CSYE 7374: Big-Data Systems and Intelligent Analytics

Midterm Project Executive Summary

Team 4

- Samir Sharan
- Harshit Shah
- Jeevan Reddy

Submitted on: 07/11/2014

Case-1 – Regression and Classification

Predict the year of the song based on multiple features. There are 90 features, 12 = timbre average, 78 = timbre covariance. The first value is the year (target), ranging from 1922 to 2011.

Source: https://archive.ics.uci.edu/ml/datasets/YearPredictionMSD#

Approach: We used 2 different approaches for solving the problem. The data set has a column which has the year the song was released in, along with many other features. We used regression techniques on this dataset to predict in which year was a particular song released. Another approach is to use classification techniques to determine whether the song was released before 1965 or after.

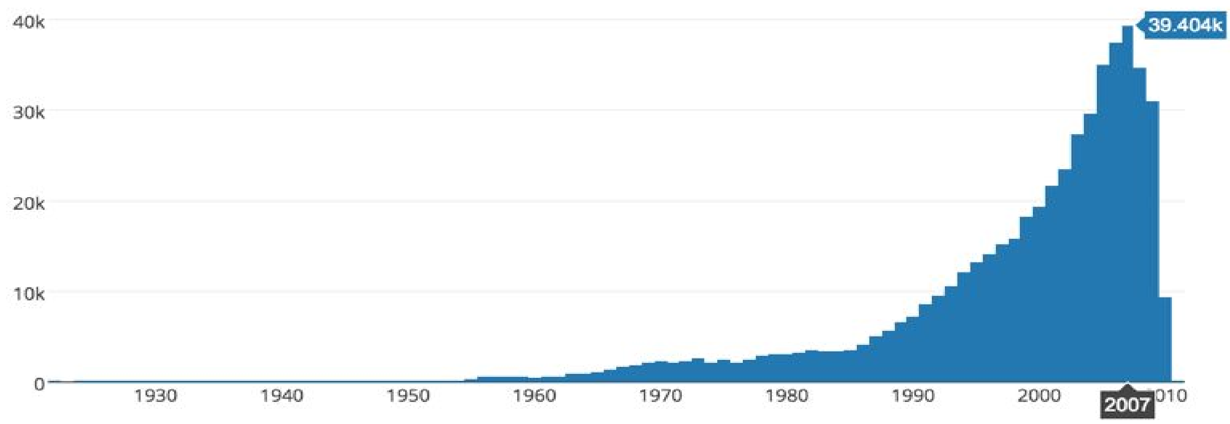
Data Exploration and Data Summarization: We used Pandas library in python to explore data. We took a sample of 100 data points and found summary statistics for the sample to get a central tendency of the data set

```
#Statistics
df_sample = df.sample(100)
print df.describe()
```

	year	c1	c2	c3	c4	c5 \
count	100.00000	100.000000	100.000000	100.000000	100.000000	100.000000
mean	1998.75000	42.893869	0.314089	11.892564	-0.241262	-2.296983
std	10.80252	7.240973	45.199512	39.623318	15.444873	24.879741
min	1964.00000	19.004600	-139.922490	-116.175080	-48.249140	-61.863550
25%	1994.75000	38.939297	-32.439545	-9.763915	-9.882523	-17.764032
50%	2002.00000	44.586750	8.461805	8.279410	-1.248245	-3.102155
75%	2006.00000	47.969360	31.131400	35.819243	6.522717	14.179785
max	2010.00000	53.445940	119.398240	152.358000	48.850960	86.584270

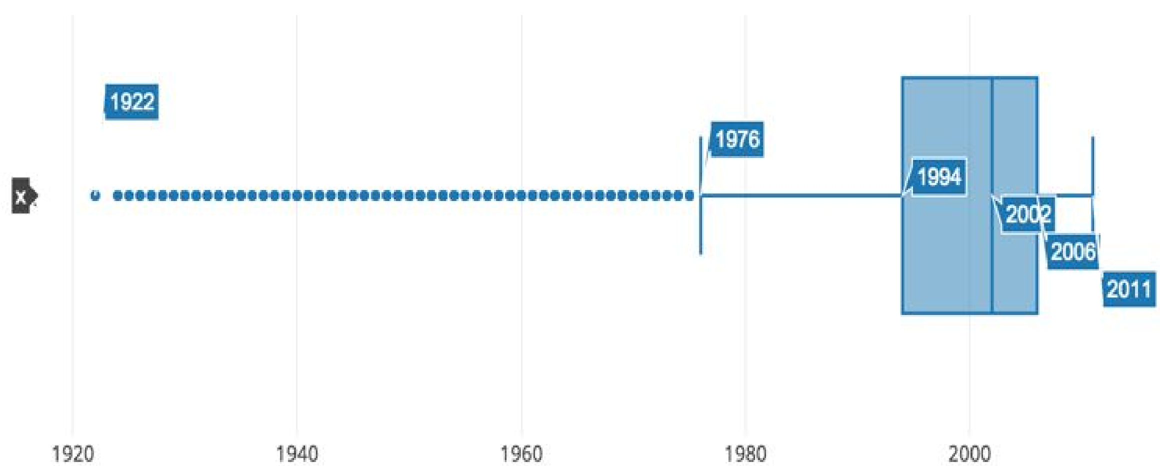
Frequency Distribution:

https://plot.ly/~harshitshah/5/year/



Identifying Outliers:

https://plot.ly/~harshitshah/45/



Here, records before 1976 are identified as outliers. But since we need to consider records before 1965 as well, we can't discard them in the classification problem.

Feature Engineering: The data here is unscaled, i.e data in each column is set on its own scale. This leads to incorrect predictions and classifications. To solve this problem, we calculate the z-score of each column on the scale of 0 to 1. z-score scales each column on the scale of 0 to 1 and hence data in each column has equal degrees.

Before Normalization:

```
print df_train.head(3)
print df_test.head(3)
```

```
[8 rows x 91 columns]
         c1        c2        c3        c4        c5        c6        c7  \
0  48.73215  18.42930  70.32679  12.94636 -10.32437 -24.83777   8.76630
1  50.95714  31.85602  55.81851  13.41693  -6.57898 -18.54940  -3.27872
2  48.24750  -1.89837  36.29772   2.58776   0.97170 -26.21683   5.05097
```

After Normalization:

```
#normalize (z-score)
df_norm = (df - df.mean())/df.std()
```

```
print df_train.head(3)
print df_test.head(3)
```

```
         c1        c2        c3        c4        c5        c6        c7  \
0  0.880921  0.332293  1.748545  0.721828 -0.164946 -1.191172  0.765678
1  1.247623  0.592599  1.337179  0.750657 -0.001111 -0.702100 -0.060917
2  0.801045 -0.061804  0.783688  0.087218  0.329178 -1.298427  0.510712
```

Data Split: We are given that the training data should be first 463,715 records and test data should be remaining 51,630 records. To achieve this, we created two separate dataframes from the original one representing train and test data.

```
#split the data into train and test
df_train = df_norm[:463715]
df_test = df_norm[463716:515345]
```

Model Classify: We applied 2 classification algorithms, SVMWithSGD and LogisticRegressionWithLBFGS. We create model by feeding it the training data and then apply the model on the test data. We train the model for 100 iterations.

```
#run SVMWithSGD
model = SVMWithSGD.train(parsedData_train, iterations=100)
```

```
#run LogisticRegressionWithLBFGS
model_Log = LogisticRegressionWithLBFGS.train(parsedData_train, iterations=100)
```

Model Evaluation: We calculated the classification error for both the algorithms to evaluate which model fits better on this data set.

```
Training Error - SVMWithSGD = 0.381551168282
Testing Error - SVMWithSGD = 0.382834895793
Training Error - Logistic Regression = 0.40553141477
Testing Error - Logistic Regression = 0.408015030604
```

Based on the model evaluation, we think that **SVMWithSGD** is a better fit for this problem.

Model Regression: We applied 2 regression algorithms, LinearRegressionWithSGD and RidgeRegressionWithSGD, to predict the year of the song. We built the model over 100 iterations.

```
#run LinearRegression
model = LinearRegressionWithSGD.train(parsedData_train)
```

```
#run RidgeRegression
model = RidgeRegressionWithSGD.train(parsedData_train)
```

Model Evaluation: For model evaluation, we calculated the Mean Square Error for both the algorithms to evaluate which model fits better on this data set.

```
Training Mean Squared Error for Linear Regression = 0.76399855085
Testing Mean Squared Error for Linear Regression = 0.757315556322
Training Mean Squared Error for Ridge Regression = 0.76431105639
Testing Mean Squared Error for Ridge Regression = 0.757672568814
```

Based on the Mean Squared Errors, we think that **LinearRegressionWithSGD** is a better fit for this problem.

Case-2 – Determine Income based on characteristics

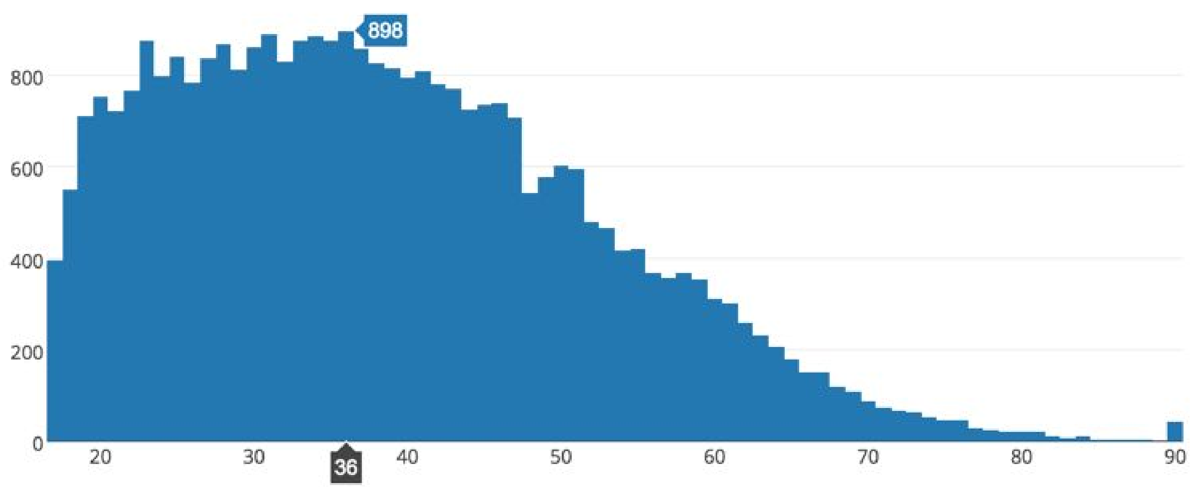
Determining whether a person has income >50K or <=50K from the given dataset with categorical and numerical variables.

Source: https://archive.ics.uci.edu/ml/datasets/Census+Income

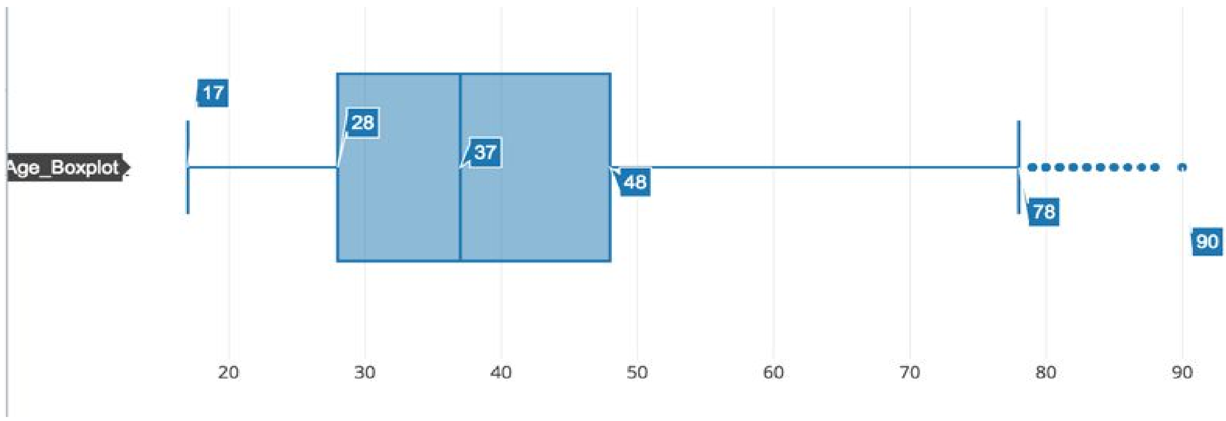
Approach: We have used Logistic regression and Linear Support Vector Machines(SVMs) classification models to determine the income based on the other given features and compared the results. Due to multiple columns with categorical data, pre-processing altogether more important to perform on this data set.

Data Exploration and Data Summarization: We used Pandas library in python to explore data. We also used Plotly to visualize and explore data. We visualized histograms of age, education and race to determine which category occupies most of our data set.

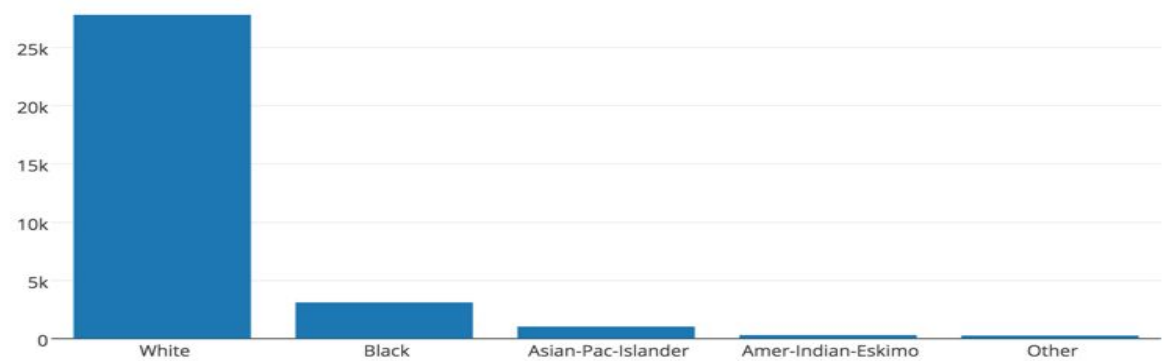
https://plot.ly/~harshitshah/5/age/



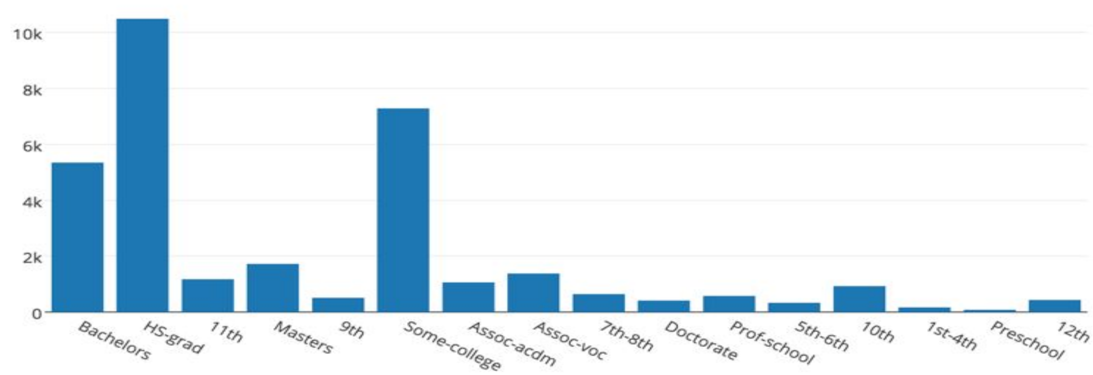
https://plot.ly/~harshitshah/45/age-boxplot/



https://plot.ly/~harshitshah/5/race/

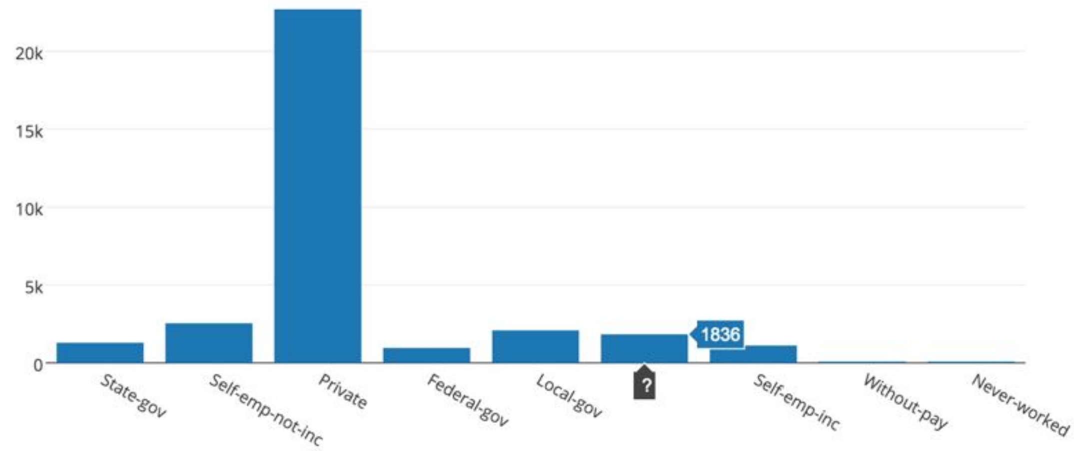


https://plot.ly/~harshitshah/5/education/



Finding Missing Values

https://plot.ly/~harshitshah/5/workclass/



We replaced the missing values with the mode.

```
#replacing missing values with mode
df.workclass = df.workclass.replace([' ?'],[mode_workclass])
df.maritalStatus = df.maritalStatus.replace([' ?'],[mode_maritalStatus])
df.occupation = df.occupation.replace([' ?'],[mode_occupation])
df.relationship = df.relationship.replace([' ?'],[mode_relationship])
df.race = df.race.replace([' ?'],[mode_race])
df.sex = df.sex.replace([' ?'],[mode_sex])
df.nativeCountry = df.nativeCountry.replace([' ?'],[mode_nativeCountry])
```

Feature Engineering: The data here is unscaled, i.e data in each column is set on its own scale. This leads to incorrect classifications. To solve this problem, we calculate the z-score of each column on the scale of 0 to 1. z-score scales each column on the scale of 0 to 1 and hence data in each column has equal degrees.

Since some of the columns represent categorical data, we convert them into dummy variables

```
#create dummy variables
df_dummy = pd.get_dummies(df)
df_dummy.drop(df_dummy.columns[[2,6,14,30,37,51,57,62,64,105]], axis=1, inplace=True)

#create dummy variables: test
df_test_dummy = pd.get_dummies(df_test)
df_test_dummy.drop(df_test_dummy.columns[[2,6,14,30,37,51,57,62,64,105]], axis=1, inplace=True)
```

Normalization:

```
#normalization
df_norm = (df_dummy - df_dummy.mean()) / (df_dummy.std())

#normalization: test
df_test_norm = (df_test_dummy - df_test_dummy.mean()) / (df_test_dummy.std())
```

Model Classify: We applied Logistic regression and Linear Support Vector Machines(SVMs) classification models. We create model by feeding it the training data and then apply the model on the test data.

```
#run LogisticRegressionWithLBFGS L1
model1 = LogisticRegressionWithLBFGS.train(parsedData, regType='l1')

#run LogisticRegressionWithLBFGS L2
model2 = LogisticRegressionWithLBFGS.train(parsedData, regType='l2')

labelsAndPredsL1 = parsedData.map(lambda p: (p.label, model1.predict(p.features)))
trainErrL1 = labelsAndPreds.filter(lambda (v, p): v != p).count() / float(parsedData.count())
testErrL1 = labelsAndPreds.filter(lambda (v, p): v != p).count() / float(parsedTestData.count())
print("Training Error L1 = " + str(trainErrL1))
print("Test Error L1 = " + str(testErrL1))

labelsAndPredsL2 = parsedData.map(lambda p: (p.label, model2.predict(p.features)))
trainErrL2 = labelsAndPreds.filter(lambda (v, p): v != p).count() / float(parsedData.count())
testErrL2 = labelsAndPreds.filter(lambda (v, p): v != p).count() / float(parsedTestData.count())
print("Training Error L2 = " + str(trainErrL2))
print("Test Error L2 = " + str(testErrL2))
```

```
#run SVMWithSGD L1
model1 = SVMWithSGD.train(parsedData, regType='l1', step=0.0001)

#run SVMWithSGD L2
model2 = SVMWithSGD.train(parsedData, regType='l2')

labelsAndPredsL1 = parsedData.map(lambda p: (p.label, model1.predict(p.features)))
labelsAndPredsL2 = parsedData.map(lambda p: (p.label, model2.predict(p.features)))

trainErrL1 = labelsAndPreds.filter(lambda (v, p): v != p).count() / float(parsedData.count())
testErrL1 = labelsAndPreds.filter(lambda (v, p): v != p).count() / float(parsedTestData.count())
print("Training Error L1 = " + str(trainErrL1))
print("Test Error L1= " + str(testErrL1))

trainErrL2 = labelsAndPreds.filter(lambda (v, p): v != p).count() / float(parsedData.count())
testErrL2 = labelsAndPreds.filter(lambda (v, p): v != p).count() / float(parsedTestData.count())
print("Training Error L2 = " + str(trainErrL2))
print("Test Error L2 = " + str(testErrL2))
```

Model Evaluation: We calculated the classification error for both the algorithms to evaluate which model fits better on this data set.

Logistic Regression

```
Training Error L1 = 0.000491385399711
Test Error L1 = 0.000982740617898
Training Error L2 = 0.000491385399711
Test Error L2 = 0.000982740617898
```

SVMWithSGD

```
Training Error L1 = 0.000491385399711
Test Error L1= 0.000982740617898
Training Error L2 = 0.000491385399711
Test Error L2 = 0.000982740617898
```

Based on our observations of the classification error, we deduce that any of the 2 algorithms used here will be a good fit for the problem.

Case-3 – Clustering

Identifying commercial blocks in news videos for television broadcast analysis and monitoring by using clustering models.

Source:
https://archive.ics.uci.edu/ml/datasets/TV+News+Channel+Commercial+Detection+Dataset

Approach: We have used K-means clustering algorithms to cluster the data into predefined number of clusters. The data set is in libsvm format. We parse the data to read the values of the index and use the parsed value to train the model

Data Exploration and Data Summarization: The data is in libsvm format and has no missing values . We extracted the value from the index in order to get the parsed data to train the algorithm.

Model Cluster:
We used K-means algorithm to cluster the data.
We load and parse the data to extracts the values and then use the K-means object to cluster the data into 6,10,14 and 18 clusters. We pass the number of desired cluster, the data and the number of iterations as argument to the K-means object. We ran the algorithm for the above mentioned clusters for 20 iterations.

```
object Midterm_Q3 {

def main(args: Array[String]) {
    val conf = new SparkConf().setAppName("Spark Pi")
        /** Create the SparkContext */
        val sc = new SparkContext(conf)

        // Load and parse the data
        val data = MLUtils.loadLibSVMFile(sc, "/Users/insignia/Desktop/Big_Data_Analytics/Midterm/TV_News_Channel.
        val parsedData = data.map(s => Vectors.dense(s.features.toArray)).cache()

        // Cluster the data into two classes using KMeans
        val numClusters = 18
        val numIterations = 20
        val clusters = KMeans.train(parsedData, numClusters, numIterations)


        // Evaluate clustering by computing Within Set Sum of Squared Errors
        val WSSSE = clusters.computeCost(parsedData)
        println("Within Set Sum of Squared Errors = " + WSSSE)
```

Model Evaluation:

We compute the Within Set Sum of Squared Error (WSSSE). As the number of cluster is increased the WSSSE decreases. The optimat k is usually the one where there is an elbow in the WSSSE vs number of cluster graph

6 Clusters:

```
15/07/11 01:37:24 INFO TaskSchedulerImpl: Removed TaskSet 55.0, whose tasks have all completed, from pool
15/07/11 01:37:24 INFO DAGScheduler: Job 34 finished: sum at KMeansModel.scala:70, took 6.489288 s
Within Set Sum of Squared Errors = 6.556307464354119E10
15/07/11 01:37:24 INFO SparkContext: Invoking stop() from shutdown hook
```

10 Clusters:

```
15/07/11 02:05:05 INFO TaskSchedulerImpl: Removed TaskSet 55.0, whose tasks have all completed, from pool
15/07/11 02:05:05 INFO DAGScheduler: Job 34 finished: sum at KMeansModel.scala:70, took 10.376243 s
Within Set Sum of Squared Errors = 4.743400823740218E10
15/07/11 02:05:05 INFO SparkContext: Invoking stop() from shutdown hook
```

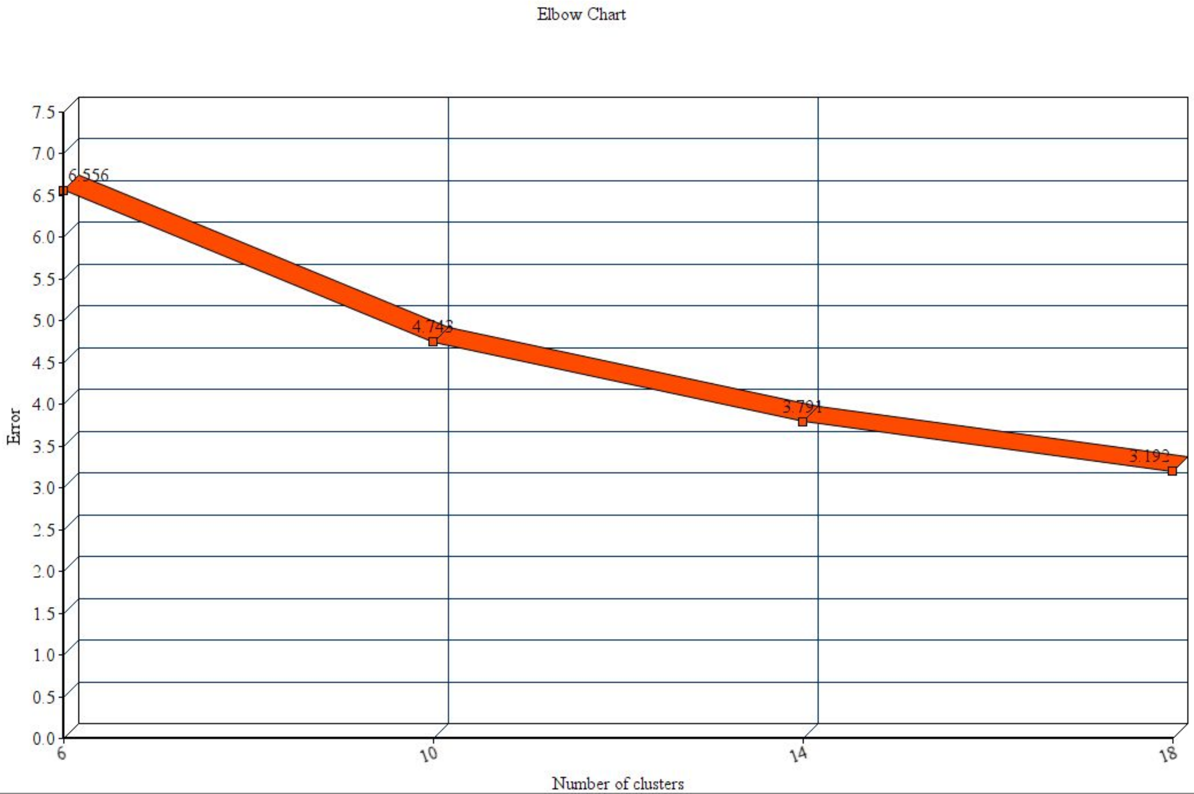
14 Clusters:

```
15/07/11 02:31:14 INFO DAGScheduler: ResultStage 55 (sum at KMeansModel.scala:70) finished in 12.249 s
15/07/11 02:31:14 INFO DAGScheduler: Job 34 finished: sum at KMeansModel.scala:70, took 12.259085 s
Within Set Sum of Squared Errors = 3.791888541643442E10
15/07/11 02:31:14 INFO SparkContext: Invoking stop() from shutdown hook
```

18 Clusters:

```
15/07/11 02:50:38 INFO DAGScheduler: Job 34 finished: sum at KMeansModel.scala:70, took 12.413750 s
Within Set Sum of Squared Errors = 3.192079357108762E10
15/07/11 02:50:38 INFO SparkContext: Invoking stop() from shutdown hook
```

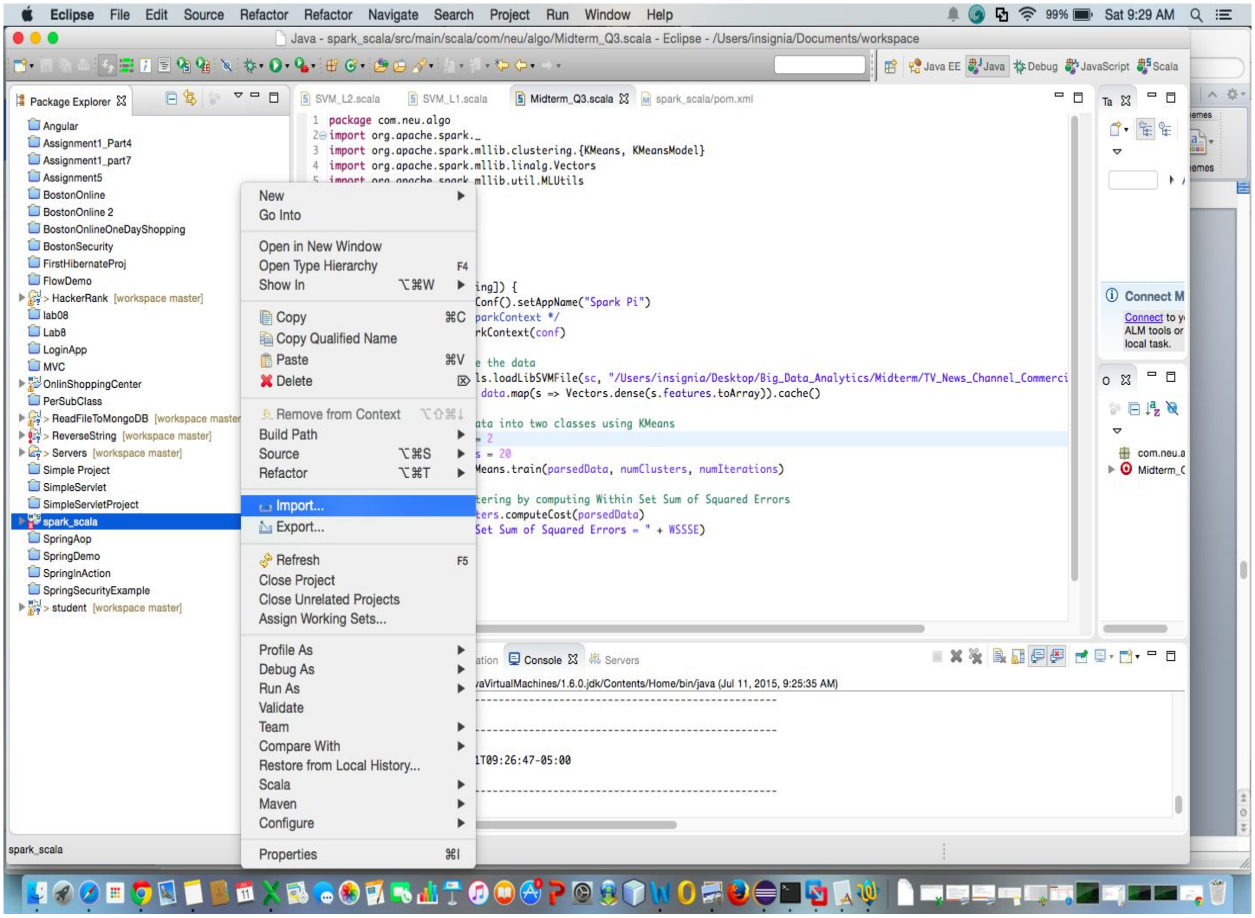
Elbow Graph:



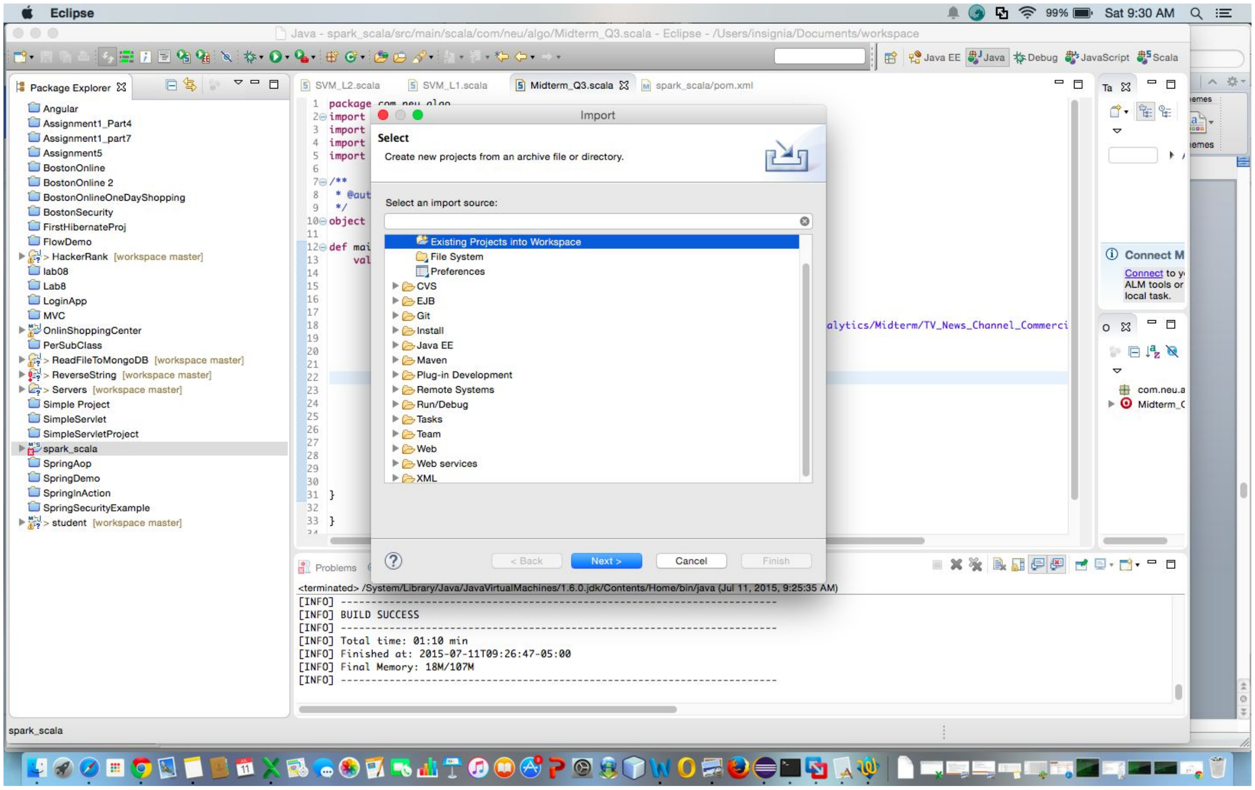
Optimal Solution **10 Clusters**

How to run the project:

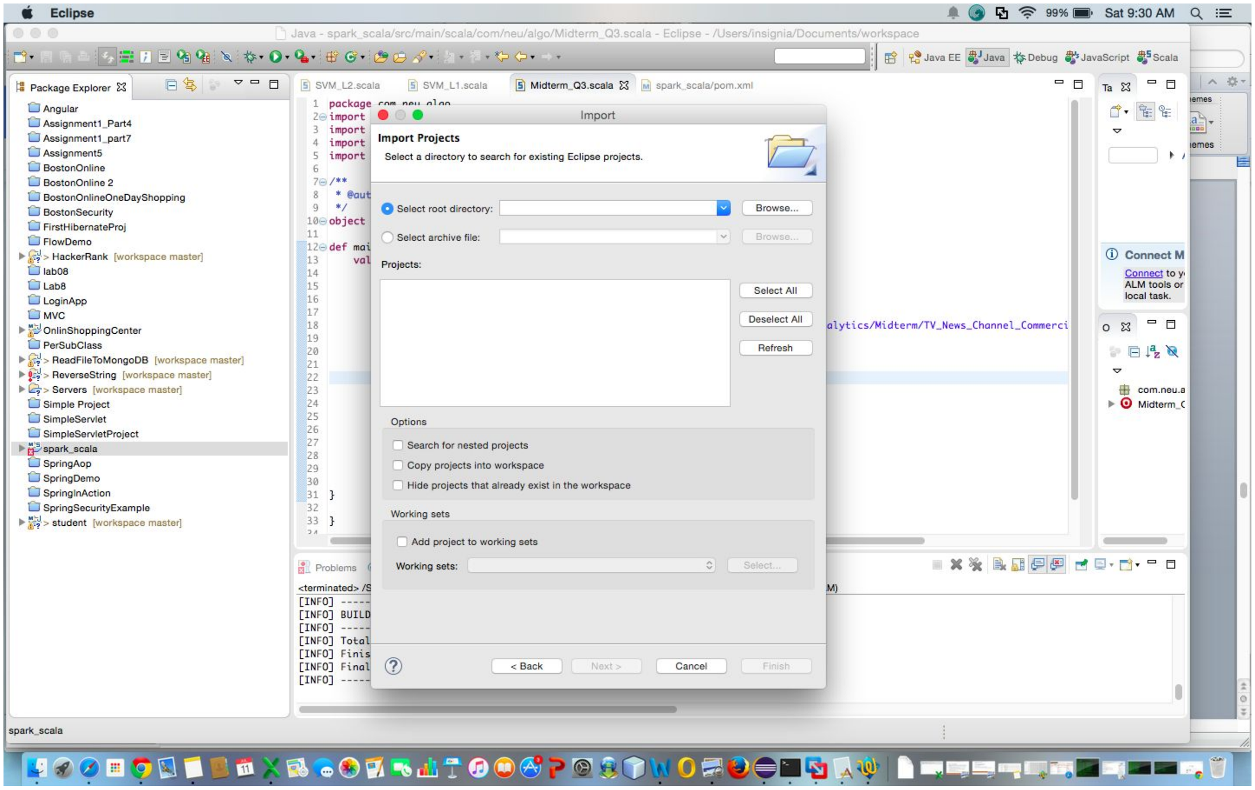
1. **Install scala plugin in Eclipse**
2. **right click and select import:**



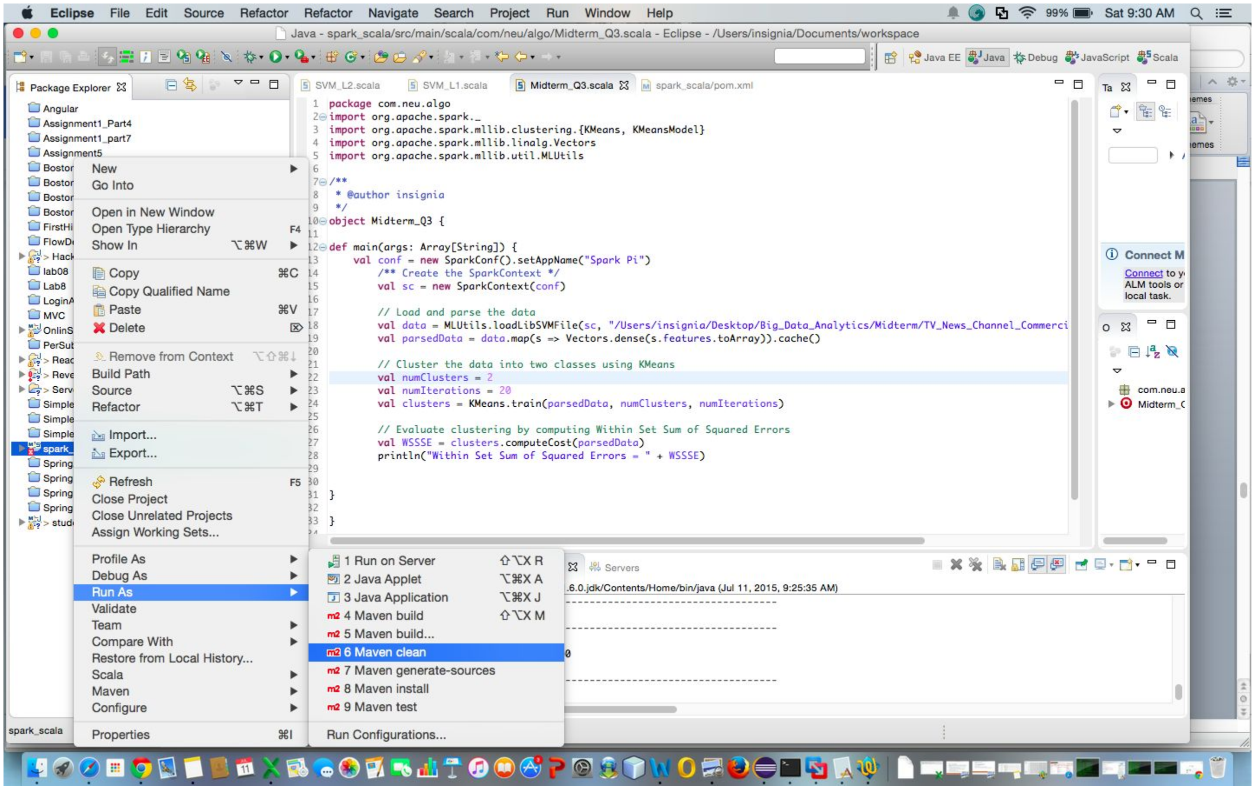
3. Then select existing project and click next :



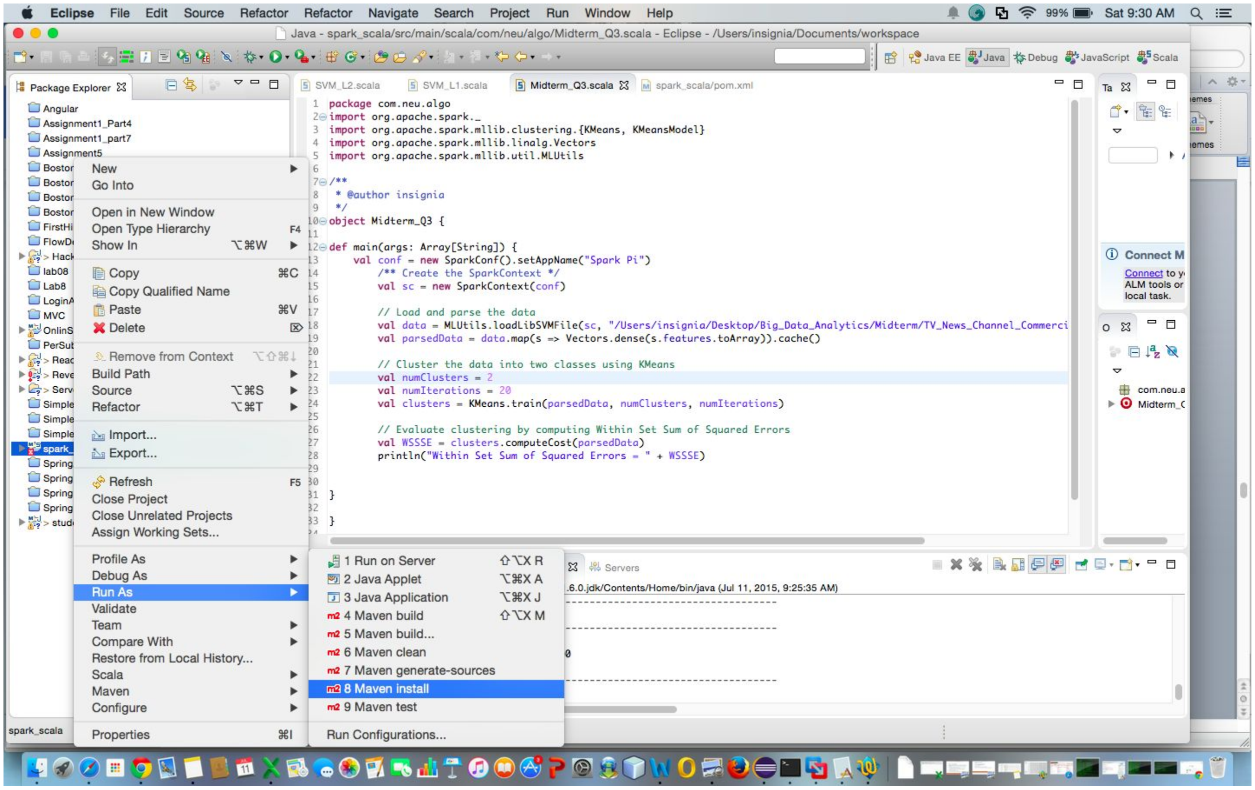
4. Browse the location where you downloaded the project and click finish:



5. Right click on the project and select maven clean:



6. After you see a Build success message in the console right click the project and select Maven install:



6. This will create the jar file under the project folder → Target folder

7. 7. Go to the same location(project folder) in the Terminal :
for me its: /Users/insignia/Downloads/spark-scala-maven-boilerplate-project-master

```
Samirs-MacBook-Air:spark-scala-maven-boilerplate-project-master insignia$ pwd
/Users/insignia/Downloads/spark-scala-maven-boilerplate-project-master
```

8. Run the following command to see the result :

/Users/insignia/Desktop/Big_Data_Analytics/spark-1.4.0/bin/spark-submit --class "com.neu.algo.Midterm_Q3" --master local[1] target/spark-scala-maven-project-0.0.1-SNAPSHOT.jar