Spring 2013

Big Data Analysis Using Amazon Web Services and Support Vector Machines

Dhruv Jalota
San Jose State University

Big Data Analysis Using Amazon Web Services and Support Vector Machines

A Writing Project

Presented To

The Faculty of the Department of Computer Science

San Jose State University

In Partial Fulfillment

of the Requirements for the Degree

Master of Science

by

Dhruv Jalota

May 2013

APPROVED FOR THE DEPARTMENT OF COMPUTER SCIENCE

Dr. Chris Tseng

Professor of Computer Science, San José State University

Dr. Chris Pollett

Professor of Computer Science, San José State University

Mr. Sumeet Trehan

Energy Resources Engineering, Stanford University

APPROVED FOR THE UNIVERSITY

ABSTRACT

Big Data Analysis Using Amazon Web Services and Support Vector Machines

by Dhruv Jalota

This writing project aims to apply the supervised machine learning technique known as Support Vector Machines to a large labeled data set, to attempt to classify an unlabeled data set using the result of training on the labeled data set, and hence perform an analysis of the various results obtained using different Amazon Elastic Cloud Compute instances, sizes of input data set, and different parameters or kernels of the SVM tool. The given data set is relatively large for SVM and the tool being used, known as libsvm, having approximately 1.3 million training examples and 341 attributes with binary classification labels i.e., true (+1) and false (-1). By using the open source tool and deploying it to the cloud, we make use of the computing power available, to get the best possible results for classification. We eventually give a detailed analysis of the performance of all the experiments conducted, and draw conclusions from these results.

ACKNOWLEDGMENT

I would like to thank Dr. Tseng for his technical guidance and his constant support through out my writing project. Dr. Tseng always gave me kind words even through the toughest time. I would also like to thank Dr. Pollett and Mr. Sumeet Trehan, for being my committee members and helping me through this process.

TABLE OF CONTENTS

List of Tables and Figures

1. Problem Description

1.1 Description of Data Set

The two data sets were provided to us by IBM as part of 'The Great Minds Challenge'. The entire data set is basically a part of what is actually used by IBM's Watson supercomputer, which was the same one which participated on the 'Jeopardy!' television show, for learning purposes. The first data set that we have, has 1,314,407 rows and 343 columns. These 1.3 million rows are known as feature vectors or training examples and the 343 columns are known as features or attributes. Of these 343 columns, the first column is a "question id" and the last column is a binary label i.e. a "true" or "false" classification.

According to IBM's description provided for this data set [1], "For each question that Watson answers many possible answers are generated using standard information retrieval techniques. These "candidate answers" along with the corresponding question are fed to a series of "scorers" that evaluate the likelihood that the answer is a correct one. These "features" are then fed into a machine learning algorithm that learns how to appropriately weight these features. Each row in the file represents a possible answer to a question. The row contains the question identifier (i.e. the question that it was a candidate answer to), the feature scores and it also contains a label indicating whether it is the right answer. The vast majority of rows in the file are for wrong answers with a smaller percentage being the correct answer. The file is in CSV format and is a comma delimited list of feature scores. The two important "columns" in the file are the first column that contains a unique question id and the last column that contains the label. Candidate answers to the same question share a common question id. The label is true for a right answer and false for an incorrect answer. Note that some questions may not have a correct answer."

This file of labeled data was provided to us in a plain text format of size 3.23 GB that contains the data in CSV format.

1.2 Brief Introduction to Amazon Web Services

What is Amazon Web Services or AWS as it is popularly known? Well, recently the terms “Cloud Computing” and “Big Data” have gained tremendous importance due to their significance to computing and analytical power growing exponentially as compared with the scenarios of the past where large scientific data crunching could be achieved only by the likes of super computers. In today’s world, powerful computing architecture can be rented out on a pay-per-use basis from companies like Amazon (AWS is one of the many such services) for processing complex data of high importance. This “cloud” as it is called, is where the computing is done, since the user never sees it himself, but only accesses it over the internet, and the data that is being run is “big” in the sense of either size or complexity and importance.

The wide-spread use and adoption of these services has made it affordable to organizations and individuals that require it, and AWS is among the most popular of these services. AWS offers many products including computing, networking, content delivery, and storage etc. [2] We will be focussing on the computing service offered that is known as Elastic Cloud Compute or EC2.

We have used EC2 instances to store and analyze our data for the classification purposes. This table [3] provides a comparison of the various EC2 options available. We will talk about these configurations in detail in a later section.

Name	Memory	Compute Units	Linux cost
Micro	0.60 GB	2 (only for short bursts)	$0.02 hourly
High-CPU Medium	1.70 GB	5 (2 cores x 2.5 units)	$0.14 hourly
M1 Small	1.70 GB	1 (1 core x 1 unit)	$0.06 hourly
M1 Medium	3.75 GB	2 (1 core x 2 units	$0.12 hourly
High-CPU Extra Large	7.00 GB	20 (8 cores x 2.5 units)	$0.58 hourly
M1 Large	7.50 GB	4 (2 cores x 2 units)	$0.24 hourly
M1 Extra Large	15.00 GB	8 (4 cores x 2 units)	$0.48 hourly
M3 Extra Large	15.00 GB	13 (4 cores x 3.25 units)	$0.50 hourly
High-Memory Extra Large	17.10 GB	6.5 (2 cores x 3.25 units)	$0.41 hourly
Cluster GPU Quadruple Extra Large	22.00 GB	33.5 (2 x Intel Xeon X5570)	$2.10 hourly
Cluster Compute Quadruple Extra Large	23.00 GB	33.5 (2 x Intel Xeon X5570)	$1.30 hourly
M3 Double Extra Large	30.00 GB	26 (8 cores x 3.25 units)	$1.00 hourly
High-Memory Double Extra Large	34.20 GB	13 (4 cores x 3.25 units)	$0.82 hourly
Cluster Compute Eight Extra Large	60.50 GB	88 (2 x Intel Xeon E5-2670)	$2.40 hourly
High I/O Quadruple Extra Large	60.50 GB	35 (8 cores + 8 hyperthreads)	$3.10 hourly
High-Memory Quadruple Extra Large	68.40 GB	26 (8 cores x 3.25 units)	$1.64 hourly

Name	Memory	Compute Units	Linux cost
High Storage Eight Extra Large	117.00 GB	35 (8 cores + 8 hyperthreads)	$4.60 hourly
High Memory Cluster Eight Extra Large	244.00 GB	88 (2 x Intel Xeon E5-2670)	$3.50 hourly

Table 1. Comparison of EC2 instances [3]

In the table above, one EC2 compute unit is equivalent to a 1-1.2 GHz Intel 2007 Xeon or Opteron processor. [6]

As we can see there are currently 18 different EC2 instance types available, but not all are affordable or suitable for this project. Initially, in an effort to keep costs low, I used the "M1 Large" machine, since it's RAM was almost double of that available on my laptop, but it was unable to process the large data sets in a relatively short amount of time .
From observation, I saw that the libsvm tool used up to 25% of RAM for each instance of it's process, and 99% CPU. Using the highest level of EC2 machine types was not feasible due to the budget limitation of $100 by the Amazon educational grant funds per instance, and hence at rates of $4 per hour it could be used for only 25 hours. So I settled on the mid-tier machine "M3 Double Extra Large" due to it's higher RAM and an affordable hourly rate which allowed me to run it for 100 hours each.

1.3 Brief Introduction to Support Vector Machines

Support Vector Machines, or SVM, is a supervised learning technique in machine learning that has gained popularity over the last 2 decades. It was originally invented by Vladmir N. Vapnik, and the now standard implementation was created by Vapnik and Corrina Cortes [4]. It has proven to be quite useful in classification tasks, particularly for the case of binary classes like ours, giving the best possible success rate [5].

Essentially, SVM, which is also called a “large margin classifier”, establishes an optimal separating hyperplane to divide points in the feature space into one of two classes (Figure 1). This is typical for linearly separable data, and can be susceptible to the “outliers” case (Figure 2). To enhance this method for non-linearly separable data, SVMs use a technique known as the “kernel trick” which establishes the separating hyperplane for data in high-dimensional feature spaces.

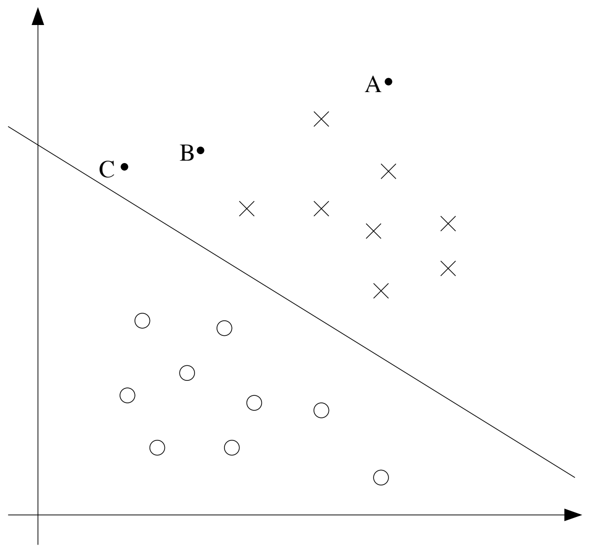


Figure 1. Linear Separating Hyperplane [5]

In the figure above, we see the separating hyperplane which has divided the points in either of the two classes (crosses and circles), and hence established positive or negative classifications respectively. In this case, when new points “A”, “B” and “C” are introduced to the system, it is relatively more confident about the classification for A than it is for B, and C has the least confidence. But this can be sensitive to the outliers case, as seen below.

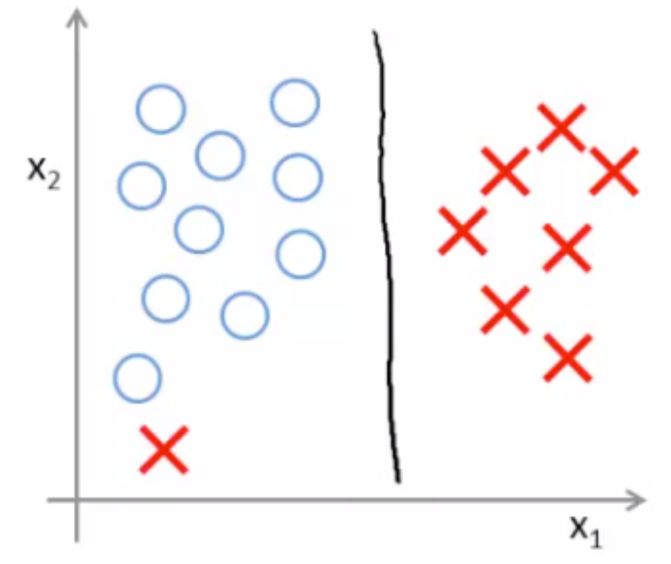


Figure 2. The “outlier” case. [7]

Here, though the data has been linearly separated by the hyperplane, there is one point which falls in the wrong category, and this is an outlier. Such a point would cause drastic changes to the hyperplane, and would not be as optimally spaced in terms of margins as before (Figure 3).

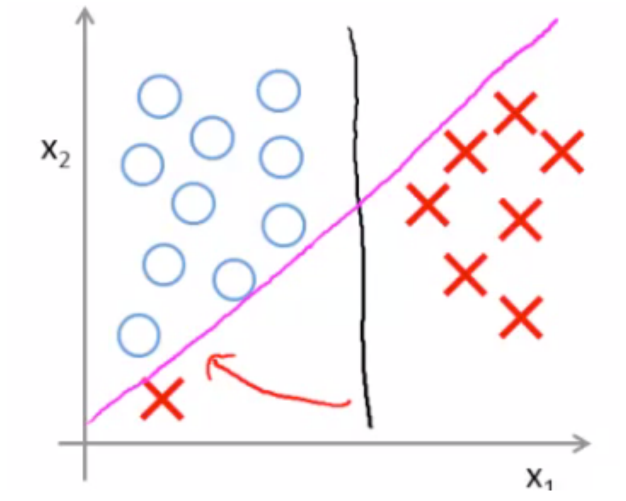


Figure 3. Outlier causes hyperplane to shift drastically. [7]

There is a link between SVMs and Perceptrons, due to the linear classifier concept. [4]

2. Amazon Web Services (AWS)

2.1 Advantages of AWS

Why are we using the AWS services here, especially since it can be expensive for high-power machines? When the project began, I experimented on my own laptop, which is a MacBook Pro with a 2GHz Intel Core i7 processor and 4GB of memory. After researching the several tools available to apply SVM and trying out a few, MATLAB prove to be the most robust for our purposes. This was mainly due to the size of the data. As we shall see later, this was one of the major challenges in this project, and crossing this hurdle in phase 1 was achieved by using MATLAB.

Other than MATLAB, the rest of the tools out there are mostly open-source, and they have their own requirements for the format of the data to be input. MATLAB was the only one which could read the CSV file directly (the format in which we received the data) and produce a matrix in it's workspace for our manipulation. So once the entire data set (3.2GB raw size) was loaded into memory, performing the required matrix manipulations and using the SVM toolbox with it, would drastically reduce the performance of the machine, and dealing with machine crashes and hangups was the major frustration.

This is where AWS came to the rescue, since it gave us a relatively higher powered machine (we went with the mid-tier machines, ranging in cost from \$0.40 to \$1.00 per hour), it was able to deal with the large data set with ease and we could better conduct our SVM analysis of the data.

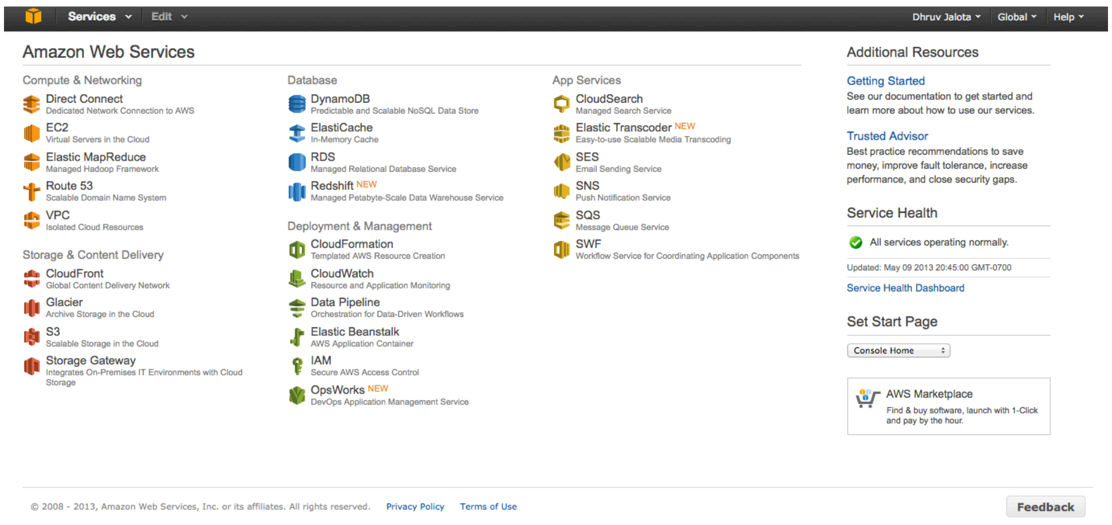
2.2 Configuration of Amazon Elastic Cloud Compute (EC2)

For my project, after trying out several of the cheaper machines available, I settled on the "M3 Double Extra Large" machines since it was not too expensive, and gave me enough memory to be able to run a lot of training and classification experiments simultaneously, which as we shall see later was very time consuming.

I will now walk through the process of setting up a machine in the cloud, or launching an instance as it is known.

First, you need an Amazon AWS account to be able to use their services. You can sign up for this by going to the web site http://aws.amazon.com and clicking on the "Sign Up" button.

After signing up with an email address and credit card, Amazon conducts and automated identity verification by calling your phone number, and giving you a unique PIN which you use to get access to your account on first time use only. Once these initial steps are complete, you are greeted by your AWS dashboard (or console) as seen below.



Amazon Web Services

Compute & Networking
- Direct Connect — Dedicated Network Connection to AWS
- EC2 — Virtual Servers in the Cloud
- Elastic MapReduce — Managed Hadoop Framework
- Route 53 — Scalable Domain Name System
- VPC — Isolated Cloud Resources

Storage & Content Delivery
- CloudFront — Global Content Delivery Network
- Glacier — Archive Storage in the Cloud
- S3 — Scalable Storage in the Cloud
- Storage Gateway — Integrates On-Premises IT Environments with Cloud Storage

Database
- DynamoDB — Predictable and Scalable NoSQL Data Store
- ElastiCache — In-Memory Cache
- RDS — Managed Relational Database Service
- Redshift NEW — Managed Petabyte-Scale Data Warehouse Service

Deployment & Management
- CloudFormation — Templated AWS Resource Creation
- CloudWatch — Resource and Application Monitoring
- Data Pipeline — Orchestration for Data-Driven Workflows
- Elastic Beanstalk — AWS Application Container
- IAM — Secure AWS Access Control
- OpsWorks NEW — DevOps Application Management Service

App Services
- CloudSearch — Managed Search Service
- Elastic Transcoder NEW — Easy-to-use Scalable Media Transcoding
- SES — Email Sending Service
- SNS — Push Notification Service
- SQS — Message Queue Service
- SWF — Workflow Service for Coordinating Application Components

Additional Resources

Getting Started
See our documentation to get started and learn more about how to use our services.

Trusted Advisor
Best practice recommendations to save money, improve fault tolerance, increase performance, and close security gaps.

Service Health

✓ All services operating normally.

Updated: May 09 2013 20:45:00 GMT-0700

Service Health Dashboard

Set Start Page

Console Home

AWS Marketplace
Find & buy software, launch with 1-Click and pay by the hour.

© 2008 - 2013, Amazon Web Services, Inc. or its affiliates. All rights reserved. Privacy Policy Terms of Use

Feedback

Figure 4. AWS console.

We can see here on the console screen, the various services offered by AWS. We are interested in the EC2 service, as seen on the left side of the screen. To launch an instance, we click on the EC2 link. By default, AWS picks the “U.S. East” zone for where the servers are hosted, as seen by the “N. Virginia” text in the top right corner. You can choose any zone you prefer, but the cost of instances in this zone are generally cheaper. The various zones available are as shown below.

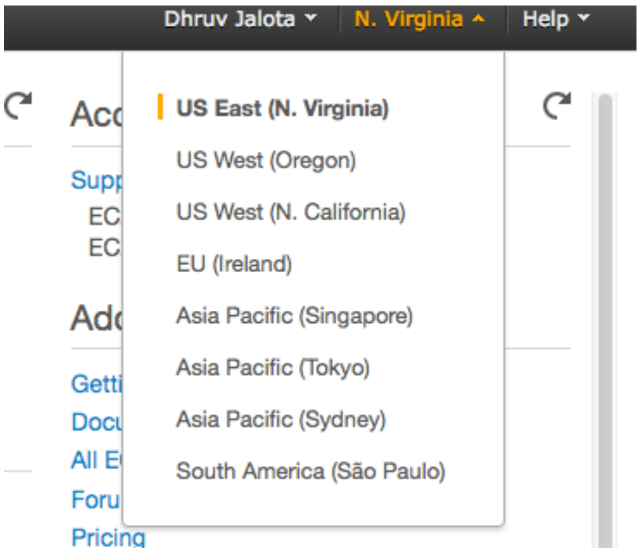
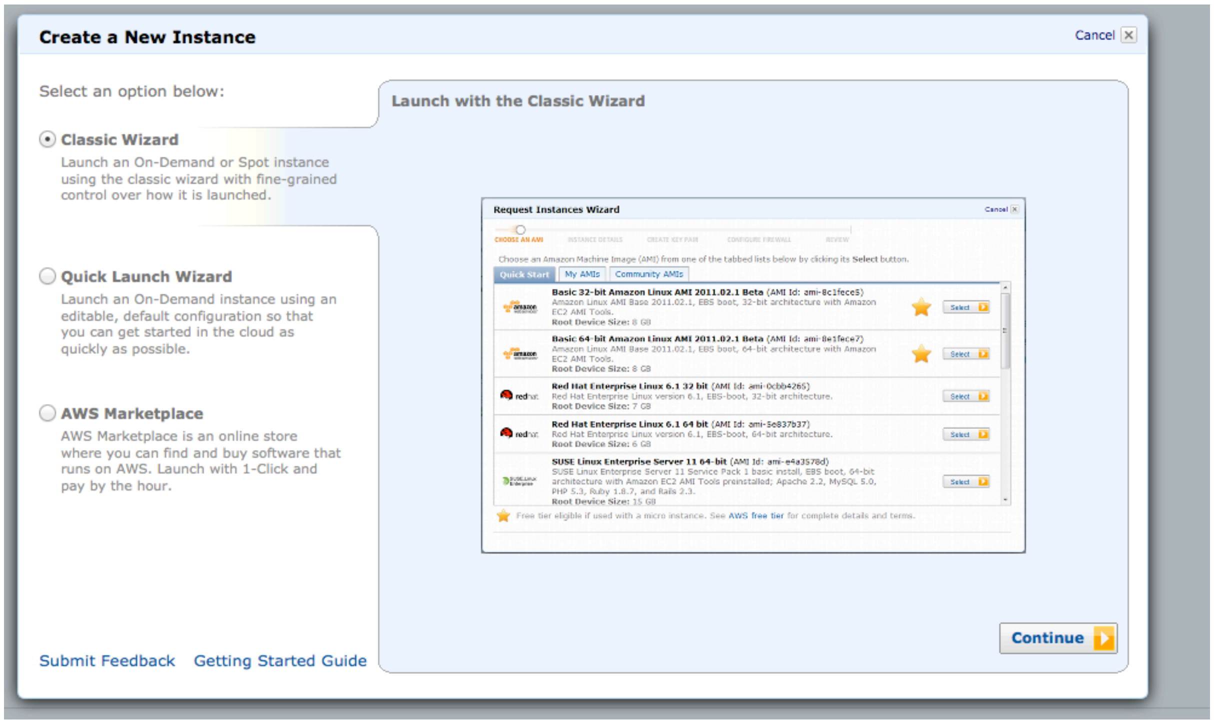


Figure 5. AWS zones.

Now, here is how we "Launch an instance".



Select an option below:

(●) **Classic Wizard**
Launch an On-Demand or Spot instance using the classic wizard with fine-grained control over how it is launched.

( ) **Quick Launch Wizard**
Launch an On-Demand instance using an editable, default configuration so that you can get started in the cloud as quickly as possible.

( ) **AWS Marketplace**
AWS Marketplace is an online store where you can find and buy software that runs on AWS. Launch with 1-Click and pay by the hour.

Launch with the Classic Wizard

Request Instances Wizard

CHOOSE AN AMI — INSTANCE DETAILS — CREATE KEY PAIR — CONFIGURE FIREWALL — REVIEW

Choose an Amazon Machine Image (AMI) from one of the tabbed lists below by clicking its **Select** button.

Quick Start | My AMIs | Community AMIs

Basic 32-bit Amazon Linux AMI 2011.02.1 Beta (AMI Id: ami-8c1fece5)
Amazon Linux AMI Base 2011.02.1, EBS boot, 32-bit architecture with Amazon EC2 AMI Tools.
Root Device Size: 8 GB

Basic 64-bit Amazon Linux AMI 2011.02.1 Beta (AMI Id: ami-8e1fece7)
Amazon Linux AMI Base 2011.02.1, EBS boot, 64-bit architecture with Amazon EC2 AMI Tools.
Root Device Size: 8 GB

Red Hat Enterprise Linux 6.1 32 bit (AMI Id: ami-0cbb4265)
Red Hat Enterprise Linux version 6.1, EBS-boot, 32-bit architecture.
Root Device Size: 7 GB

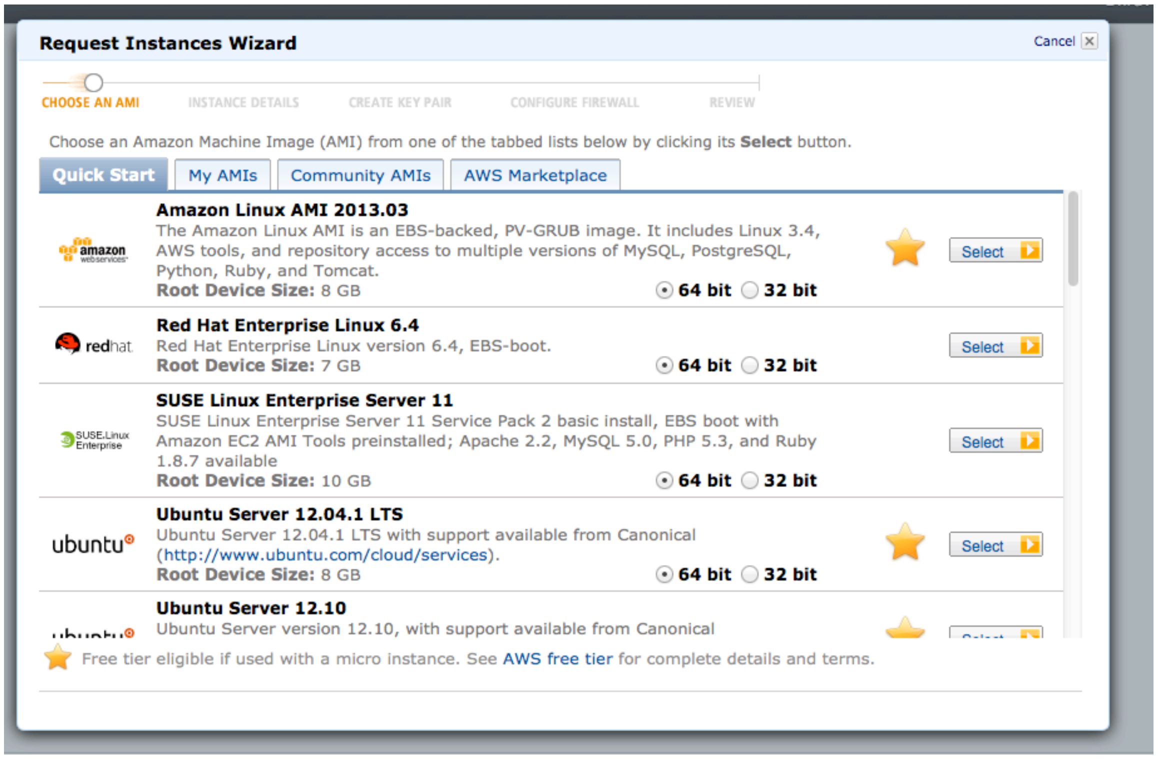
Red Hat Enterprise Linux 6.1 64 bit (AMI Id: ami-5e837b37)
Red Hat Enterprise Linux version 6.1, EBS-boot, 64-bit architecture.
Root Device Size: 6 GB

SUSE Linux Enterprise Server 11 64-bit (AMI Id: ami-e4a3578d)
SUSE Linux Enterprise Server 11 Service Pack 1 basic install, EBS boot, 64-bit architecture with Amazon EC2 AMI Tools preinstalled; Apache 2.2, MySQL 5.0, PHP 5.3, Ruby 1.8.7, and Rails 2.3.
Root Device Size: 15 GB

Free tier eligible if used with a micro instance. See AWS free tier for complete details and terms.

Figure 6. Launch an instance.

We select “Classic Wizard” and click on “Continue” (Figure 6). On the next screen we select the first option, with 64-bit, and then press “Select” (Figure 7). Next, we select the “m3.2xlarge” option from the drop-down list, and click on “Continue (Figure 8). Next, click on “Continue” (Figure 9).



Choose an Amazon Machine Image (AMI) from one of the tabbed lists below by clicking its **Select** button.

Amazon Linux AMI 2013.03
The Amazon Linux AMI is an EBS-backed, PV-GRUB image. It includes Linux 3.4, AWS tools, and repository access to multiple versions of MySQL, PostgreSQL, Python, Ruby, and Tomcat.
Root Device Size: 8 GB [✓ 64 bit] 32 bit

Red Hat Enterprise Linux 6.4
Red Hat Enterprise Linux version 6.4, EBS-boot.
Root Device Size: 7 GB [✓ 64 bit] 32 bit

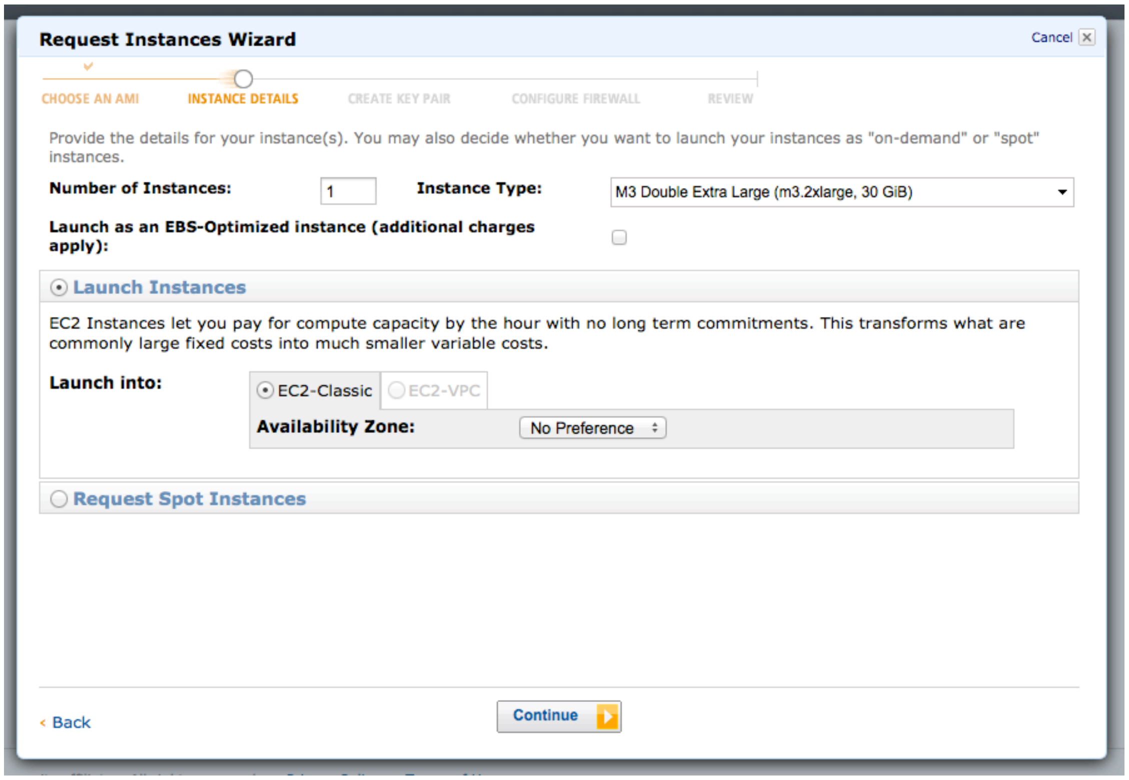
SUSE Linux Enterprise Server 11
SUSE Linux Enterprise Server 11 Service Pack 2 basic install, EBS boot with Amazon EC2 AMI Tools preinstalled; Apache 2.2, MySQL 5.0, PHP 5.3, and Ruby 1.8.7 available
Root Device Size: 10 GB [✓ 64 bit] 32 bit

Ubuntu Server 12.04.1 LTS
Ubuntu Server 12.04.1 LTS with support available from Canonical (http://www.ubuntu.com/cloud/services).
Root Device Size: 8 GB [✓ 64 bit] 32 bit

Ubuntu Server 12.10
Ubuntu Server version 12.10, with support available from Canonical

Free tier eligible if used with a micro instance. See AWS free tier for complete details and terms.

Figure 7. Launch an instance step 2.



Provide the details for your instance(s). You may also decide whether you want to launch your instances as "on-demand" or "spot" instances.

Number of Instances: 1 **Instance Type:** M3 Double Extra Large (m3.2xlarge, 30 GiB)

Launch as an EBS-Optimized instance (additional charges apply): ☐

[✓ Launch Instances]

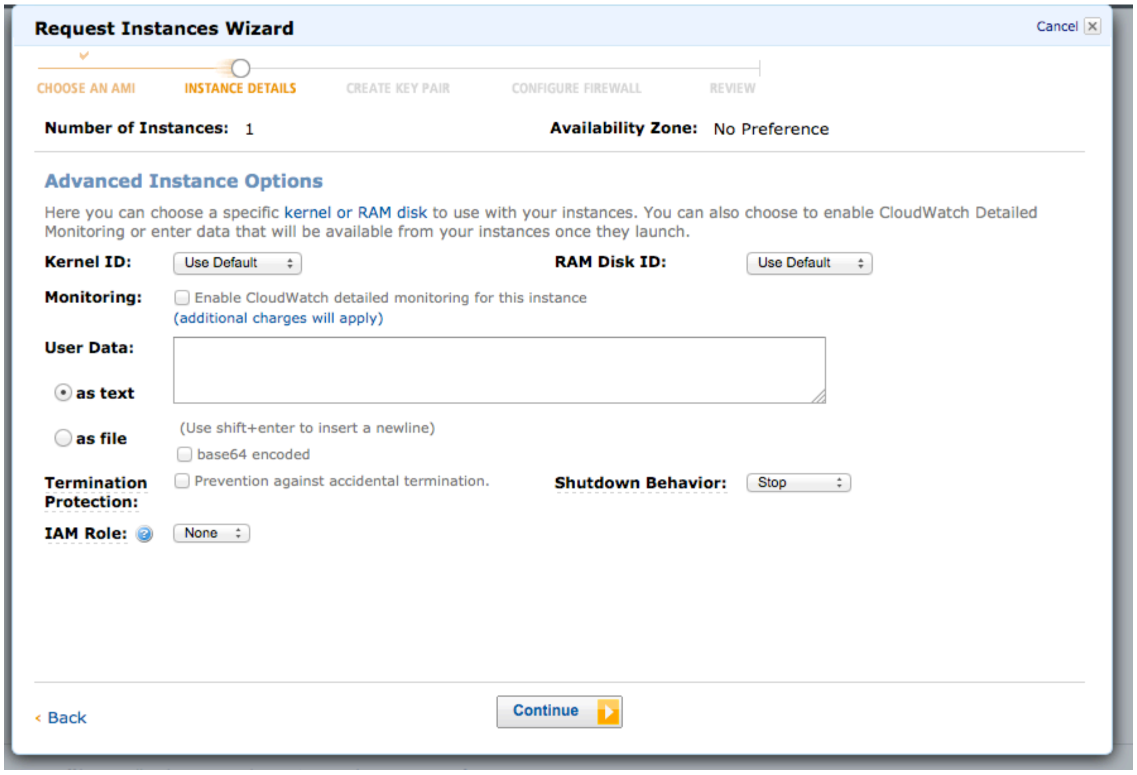
EC2 Instances let you pay for compute capacity by the hour with no long term commitments. This transforms what are commonly large fixed costs into much smaller variable costs.

Launch into: [✓ EC2-Classic] EC2-VPC

Availability Zone: No Preference

Request Spot Instances

Figure 8. Launch an instance step 3.



Number of Instances: 1 **Availability Zone:** No Preference

Advanced Instance Options

Here you can choose a specific kernel or RAM disk to use with your instances. You can also choose to enable CloudWatch Detailed Monitoring or enter data that will be available from your instances once they launch.

Kernel ID: Use Default **RAM Disk ID:** Use Default

Monitoring: ☐ Enable CloudWatch detailed monitoring for this instance (additional charges will apply)

User Data:

[✓ as text] as file

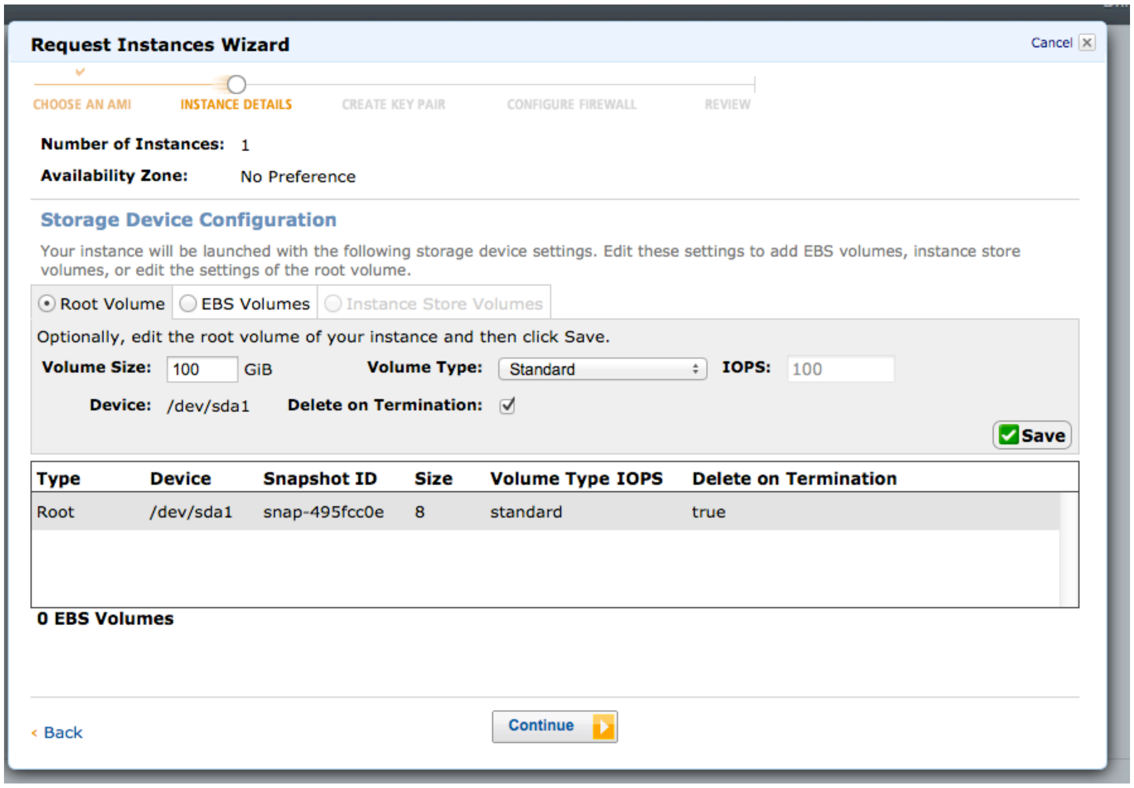
(Use shift+enter to insert a newline)

☐ base64 encoded

Termination Protection: ☐ Prevention against accidental termination. **Shutdown Behavior:** Stop

IAM Role: None

Figure 9. Launch an instance step 4.



Number of Instances: 1

Availability Zone: No Preference

Storage Device Configuration

Your instance will be launched with the following storage device settings. Edit these settings to add EBS volumes, instance store volumes, or edit the settings of the root volume.

Root Volume [✓] | EBS Volumes | Instance Store Volumes

Optionally, edit the root volume of your instance and then click Save.

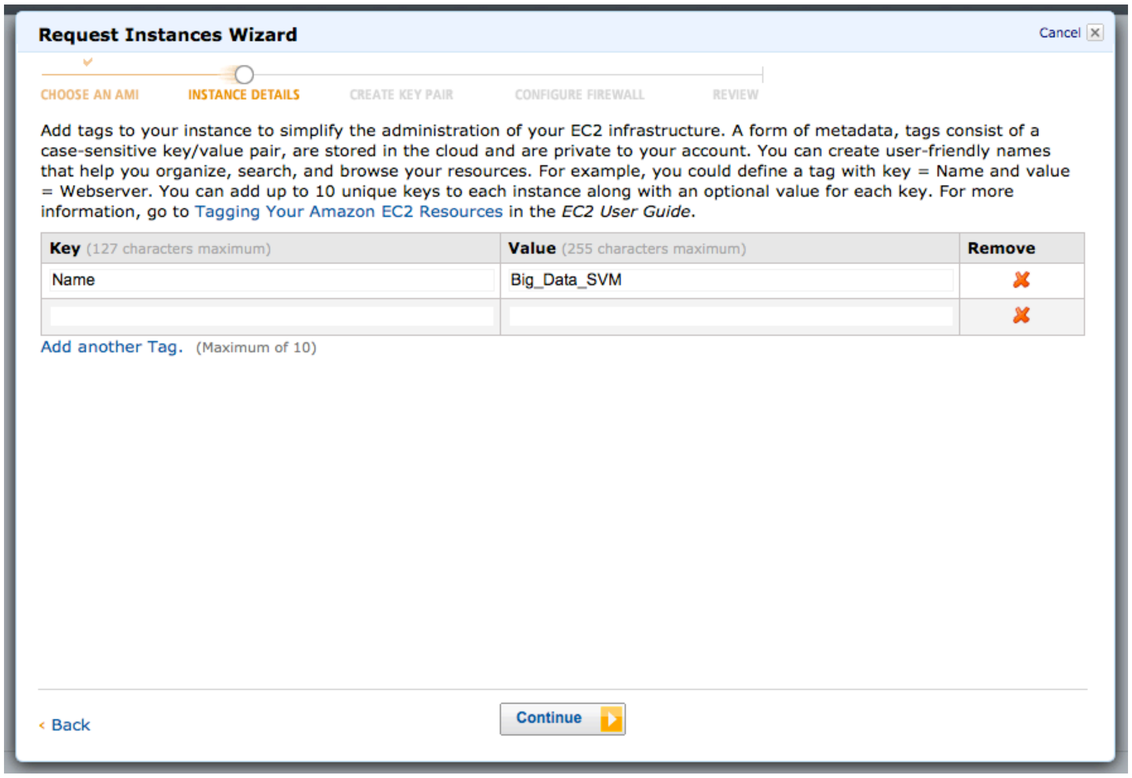
Volume Size: 100 GiB **Volume Type:** Standard **IOPS:** 100

Device: /dev/sda1 **Delete on Termination:** ✓

Type	Device	Snapshot ID	Size	Volume Type	IOPS	Delete on Termination
Root	/dev/sda1	snap-495fcc0e	8	standard		true

0 EBS Volumes

Figure 10. Launch an instance step 5.

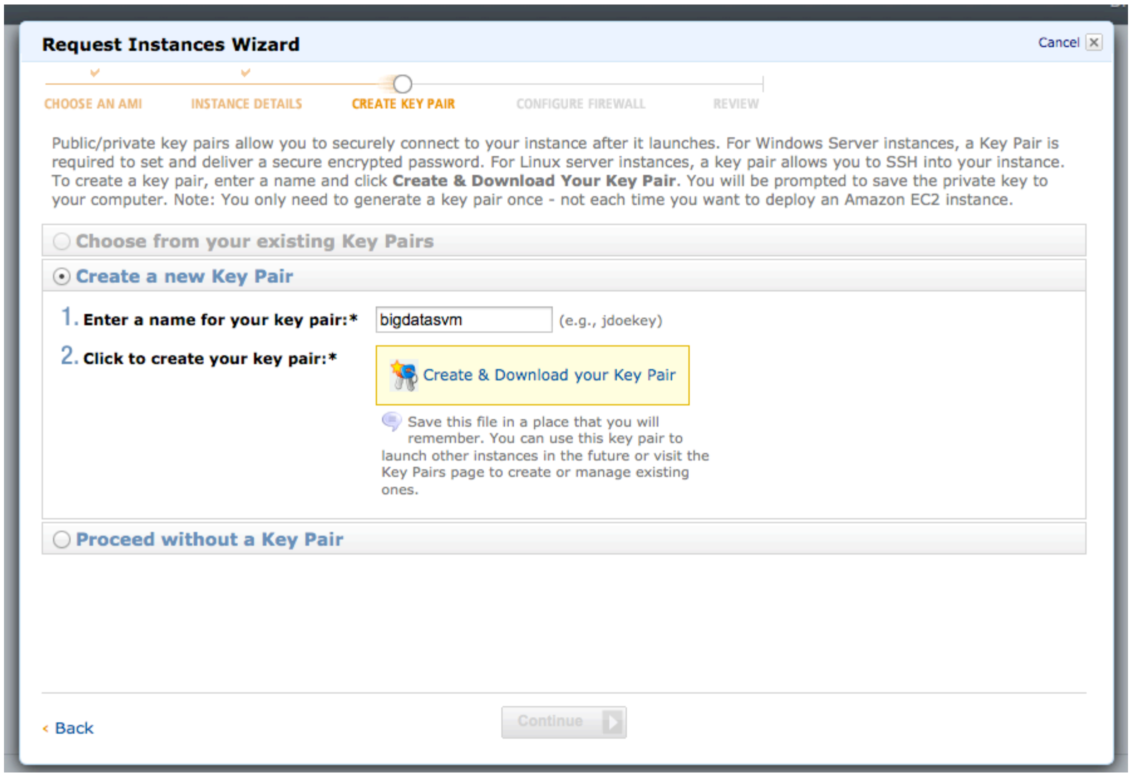


Add tags to your instance to simplify the administration of your EC2 infrastructure. A form of metadata, tags consist of a case-sensitive key/value pair, are stored in the cloud and are private to your account. You can create user-friendly names that help you organize, search, and browse your resources. For example, you could define a tag with key = Name and value = Webserver. You can add up to 10 unique keys to each instance along with an optional value for each key. For more information, go to Tagging Your Amazon EC2 Resources in the *EC2 User Guide*.

Key (127 characters maximum)	**Value** (255 characters maximum)	**Remove**
Name	Big_Data_SVM	✗
		✗

Add another Tag. (Maximum of 10)

Figure 11. Launch an instance step 6.



Request Instances Wizard

Cancel

CHOOSE AN AMI ✓ — INSTANCE DETAILS ✓ — CREATE KEY PAIR — CONFIGURE FIREWALL — REVIEW

Public/private key pairs allow you to securely connect to your instance after it launches. For Windows Server instances, a Key Pair is required to set and deliver a secure encrypted password. For Linux server instances, a key pair allows you to SSH into your instance. To create a key pair, enter a name and click **Create & Download Your Key Pair**. You will be prompted to save the private key to your computer. Note: You only need to generate a key pair once - not each time you want to deploy an Amazon EC2 instance.

○ Choose from your existing Key Pairs

● Create a new Key Pair

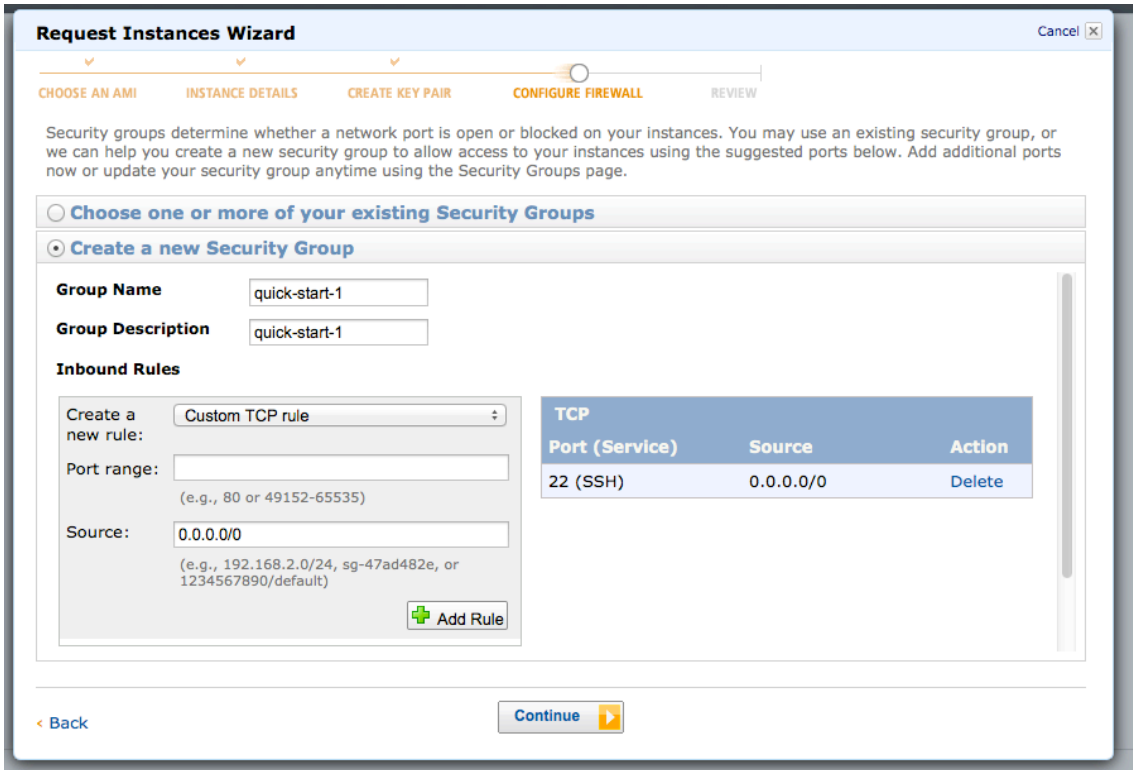
1. **Enter a name for your key pair:*** bigdatasvm (e.g., jdoekey)
2. **Click to create your key pair:*** Create & Download your Key Pair

Save this file in a place that you will remember. You can use this key pair to launch other instances in the future or visit the Key Pairs page to create or manage existing ones.

○ Proceed without a Key Pair

‹ Back   Continue

Figure 12. Launch an instance step 7.



Request Instances Wizard

Cancel

CHOOSE AN AMI ✓ — INSTANCE DETAILS ✓ — CREATE KEY PAIR ✓ — CONFIGURE FIREWALL — REVIEW

Security groups determine whether a network port is open or blocked on your instances. You may use an existing security group, or we can help you create a new security group to allow access to your instances using the suggested ports below. Add additional ports now or update your security group anytime using the Security Groups page.

○ Choose one or more of your existing Security Groups

● Create a new Security Group

Group Name quick-start-1

Group Description quick-start-1

Inbound Rules

Create a new rule: Custom TCP rule

Port range: (e.g., 80 or 49152-65535)

Source: 0.0.0.0/0 (e.g., 192.168.2.0/24, sg-47ad482e, or 1234567890/default)

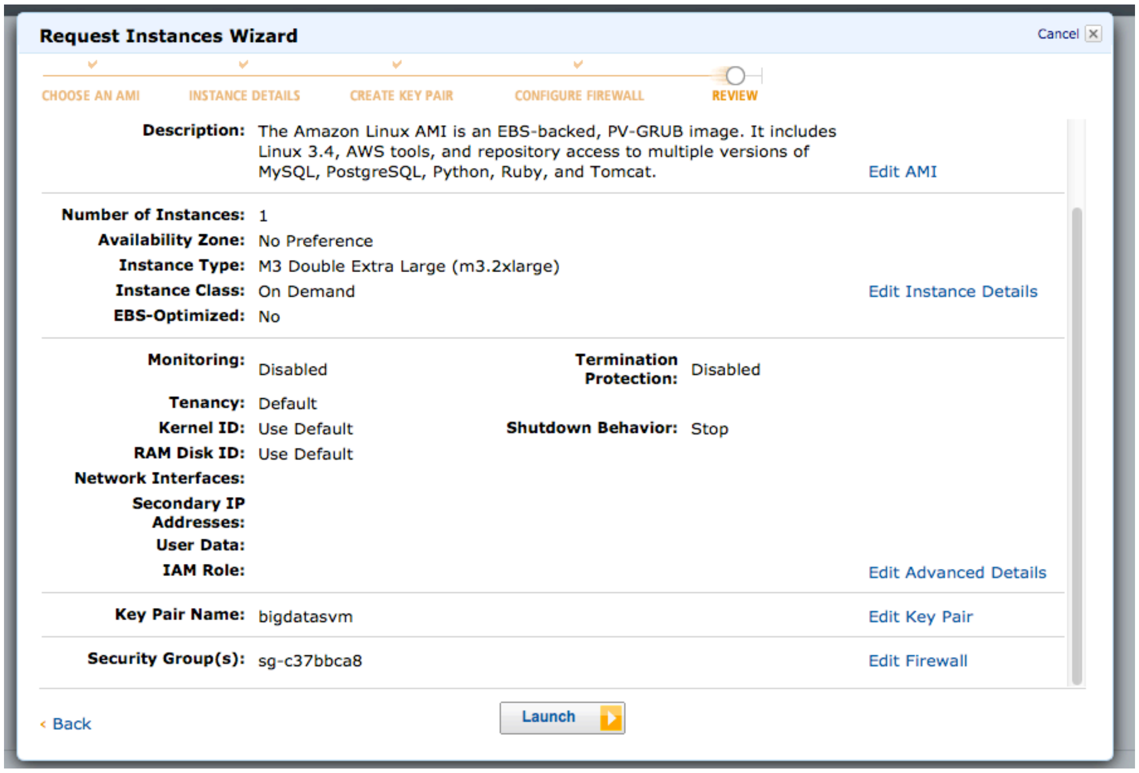
Add Rule

TCP

Port (Service)	Source	Action
22 (SSH)	0.0.0.0/0	Delete

‹ Back   Continue

Figure 13. Launch an instance step 8.



Description: The Amazon Linux AMI is an EBS-backed, PV-GRUB image. It includes Linux 3.4, AWS tools, and repository access to multiple versions of MySQL, PostgreSQL, Python, Ruby, and Tomcat. Edit AMI

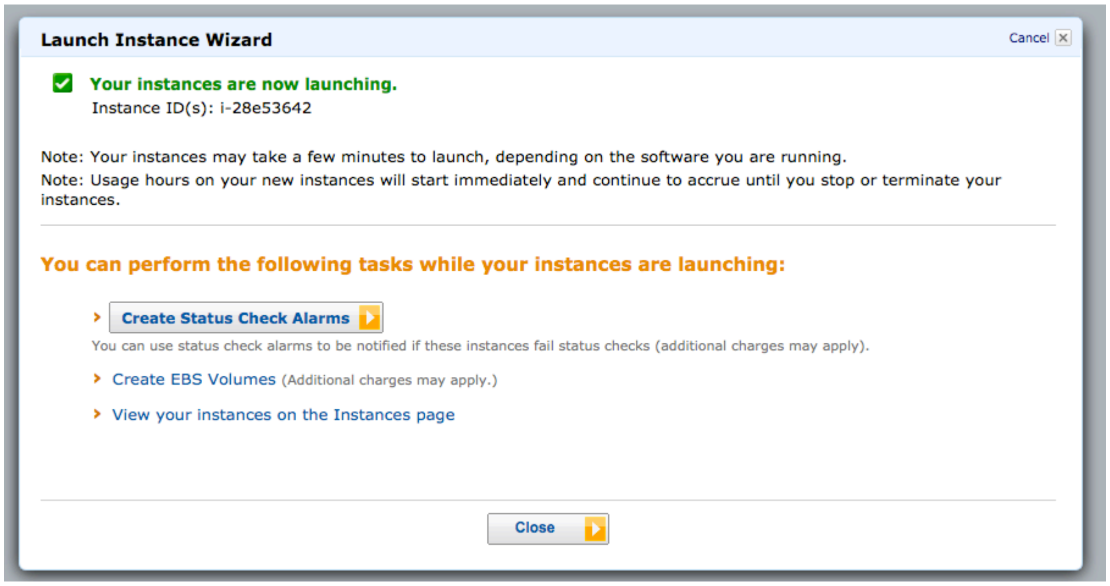
Number of Instances: 1
Availability Zone: No Preference
Instance Type: M3 Double Extra Large (m3.2xlarge)
Instance Class: On Demand Edit Instance Details
EBS-Optimized: No

Monitoring: Disabled **Termination Protection:** Disabled
Tenancy: Default
Kernel ID: Use Default **Shutdown Behavior:** Stop
RAM Disk ID: Use Default
Network Interfaces:
Secondary IP Addresses:
User Data:
IAM Role: Edit Advanced Details

Key Pair Name: bigdatasvm Edit Key Pair

Security Group(s): sg-c37bbca8 Edit Firewall

Figure 14. Launch an instance step 9.



Your instances are now launching.
Instance ID(s): i-28e53642

Note: Your instances may take a few minutes to launch, depending on the software you are running.
Note: Usage hours on your new instances will start immediately and continue to accrue until you stop or terminate your instances.

You can perform the following tasks while your instances are launching:

› Create Status Check Alarms
You can use status check alarms to be notified if these instances fail status checks (additional charges may apply).

› Create EBS Volumes (Additional charges may apply.)

› View your instances on the Instances page

Figure 15. Launch an instance step 10.

We obtain the IP address of this instance from the EC2 dashboard, ec2-50-19-28-27.compute1.amazonaws.com, we are now going to use this to access the instance. On my Mac, I use the ‘Terminal’ program, and enter the following command to access my instance,

```
> ssh -i bigdatasvm.pem ec2-user@ec2-50-19-28-27.compute1.amazonaws.com
```

But, before this instance can be successfully accessed, the file permissions on the key-pair need to be changed. We do this by using the command,

```
> chmod 400 bigdatasvm.pem
```

Now we can access the instance by using the ssh command as shown earlier.

To ensure our instance uses the entire 100GB from it’s EBS (Elastic Block Storage) volume that was set up we use this command,

```
> sudo resize2fs /dev/xvda1
```

3. Support Vector Machines (SVM)

3.1 Description of How SVM works

As discussed in the earlier section on SVM, I had mentioned that SVM establishes the optimal margin hyperplane for the data, this is illustrated in the figure below.

As we can see, the black separating hyperplane has maximum margin from the closest points to it, these are known as the support vectors.

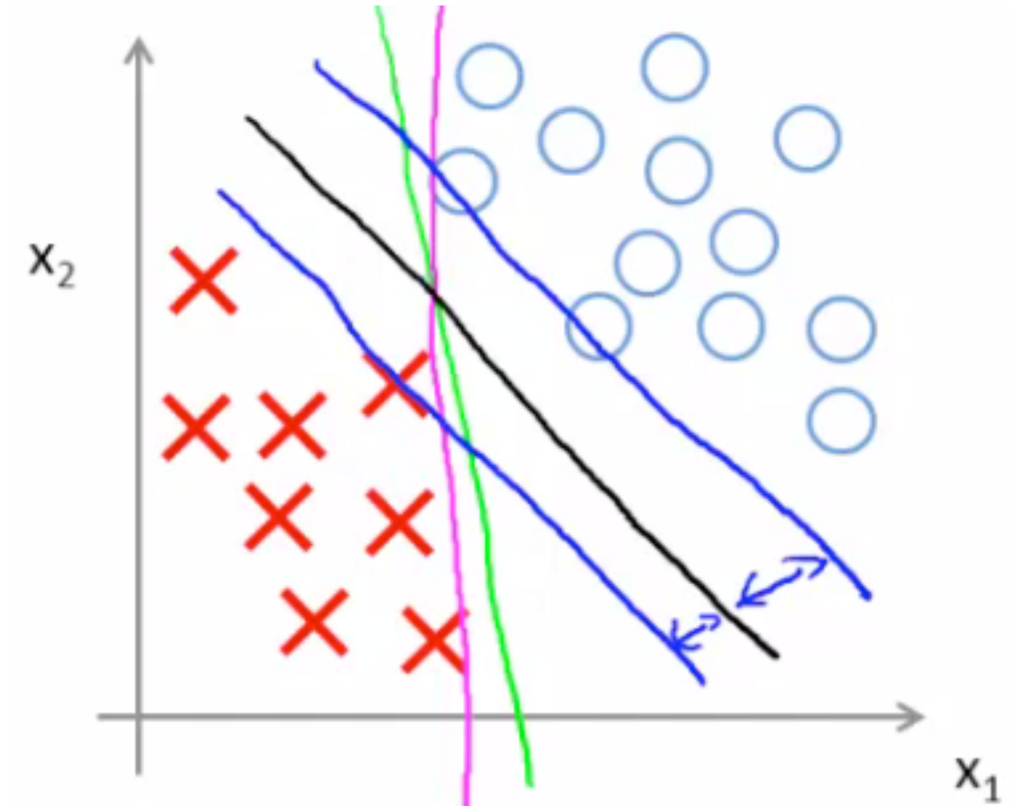


Figure 16. Optimal margin classifier is the black separating hyperplane. [7]

I will now illustrate mathematically how SVM performs the optimization to arrive on the hyperplane which classifies the data.

$$\min_{\theta} C \sum_{i=1}^{m} \left[y^{(i)} cost_1(\theta^T x^{(i)}) + (1 - y^{(i)}) cost_0(\theta^T x^{(i)}) \right] + \frac{1}{2} \sum_{i=1}^{n} \theta_j^2$$

This is the SVM cost function (for the linear case), [7]

where,

'C' is the cost parameter,
θ is the weight parameter,
'm' is the number of training examples,
y(i) is the classification label for the ith example,
x(i) is the ith attribute,
'n' is the number of attributes,
$cost_1(\theta^T x)$ is ($-\log h_\theta(x(i))$) and
$cost_0(\theta^T x)$ is ($-\log (1 - h_\theta(x(i))$)

A prediction of '1' or '0' is made by the hypothesis,

$h_\theta(x) = 0$, if $\theta^T x < 0$ and
$h_\theta(x) = 1$, if if $\theta^T x >= 0$

For the non-linear case, the optimization objective is as follows, [7]

$$\min_{\theta} C \sum_{i=1}^{m} y^{(i)} cost_1(\theta^T f^{(i)}) + (1 - y^{(i)}) cost_0(\theta^T f^{(i)}) + \frac{1}{2} \sum_{j=1}^{n} \theta_j^2$$

where,

'f(i)' is the i'th feature, where features are the high order polynomials from the attributes based on the kernel function used. The rest of the variables are the same as before. This leads to a hyperplane as shown below,

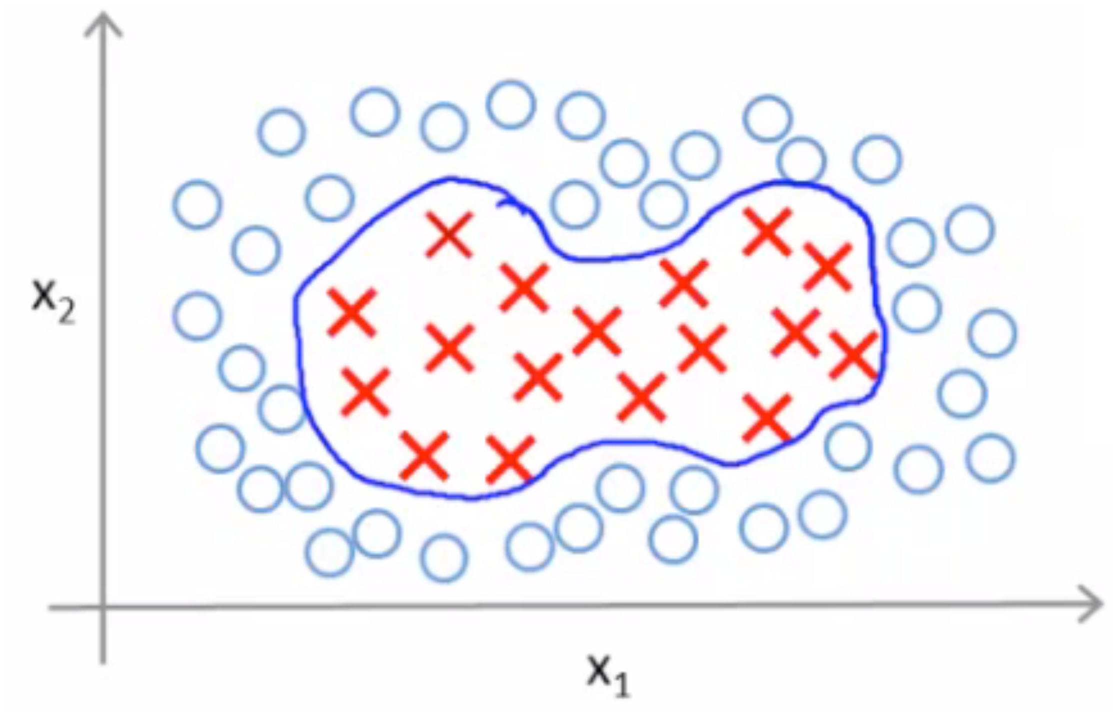


Figure 17. Non-linear separating hyperplane [7]

The various kernel options available in the ‘libsvm’ tool (described later) are as follows,

Linear Kernel [8], $k(x, y) = x^T y + c$

Polynomial Kernel[8], $k(x, y) = (\alpha x^T y + c)^d$

Radial Basis Function Kernel[8], $k(x, y) = \exp\left(-\gamma ||x - y||^2\right)$

Sigmoid Kernel[8], $k(x, y) = \tanh(\alpha x^T y + c)$

These kernel functions are basically similarity functions for the features which are mapped from the attributes.

3.2 Using MATLAB SVM Toolbox

In phase 1 of the project, I was using the student version of MATLAB 2012a on my MacBook Pro, with the “Bioinformatics” toolbox which includes the svm functions that we need to use.

To get the data into the workspace, we use the “Import Data” tool from the File menu, and select our CSV file with the labeled data set. We select the options to format the data viz. the column separator is set to “Comma” and then the data is imported. This creates a matrix, which is used for svm operations.

This is the MATLAB code I wrote to generate my svm training model,

```
> data = labeledmatrix(1:end, 2:342);
> group = labeledmatrix(1:end,343:343);
> svmstruct = svmtrain(data,group);
```

To classify data using the generated ‘svmstruct’ model, I wrote this code,

```
> class = svmclassify(svmstruct,unlabeledmatrix);
```

This gives us a ‘class’ variable which contains the required predictions of classification of the unlabeled data.

As I mentioned earlier, this was a very slow process due to the limitations of my laptop. On my machine, I was never able to successfully train on the entire data set. The memory restrictions, and lessons learned from this phase were used to be applied in the phase 2 of the project, when using AWS.

3.3 Tutorial on LIBSVM

The open source tool that I used for my svm training and classification is called libsvm. Available from here, http://www.csie.ntu.edu.tw/~cjlin/libsvm/ it is one of the more popular open source tools available for using svm. The current version is 3.17, which is the one that I used.

My AWS instance was a linux based one, and these are the steps I used to install and use the libsvm tool on that system.

First, we transfer the libsvm files over to the instance using this command (after downloading to your local system),

```
> scp -i bigdatasvm.pem libsvm-3.17.zip ec2-
user@ec2-50-19-28-27.compute1.amazonaws.com:/home/ec2-user/
```

Then we unzip the files on the instance using,

```
> unzip libsvm-3.17.zip
```

Then we need to build the libsvm files,

```
> cd libsvm-3.17
> make
```

This gives us the required executables, svm-train and svm-predict.

Reading the README file within this folder gives a good description of how to use the software. The key part is the data format, which we will come to next.

3.4 Converting the Data to Libsvm Format

The data set given to us by IBM was in the format as described earlier, where the first column was a "question id", the next 342 columns were features (real numbers), and the final column was a label ("true" or "false"). And there were approximately 1.3 million rows of data.

The libsvm tool requires the data to be in a format as such [9],

```
<label> <index1>:<value1> <index2>:<value2> ...
```

where, "Each line contains an instance and is ended by a '\n' character. For classification, <label> is an integer indicating the class label. The pair <index>:<value> gives a feature (attribute) value: <index> is an integer starting from 1 and <value> is a real number."

To achieve this conversion I wrote a Perl script, the code for which is below.

```
#! /usr/bin/perl
use warnings;
$my_dir = "/Users/Dhruv/Documents/MATLAB/data/labelled_2000_lines/";
opendir MYDIR, $my_dir;
readdir MYDIR;
readdir MYDIR;
readdir MYDIR;
while ( $filename = readdir MYDIR ) {
        open SVMFILE, "<", "$my_dir$filename" or die "$!";
        open OUTFILE, ">>", "out" or die "$!";
        while (<SVMFILE>) {
                chomp($line = $_);
                @splitline = split(',',$line);
                $qid = shift(@splitline);
                $label = pop(@splitline);
                if ($label eq "false") {
                    $newlabel = '-1';
                } elsif ($label eq "true") {
                    $newlabel = '+1';
                } else {
                    $newlabel = '0';
                }
                $i = 1;
                foreach $value (@splitline) {
                    push(@newline,"$i:$value ");
                    $i += 1;
                }
                unshift(@newline,"$newlabel ");
                print OUTFILE @newline,"\n";
                undef @splitline;
                undef @newline;
        }
        close(SVMFILE);
        close(OUTFILE);
}
closedir MYDIR;
exit;
```

This code basically, opens the labeled data set file, reads it line-by-line, checks what the label assigned is, converts it to +1 for true, -1 for false and 0 otherwise, and then rearranges the data with the required feature index values, and then writes it out to a new file, again line-by-line. This successfully converts the IBM supplied data to the required format of libsvm.

4. Execution and Analysis

4.1 Training and Classification of Various Data Sets

For running my training and classification of data, I used 6 subsets of the entire data set and the entire data set as a whole, making seven data sets in total. For this, I divided the original data set into subsets of size 2,000 rows, 10,000 rows, 50,000 rows, 100,000 rows, 200,000 rows, 500,000 rows and then the entire data set of almost 1.3 million rows. I used the linux 'split' utility to obtain these subsets. This was executed as such,

```
> split -l <number_of_rows> <labeled_data>
```

As per the documentation of libsvm [9], training is achieved by using the 'svm-train' command as such,

```
Usage: svm-train [options] training_set_file [model_file]
options:
-s svm_type : set type of SVM (default 0)
-t kernel_type : set type of kernel function (default 2)
        0 -- linear: u'*v
        1 -- polynomial: (gamma*u'*v + coef0)^degree
        2 -- radial basis function: exp(-gamma*|u-v|^2)
        3 -- sigmoid: tanh(gamma*u'*v + coef0)
        4 -- precomputed kernel (kernel values in training_set_file)
-d degree : set degree in kernel function (default 3)
-g gamma : set gamma in kernel function (default 1/num_features)
-r coef0 : set coef0 in kernel function (default 0)
-c cost : set the parameter C of C-SVC, epsilon-SVR, and nu-SVR (default 1)
-n nu : set the parameter nu of nu-SVC, one-class SVM, and nu-SVR (default 0.5)
-p epsilon : set the epsilon in loss function of epsilon-SVR (default 0.1)
```

-m cachesize : set cache memory size in MB (default 100)
-e epsilon : set tolerance of termination criterion (default 0.001)
-h shrinking : whether to use the shrinking heuristics, 0 or 1 (default 1)
-b probability_estimates : whether to train a SVC or SVR model for probability estimates, 0 or 1 (default 0)
-wi weight : set the parameter C of class i to weight*C, for C-SVC (default 1)
-v n: n-fold cross validation mode
-q : quiet mode (no outputs)

and, classification with the 'svm-predict' function as such [9],

Usage: svm-predict [options] test_file model_file output_file

4.2 Analysis of Outputs

I ran training runs with the 4 different kernel options available, and each had different training time, classification time, and accuracy rate for the various data set sizes.

For the six subsets, I trained and classified on the same size data set, but the trained data set was seen and the classified one was an unseen one by the system. For training of the entire data set, I run classification on a sub set of 500,000 rows of data.

I present here a table comparing my various outputs from the training and classification experiments.

Kernel Type	Data Set Size (rows)	Training Time (hh:mm:ss)	Classification Time (hh:mm:ss)	Classification Accuracy (%)
RBF	2,000	00:00:01.49	00:00:01.33	99.55
RBF	10,000	00:01:04.36	00:00:33.87	99.67
RBF	50,000	00:37:24.09	00:13:35.24	99.66
RBF	100,000	02:10:58	00:36:36.72	99.626
RBF	200,000	06:20:34	01:59:12	99.353
RBF	500,000	57:13:19	18:20:01	99.5746
RBF	1,314,407	100:00:00+	-	-
Linear	2,000	00:00:01.47	00:00:00.14	98.85
Linear	10,000	00:01:06.52	00:00:00.99	97.35
Linear	50,000	00:02:55.598	00:00:11.002	98.534
Linear	100,000	00:16:31.525	00:00:35.764	96.467
Linear	200,000	01:29:05.974	00:02:05.891	97.651
Linear	500,000	19:09:17	00:32:19.72	99.9898
Linear	1,314,407	91:50:21	00:48:34.30	98.7198
Sigmoid	2,000	00:00:00.18	00:00:00.12	99.55
Sigmoid	10,000	00:00:01.58	00:00:00.94	99.53
Sigmoid	50,000	00:00:21.767	00:00:13.682	99.412
Sigmoid	100,000	00:01:16.990	00:00:45.777	99.322
Sigmoid	200,000	00:07:07.475	00:02:51.696	98.92
Sigmoid	500,000	03:14:45.664	00:29:15.549	99.1438
Sigmoid	1,314,407	28:52:27	01:08:02	99.0208
Polynomial	2,000	00:00:00.95	00:00:00.14	98.85
Polynomial	10,000	00:00:25.01	00:00:01.31	42.99

Kernel Type	Data Set Size (rows)	Training Time (hh:mm:ss)	Classification Time (hh:mm:ss)	Classification Accuracy (%)
Polynomial	50,000	00:03:01.552	00:00:11.21	33.356
Polynomial	100,000	00:14:28.676	00:00:36.698	10.389
Polynomial	200,000	03:28:27.041	00:02:08.221	98.776
Polynomial	500,000	14:46:31	00:19:48.68	21.4872
Polynomial	1,314,407	19:55:52	01:14:58	43.1378

Table 2. Comparison of Training and Classification outputs from libsvm.

*** RBF: Radial Basis Function**

Note:

1) The classification accuracy is calculated by the libsvm tool, and forms part of the output.

2) The time taken to train and classify was found by using the linux ‘time’ command when executing the train and classify commands, this is the real elapsed time which is the output of ‘time’ command.

3) The kernels are varied by using the kernel option in libsvm as described in the ‘svm-train’ usage earlier.

4) RBF training takes more than 100 hours, which is the maximum credit I can use for a single AWS instance, and hence we leave this part as unresolved.

I now present the graphical analysis for these outputs here.

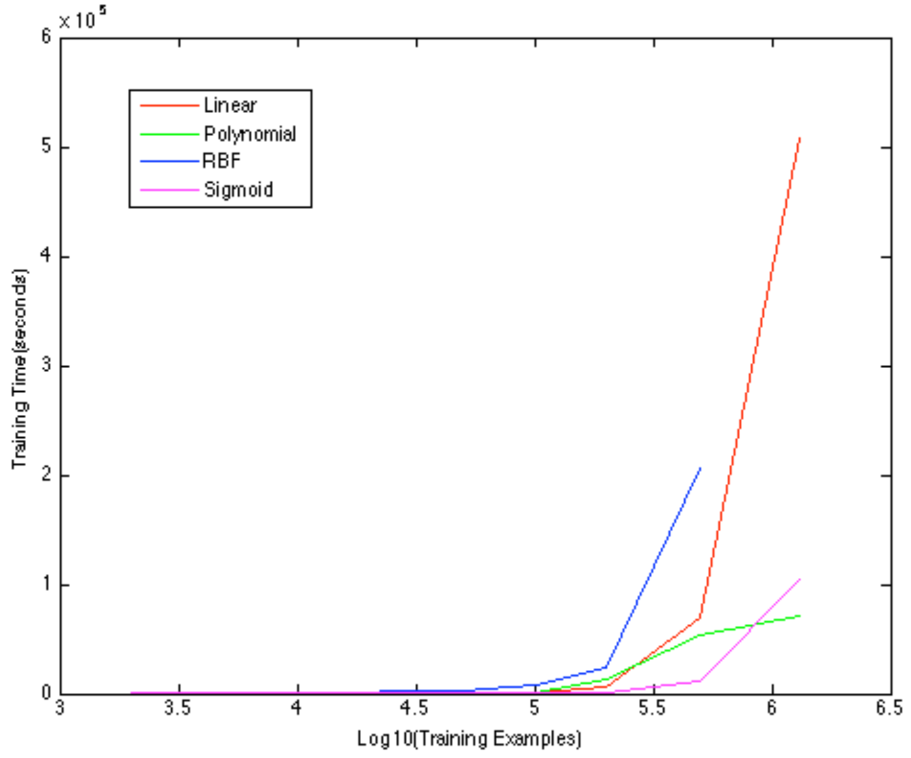


Figure 18. Graph of Training Examples vs Training Time for Various Kernels

As we can see above RBF Kernel took the most time to train (even though it has only 6 data points) since the kernel function it uses is an exponential one, and the curve increases exponentially.

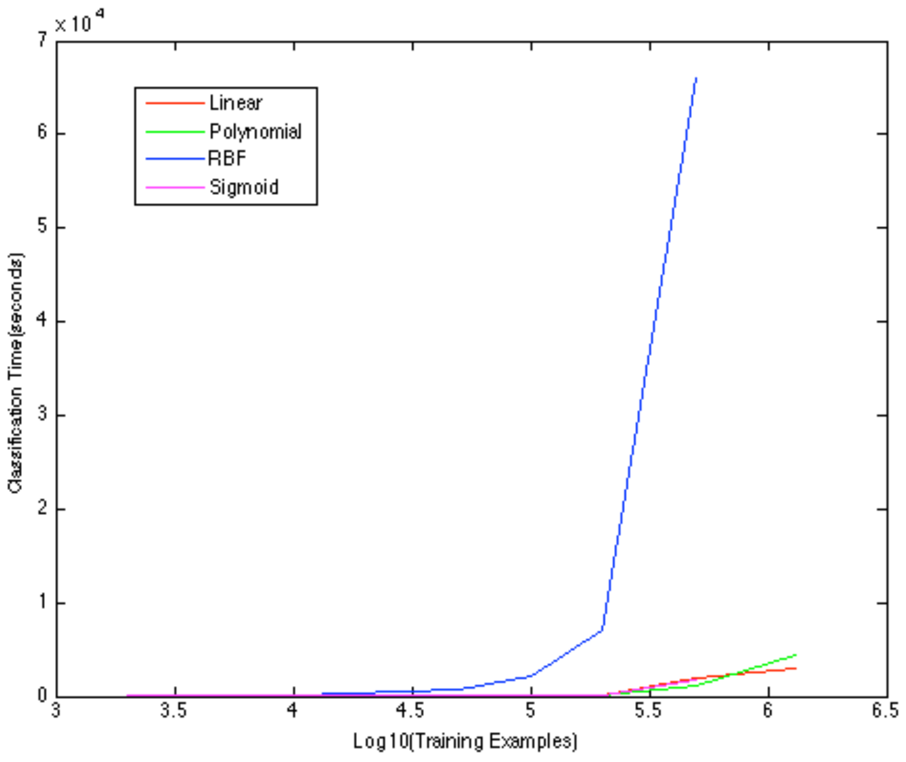


Figure 19. Graph of Training Examples vs Classification Time for Various Kernels

As we can see here again classification time for RBF kernel is most and increases exponentially again.

For the two graphs below, we can see that RBF kernel gives best accuracy of classifications and polynomial is most erratic. Also, fluctuations in linear kernel output says that data is not linearly separable.

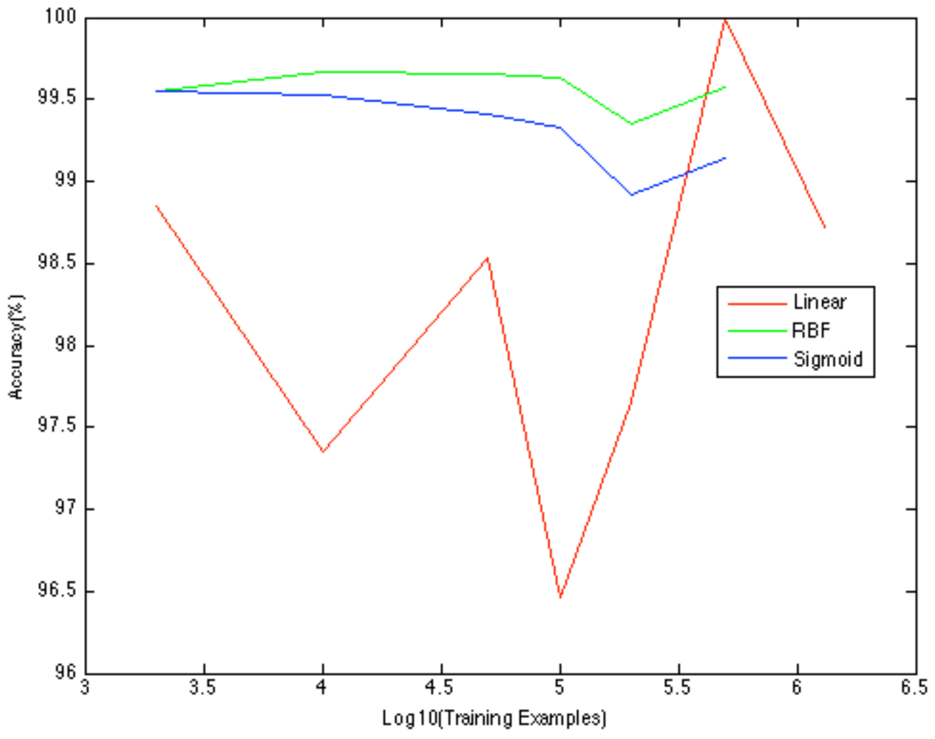


Figure 20. Graph of Training Examples vs Accuracy Rates for Various Kernels

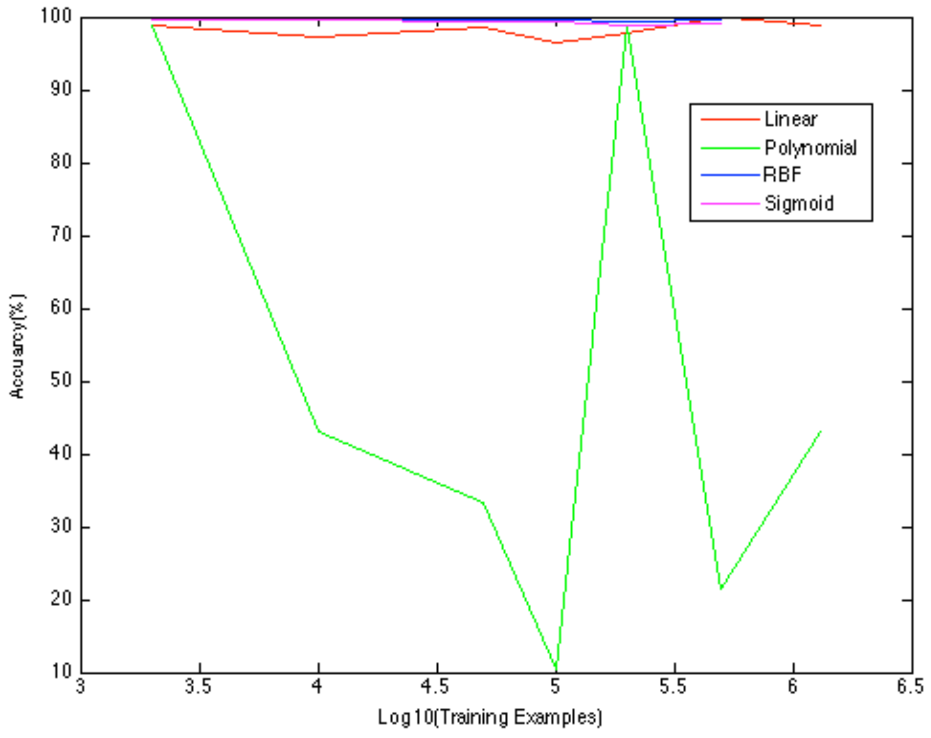


Figure 21. Graph of Training Examples vs Accuracy Rates for Various Kernels

Kernel Type	Training Cost (approx.)	Classification Cost (approx.)	Total Cost
RBF	$172	$25	$197
Linear	$118	$8	$126
Sigmoid	$38	$8	$46
Polynomial	$43	$8	$51

Table 3. Cost spent on AWS hours for all experiments of various kernels

As can be seen from the above table RBF cost the most in terms of dollars spent on AWS, whereas Sigmoid and Polynomial were the least. We see the graph of comparisons below.

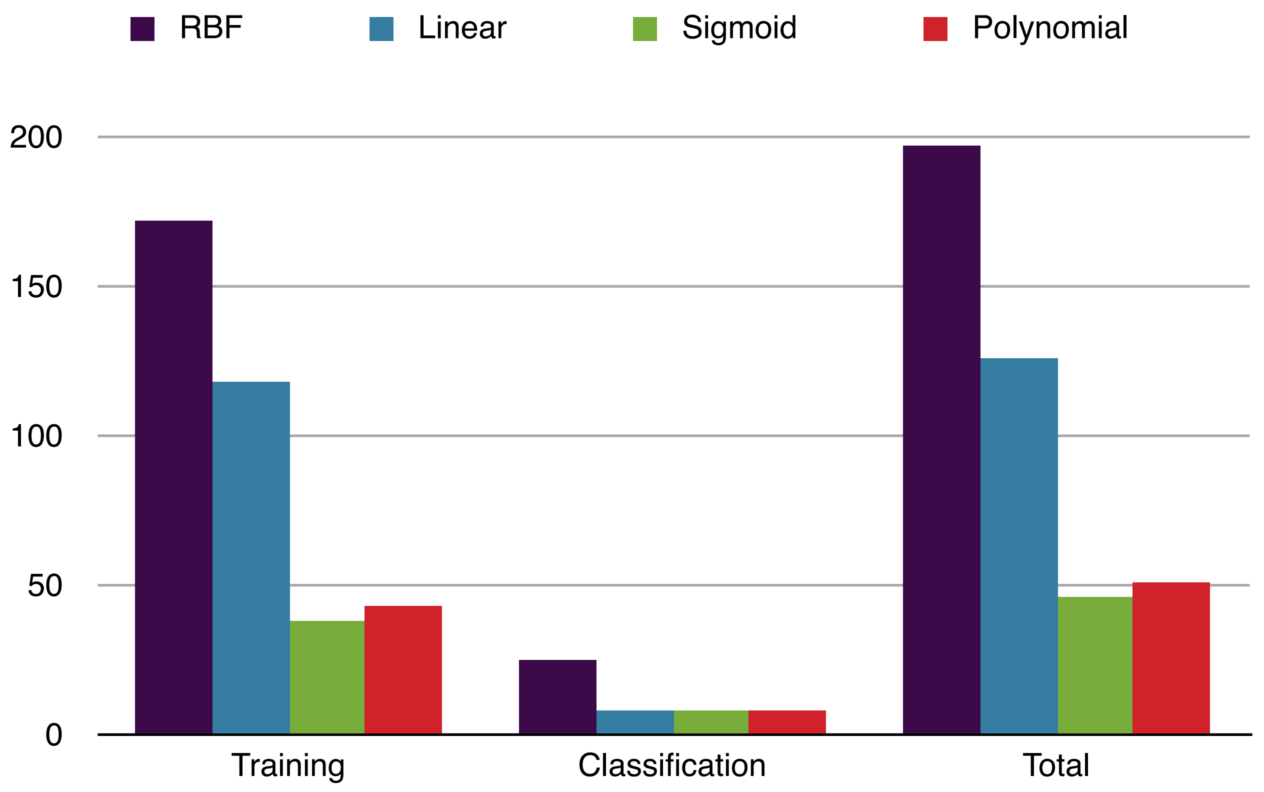


Figure 22. Comparison of AWS costs spent on experiments for kernels

5. Conclusion

As has been seen from this project, the major challenge in such a task was dealing with the large data set size. Once the hurdle of machine power was crossed using AWS, data preparation was crucial for the tool being used. The libsvm tool, and the SVM technique in general, has proven to be quite effective for this task of classification. Using it's powerful method of kernels, it can deal with complicated data that cannot be linearly separated. Though these are computationally expensive for large data sets, AWS offers an affordable solution to the average user.

RBF kernel prove to be most expensive in cost and took most time to train and classify but gave best classification results. Whereas Sigmoid kernel prove to be most cost-effective by giving consistent results and not taking long to train or classify and being cheap in AWS costs.

6. References

[1] IBM 'The Great Minds Challenge' introduction documentation for technical pilot at SJSU.

[2] https://en.wikipedia.org/wiki/Amazon_Web_Services May 8, 2013

[3] http://www.ec2instances.info May, 8, 2013

[4] http://en.wikipedia.org/wiki/Support_vector_machine May 9, 2013

[5] http://cs229.stanford.edu/notes/cs229-notes3.pdf May 9, 2013 (Andrew Ng's course notes for CS229 at Stanford University)

[6] http://stackoverflow.com/questions/11253746/amazon-ec2-compute-unit-and-gceu-google-compute-engine-unit#11254338 May 9, 2013

[7] http://www.holehouse.org/mlclass/12_Support_Vector_Machines.html May 9, 2013

[8] http://www.crsouza.blogspot.com/2010/03/kernel-functions-for-machine-learning.html May 10, 2013

[9] http://www.csie.ntu.edu.tw/~cjlin/libsvm/ May 10, 2013

[10] http://www.csie.ntu.edu.tw/~cjlin/papers/guide/guide.pdf May 10, 2013

[11] https://class.coursera.org/machlearning-001/lecture/index May 10, 2013

[12] https://class.coursera.org/ml-003/lecture/index May 10, 2013