1000) # generate X-values for plotting
```

```
plt.plot(fx, f1(fx), linewidth=4)
```



This will produce the following plot:



It seems like the first 4 weeks are not that far off, although we clearly see that there is something wrong with our initial assumption that the underlying model is a straight line. And then, how good or how bad actually is the error of 317,389,767.34?

The absolute value of the error is seldom of use in isolation. However, when comparing two competing models, we can use their errors to judge which one of them is better. Although our first model clearly is not the one we would use, it serves a very important purpose in the workflow. We will use it as our baseline until we find a better one. Whatever model we come up with in the future, we will compare it against the current baseline.

## Towards some advanced stuff

Let's now fit a more complex model, a polynomial of degree 2, to see whether it better understands our data:

- `f2p = sp.polyfit(x, y, 2)`

- `print(f2p)`

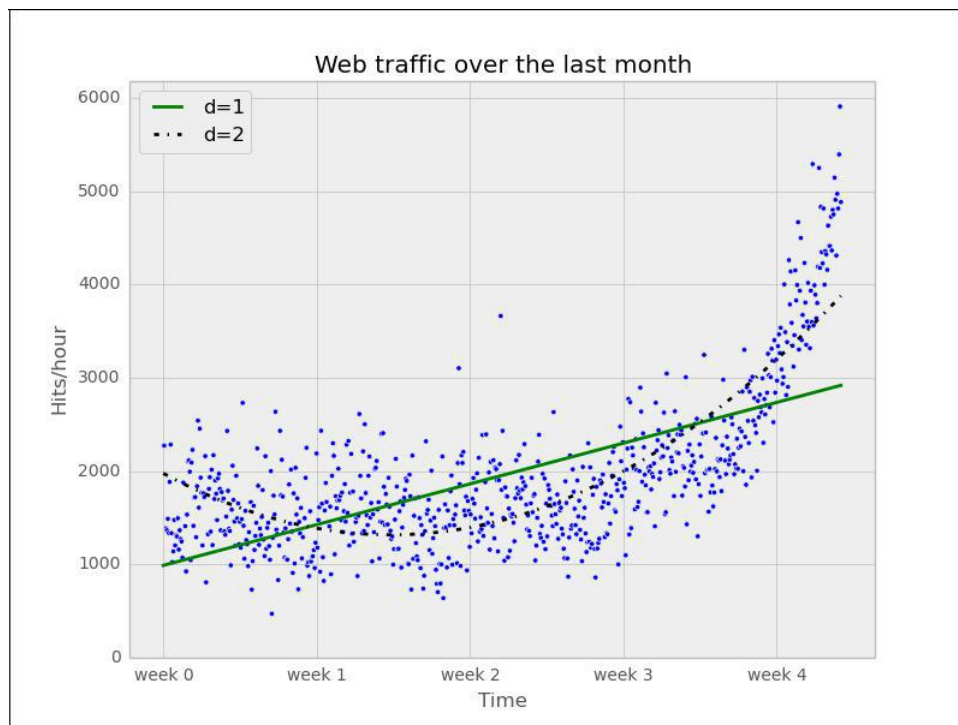
```
array( [ 1.05322215e-02 , -5.26545650e+00 , 1.97476082e+03 ] )
```

- `f2 = sp.poly1d(f2p)`

- `print(error(f2, x, y))`

```
179983507.878
```

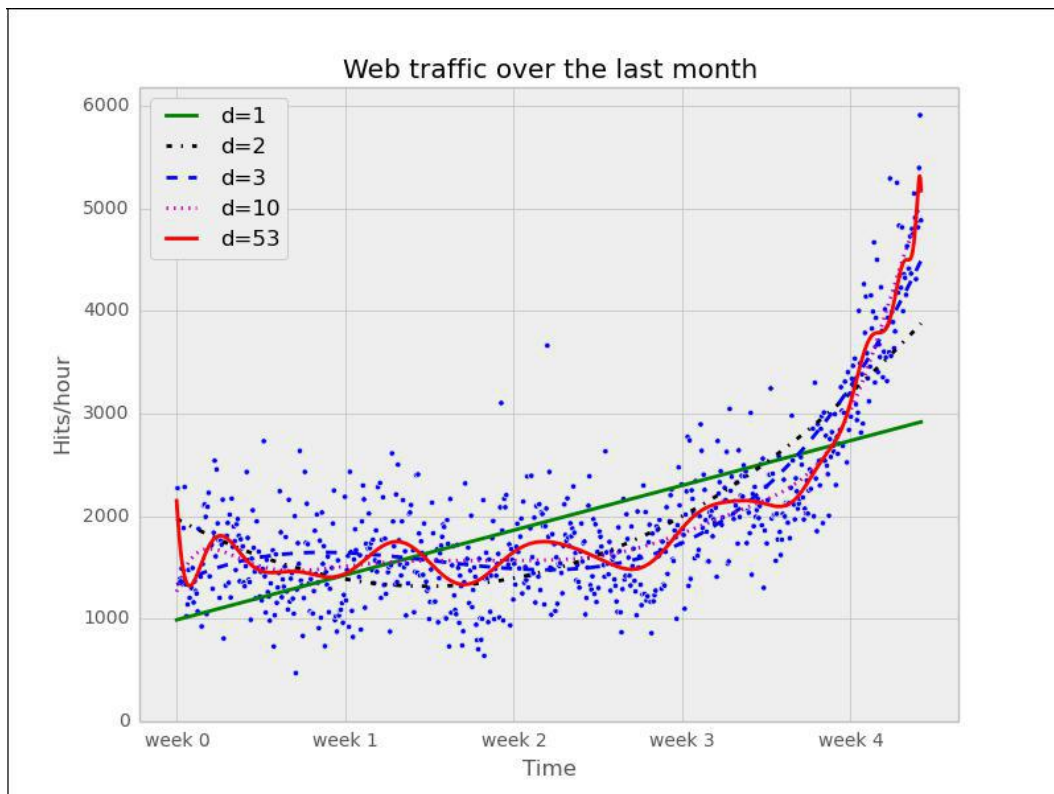
You will get the following plot:



The error is 179,983,507.878, which is almost half the error of the straight line model. This is good but unfortunately this comes with a price: We now have a more complex function, meaning that we have one parameter more to tune inside `polyfit()`. The fitted polynomial is as follows:

$$f(x) = 0.0105322215 * x^{**2} - 5.26545650 * x + 1974.76082$$

So, if more complexity gives better results, why not increase the complexity even more? Let's try it for degrees 3, 10, and 100.



Interestingly, we do not see  $d=53$  for the polynomial that had been fitted with 100 degrees.

Instead, we see lots of warnings on the console:

## RankWarning: Polyfit may be poorly conditioned

This means because of numerical errors, polyfit cannot determine a good fit with 100 degrees. Instead, it figured that 53 must be good enough. It seems like the curves capture and better the fitted data the more complex they get. And also, the errors seem to tell the same story:

**Error d=1: 317,389,767.339778**

**Error d=2: 179,983,507.878179**

**Error d=3: 139,350,144.031725**

**Error d=10: 121,942,326.363461**

**Error d=53: 109,318,004.475556**

However, taking a closer look at the fitted curves, we start to wonder whether they also capture the true process that generated that data. Framed differently, do our models correctly represent the underlying mass behavior of customers visiting our website? Looking at the polynomial of degree 10 and 53, we see wildly oscillating behavior. It seems that the models are fitted too much to the data. So much that it is now capturing not only the underlying process but also the noise. This is called **overfitting**.

At this point, we have the following choices:

- Choosing one of the fitted polynomial models.
- Switching to another more complex model class. Splines?
- Thinking differently about the data and start again.

Out of the five fitted models, the first order model clearly is too simple, and the models of order 10 and 53 are clearly overfitting. Only the second and third order models seem to somehow match the data. However, if we extrapolate them at both borders, we see them going berserk.

Switching to a more complex class seems also not to be the right way to go. What arguments would back which class? At this point, we realize that we probably have not fully understood our data.

## Stepping back to go forward – another look at our data

So, we step back and take another look at the data. It seems that there is an inflection point between weeks 3 and 4. So let's separate the data and train two lines using week 3.5 as a separation point:

```
inflection = 3.5*7*24 # calculate the inflection point in hours
```

```
xa = x[:inflection] # data before the inflection point
```

```
ya = y[:inflection]
```

```
xb = x[inflection:] # data after
```

```
yb = y[inflection:]
```

```
fa = sp.poly1d(sp.polyfit(xa, ya, 1))
```

```
fb = sp.poly1d(sp.polyfit(xb, yb, 1))
```

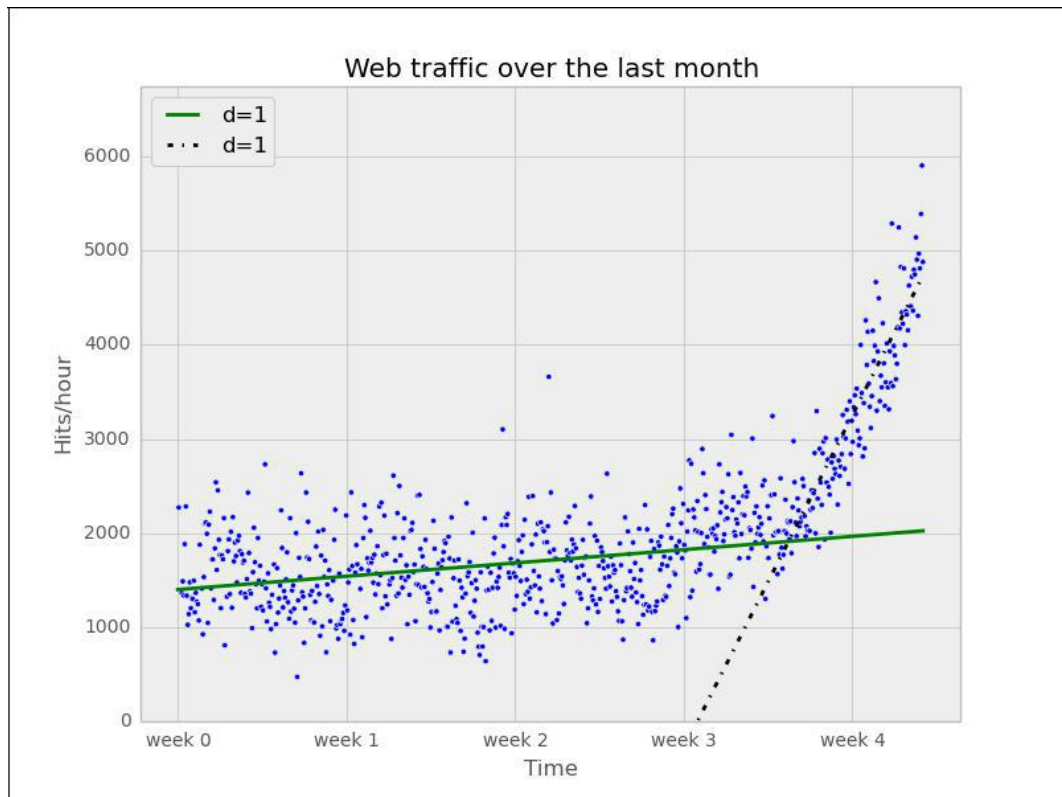
```
fa_error = error(fa, xa, ya)
```

```
fb_error = error(fb, xb, yb)
```

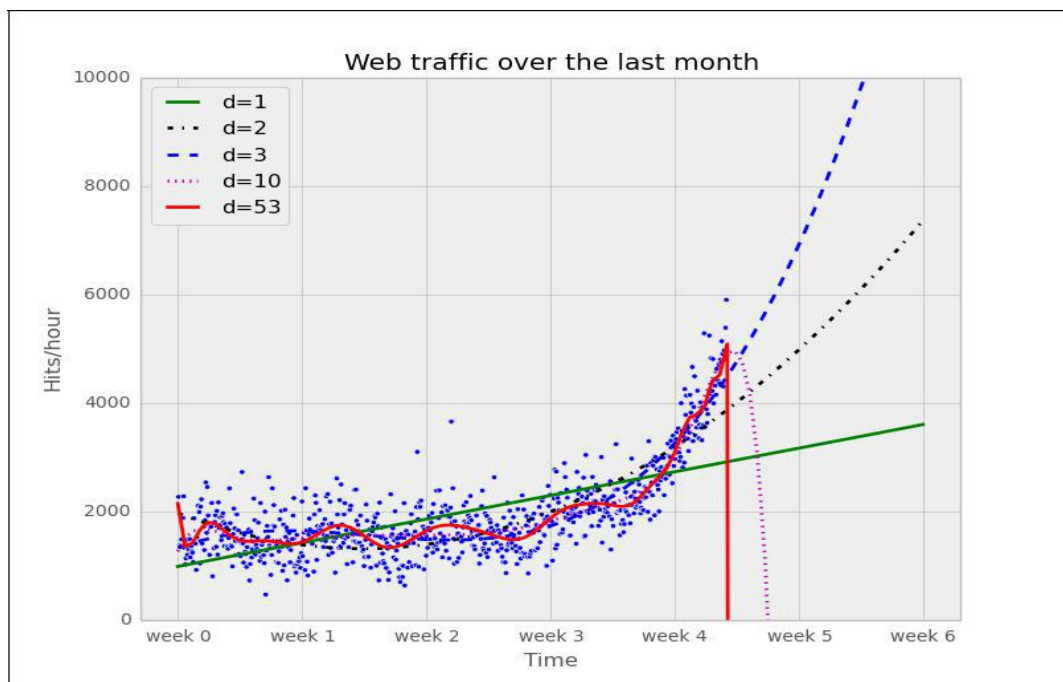
```
print("Error inflection=%f" % (fa_error + fb_error))
```

```
Error inflection=132950348.197616
```

From the first line, we train with the data up to week 3, and in the second line we train with the remaining data.

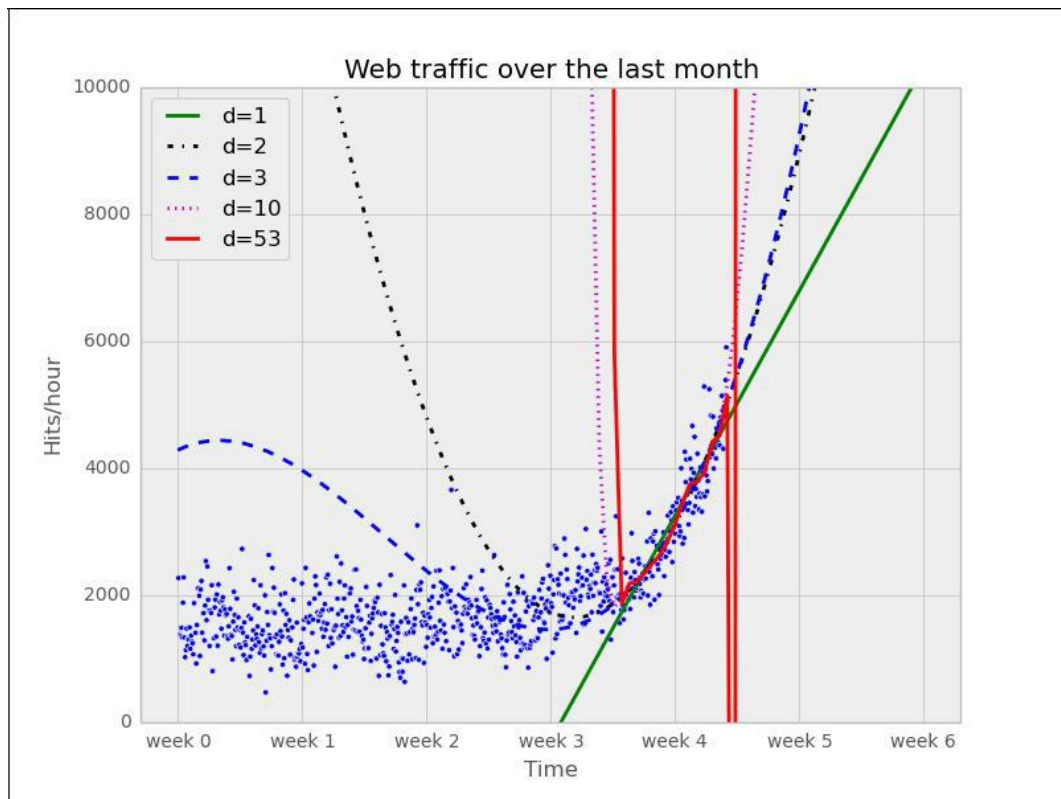


Clearly, the combination of these two lines seems to be a much better fit to the data than anything we have modeled before. But still, the combined error is higher than the higher order polynomials. Can we trust the error at the end?



The models of degree 10 and 53 don't seem to expect a bright future of our start-up. They tried so hard to model the given data correctly that they are clearly useless to extrapolate beyond. This is called overfitting. On the other hand, the lower degree models seem not to be capable of capturing the data good enough. This is called **underfitting**.

So let's play fair to models of degree 2 and above and try out how they behave if we fit them only to the data of the last week. After all, we believe that the last week says more about the future than the data prior to it. The result can be seen in the following psychedelic chart, which further shows how badly the problem of overfitting is.



Still, judging from the errors of the models when trained only on the data from week 3.5 and later, we still should choose the most complex one (note that we also calculate the error only on the time after the inflection point):

**Error d=1: 22,143,941.107618**

**Error d=2: 19,768,846.989176**

**Error d=3: 19,766,452.361027**

**Error d=10: 18,949,339.348539**

**Error d=53: 18,300,702.038119**

## **Training and testing**

If we only had some data from the future that we could use to measure our models against, then we should be able to judge our model choice only on the resulting approximation error.

Although we cannot look into the future, we can and should simulate a similar effect by holding out a part of our data. Let's remove, for instance, a certain percentage of the data and train on the remaining one. Then we used the held-out data to calculate the error. As the model has been trained not knowing the held-out data, we should get a more realistic picture of how the model will behave in the future.

The test errors for the models trained only on the time after inflection point now show a completely different picture:

**Error d=1: 6397694.386394**

**Error d=2: 6010775.401243**

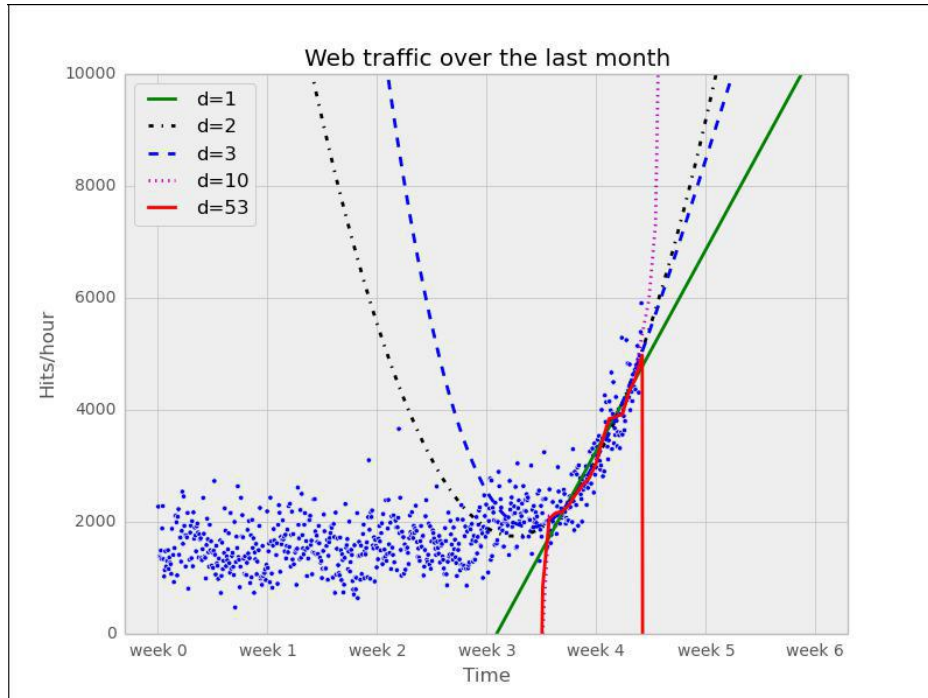
**Error d=3: 6047678.658525**

**Error d=10: 7037551.009519**

**Error d=53: 7052400.001761**



Have a look at the following plot:



It seems that we finally have a clear winner: The model with degree 2 has the lowest test error, which is the error when measured using data that the model did not see during training. And this gives us hope that we won't get bad surprises when future data arrives.

## **RESULTS AND DISCUSSION**

Finally we have arrived at a model which we think represents the underlying process best; it is now a simple task of finding out when our infrastructure will reach 100,000 requests per hour.

We have to calculate when our model function reaches the value 100,000.

Having a polynomial of degree 2, we could simply compute the inverse of the function and calculate its value at 100,000. Of course, we would like to have an approach that is applicable to any model function easily.

This can be done by subtracting 100,000 from the polynomial, which results in another polynomial, and finding its root. SciPy's optimize module has the function `fsolve` that achieves this, when providing an initial starting position with parameter `x0`. As every entry in our input data file corresponds to one hour, and we have 743 of them, we set the starting position to some value after that. Let `fbt2` be the winning polynomial of degree 2.

- **`fbt2 = sp.poly1d(sp.polyfit(xb[train], yb[train], 2))`**
- **`print("fbt2(x)= \n%s" % fbt2)`**
- **`print("fbt2(x)-100,000= \n%s" % (fbt2-100000))`**
- **`from scipy.optimize import fsolve`**
- **`reached_max = fsolve(fbt2-100000, x0=800)/(7*24)`**
- **`print("100,000 hits/hour expected at week %f" % reached_max[0])`**

It is expected to have 100,000 hits/hour at week 9.616071, so our model tells us that, given the current user behavior and traction of our start-up, it will take another month until we have reached our capacity threshold.

## **CONCLUSION**

Of course, there is a certain uncertainty involved with my prediction. To get a real picture of it, one could draw in more sophisticated statistics to find out about the variance we have to expect when looking farther and farther into the future.

And then there are the user and underlying user behavior dynamics that we cannot model accurately. However, at this point, I'm fine with the current predictions. After all, I can prepare all time-consuming actions now. If I monitor my web traffic closely, I will see in time when I have to allocate new resources.

## **References**

### **Books**

- [1] Sebastian Raschica, Python Machine Learning, 2nd ed. Brimingham, U.K : Packt Publishing, September 2015 .
- [2] L. Stein, *Computers and You*, J. S. Brake, Ed. New York, USA: Wiley, 1994.
- [3] M. Abramowitz and I. A. Stegun, Eds., *Handbook of Mathematical Functions* (Applied Mathematics Series 55). Washington, DC, USA: NBS, 1964.

### **Online Sources**

#### ***FTP***

- [1] <https://github.com/luispedro/BuildingMachineLearningSystemsWithPython/tree/master/ch01/data>

#### ***WWW***

- [1] [https://en.wikipedia.org/wiki/Python\\_\(programming\\_language\)](https://en.wikipedia.org/wiki/Python_(programming_language))
- [2] <https://en.wikipedia.org/wiki/Numpy>
- [3] [https://en.wikipedia.org/wiki/Pandas\\_\(software\)](https://en.wikipedia.org/wiki/Pandas_(software))
- [4] <https://en.wikipedia.org/wiki/Matplotlib>
- [5] <https://en.wikipedia.org/wiki/Scipy>
- [6] <https://en.wikipedia.org/wiki/Scikit-learn>