MID TERM REPORT

Prepared by

Anirudha Deepak Bedre

Avikal Chhetri

Table of Contents

Introduction

The following report consists of three cases, namely:

Case 1: Predicting the year of song using classification and regression techniques

Case 2: Determining the income of an individual weather greater than 50k or less than 50k using classification techniques

Case 3: Using clustering model to identify commercial blocks in news videos.

Disclaimer:

We primarily used **Scala** as our algorithm of choice in the main modules of all the three cases and used **Apache Spark libraries** wherever possible.

We ran all of our Scala Code in **Apache Zeppelin** in a **Mac OS** environment.

Case 1 : Regression and Classification – Songs Dataset

Source Dataset Description

The source dataset was a comma-separated text file.

The first value is the year (target), ranging from 1922 to 2011.

The dataset contains 90 features: 12 timbre average, 78 = timbre covariance.

Data Cleansing

The data in the dataset was complete and had no missing values.

Data Transformation

For classification, the values target output column 'Year' were changed to 0, 1 for years <1965 and >1965 respectively. For regression the year value remained the same.

Principal Component Analysis (PCA) was done in addition to no feature reduction to reduce the amount of features.

All the algorithms in this case were done using the Apache ML Pipeline API and crossvalidator for model selection and fine tuning with a variety of parameters.

Strategy

Regression

Two regression algorithms – Linear Regression & Gradient Boosted Trees (GBT) Regressor were performed- each with and without PCA (feature reduction).

The following are the results we obtained:

Algorithm	Training score(RMS Error)	Test score(RMS Error)
Linear Regression Pipeline with PCA	10.46535826	10.39148337
Linear Regression Pipeline	9.588152804	9.529267232
GBT Regressor Pipeline with PCA	0.120496263	0.118432861
GBT Regressor Pipeline	0.120496263	0.118432861

Classification

For classification – Logistic Regression was performed- with and without PCA (feature reduction).

The following are the results we obtained:

Algorithm	Training score(Accuracy)	Test score(Accuracy)
Classification Pipeline with PCA	0.857498643	0.854570848
Classification Pipeline	0.894304322	0.895376243

Executive Summary

With respect to Regression, the GBT Regressor (with PCA) gave us much better results than plain old Linear Regression.

With respect to Classification, Logistic Regression results gave us an accuracy of around 89%. Unfortunately, we were unable to execute the GBT classifier or any other Tree classifier in the Spark-ML library.

Case 2 : Classification – Predicting the income (AdultDataset)

Source Dataset Description

The source dataset was a comma-separated text file.

The target value is the year income represented as >50K, <=50K for indication of an income greater than $50,000 or less than or equal to $50,000 respectively.

It had the following feature values:

age: continuous.
workclass: Private, Self-emp-not-inc, Self-emp-inc, Federal-gov, Local-gov, State-gov, Without-pay, Never-worked.
fnlwgt: continuous.
education: Bachelors, Some-college, 11th, HS-grad, Prof-school, Assoc-acdm, Assoc-voc, 9th, 7th-8th, 12th, Masters, 1st-4th, 10th, Doctorate, 5th-6th, Preschool.
education-num: continuous.
marital-status: Married-civ-spouse, Divorced, Never-married, Separated, Widowed, Married-spouse-absent, Married-AF-spouse.
occupation: Tech-support, Craft-repair, Other-service, Sales, Exec-managerial, Prof-specialty, Handlers-cleaners, Machine-op-inspct, Adm-clerical, Farming-fishing, Transport-moving, Priv-house-serv, Protective-serv, Armed-Forces.
relationship: Wife, Own-child, Husband, Not-in-family, Other-relative, Unmarried.
race: White, Asian-Pac-Islander, Amer-Indian-Eskimo, Other, Black.
sex: Female, Male.
capital-gain: continuous.
capital-loss: continuous.
hours-per-week: continuous.
native-country: United-States, Cambodia, England, Puerto-Rico, Canada, Germany, Outlying-US(Guam-USVI-etc), India, Japan, Greece, South, China, Cuba, Iran, Honduras, Philippines, Italy, Poland, Jamaica, Vietnam, Mexico, Portugal, Ireland, France, Dominican-Republic, Laos, Ecuador, Taiwan, Haiti, Columbia, Hungary, Guatemala, Nicaragua, Scotland, Thailand, Yugoslavia, El-Salvador, Trinadad&Tobago, Peru, Hong, Holand-Netherlands.

Data Cleansing

The data had a few rows of missing values indicated by a **'?'** for the columns "Workclass", "Occupation" and "Native-Country". Since the missing values were very few records (with respect to the dataset) the decision was made to delete them.

Data Transformation

For classification, the values target output column 'Income' were changed to 0, 1 for values >50k and <50k respectively.

The categorical columns were converted to numerical columns by using dummy variables.

There were issues in performing this conversion with the techniques in Apache Spark API, so we performed the conversion process alone in Python using the Pandas package and then fed the output of this file to the script in Apache Spark.

The python script used to convert the categorical values to numerical values using dummy variables:

```
%pylab inline
# Pandas package... using data frames in python
import numpy as np
import pandas as pd
from sklearn.ensemble import ExtraTreesClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.svm import SVC
from numpy import genfromtxt, savetxt
import random
df = pd.read_csv('C:\\Users\\anirudhbedre\\Downloads\\Adult
dataset\\adult.data',header=None,skipinitialspace=True)
dummies = pd.get_dummies(df.iloc[:,1])
df = df.join(dummies)
df=df.drop(["?","Federal-gov"],axis=1)
dummies2 = pd.get_dummies(df.iloc[:,5])
df = df.join(dummies2)
df=df.drop("Never-married",axis=1)
dummies3 = pd.get_dummies(df.iloc[:,6])
df = df.join(dummies3)
df=df.drop("Adm-clerical",axis=1)
df=df.drop("?",axis=1)
dummies4 = pd.get_dummies(df.iloc[:,7])
df = df.join(dummies4)
df=df.drop("Not-in-family",axis=1)
dummies5 = pd.get_dummies(df.iloc[:,8])
df = df.join(dummies5)
df=df.drop("White",axis=1)
dummies6 = pd.get_dummies(df.iloc[:,9])
```

```
df = df.join(dummies6)
df=df.drop("Male",axis=1)
dummies7 = pd.get_dummies(df.iloc[:,13])
df = df.join(dummies7)
df=df.drop("Cuba",axis=1)
df=df.drop("?",axis=1)
df=df.drop([1,3,5,6,7,8,9,13],axis=1)
cols = df.columns.tolist()
#rearranging columns  -- shifted 'hour' column in front here
cols = [0,2,4,10,11,12,'Local-gov','Never-worked','Private','Self-emp-inc','Self-emp-not-inc','State-gov','Without-pay','Divorced','Married-AF-spouse','Married-civ-spouse','Married-spouse-absent','Separated','Widowed','Armed-Forces','Craft-repair','Exec-managerial','Farming-fishing','Handlers-cleaners','Machine-op-inspct','Other-service','Priv-house-serv','Prof-specialty','Protective-serv','Sales','Tech-support','Transport-moving','Husband','Other-relative','Own-child','Unmarried','Wife','Amer-Indian-Eskimo','Asian-Pac-Islander','Black','Other','Female','Cambodia','Canada','China','Columbia','Dominican-Republic','Ecuador','El-Salvador','England','France','Germany','Greece','Guatemala','Haiti','Holand-Netherlands','Honduras','Hong','Hungary','India','Iran','Ireland','Italy','Jamaica','Japan','Laos','Mexico','Nicaragua','Outlying-US(Guam-USVI-etc)','Peru','Philippines','Poland','Portugal','Puerto-Rico','Scotland','South','Taiwan','Thailand','Trinadad&Tobago','United-States','Vietnam','Yugoslavia',14]
df = df[cols]
df.to_csv(path_or_buf='adult_withdummies.csv')
```

All the algorithms in this case were done using the Apache ML Pipeline API and crossvalidator for model selection and fine tuning with different parameters.

Strategy

Logistic Regression and Gradient Boosted Trees (GBT) Classifier were classification algorithms attempted to perform- with the Pipeline API.

Logistic Regression

After applying various parameters and fine tuning using the Pipline API and crossvalidator the best results() we obtained were:

Training Output (Accuracy) = 0.8930300922430976
Test Output (Accuracy) = 0.8925648193179296

Gradient Boosted Trees (GBT) Classifier

We attempted in trying out the GBT Classifier using the Pipeline API but ran into issues with fitting of the data. We have attached the source code along with the rest of the files.

Executive Summary

We have perform Logistic Regression with Pipeline API and obtained decent results with an accuracy of around 89%. However we weren't able to get the GBT Classifier to work in Spark.

Case 3: Clustering – TV News Channel Dataset

Source Dataset Description

The source dataset was in a Lib SVM format.

The dataset contains 12 features: 7 Audio (namely Short term energy, zero crossing rate, Spectral Centroid, spectral Flux, spectral Roll off frequency, fundamental frequency and MFCC Bag of Audio Words) and 5 visual Features (namely Video shot length, Screen Text Distribution, Motion Distribution, Frame Difference Distribution, Edge Change Ratio)

Data Cleansing

The data in the dataset was complete and had no missing values.

Data Transformation

For clustering, all the features had continuous values.

Normalization was performed on the data to avoid bias on columns.

Strategy

Two Clustering algorithms – K-means and Gaussian Mixture were performed

Clustering Analysis

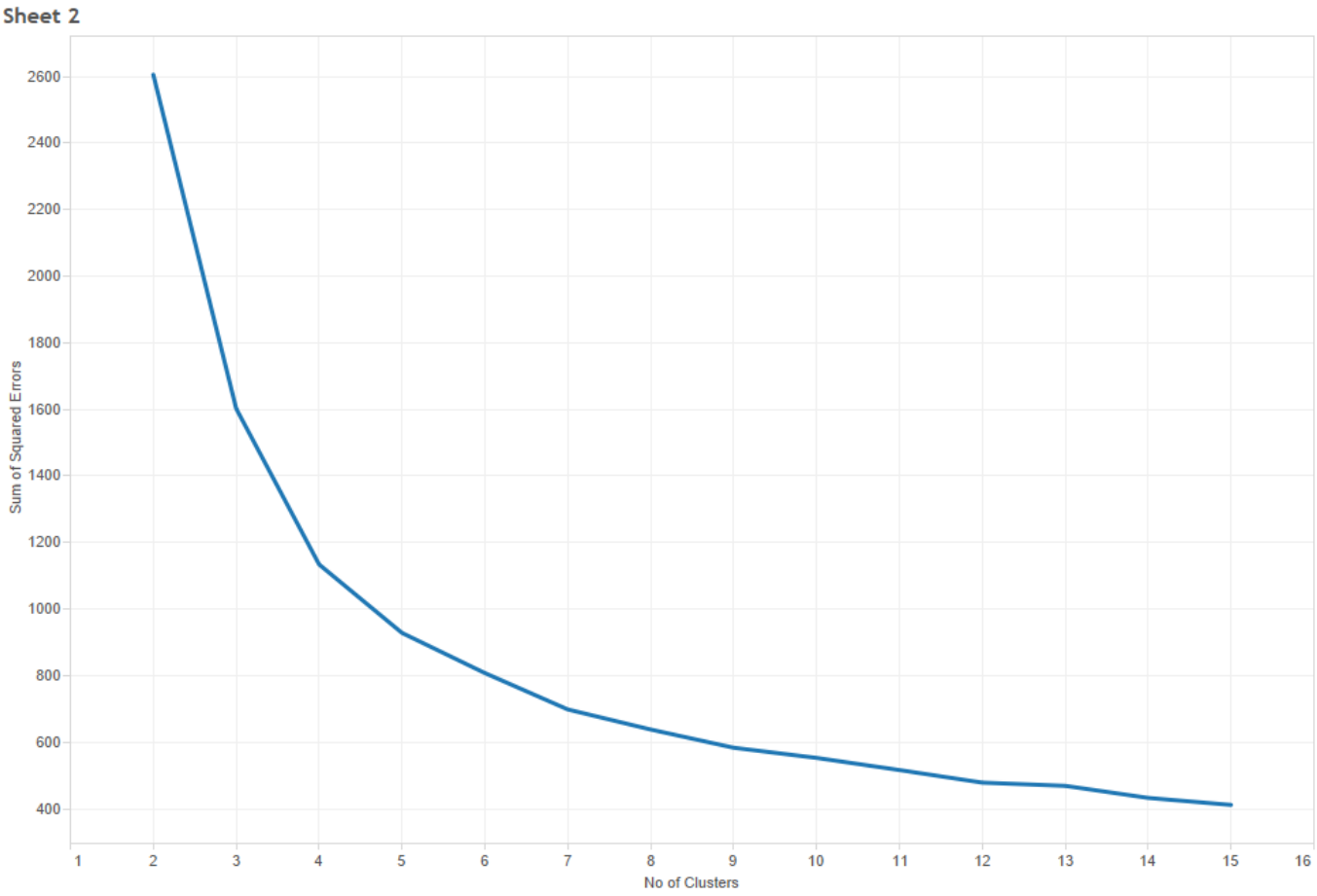
K-Means Model

Following are the Sum of Squared Errors obtained with different number of clusters and keeping the iteration cycle constant.

Number of Clusters	WSSSE
2	2605.2540840819343
3	1602.0297950672048
4	1133.9341159579435
5	928.2588138608354
6	807.6327045338741
7	698.324164966572
8	638.0527278734086

9	583.3397620174323
10	552.8560206593854
11	516.3461122909365
12	478.47017309247667
13	468.8158253601142
14	432.6391959617786
15	411.56842266444505

The following graph was plotted with the above values:



The elbow point is observed at value 'No of Clusters: 5'. Hence the ideal K value for this cluster is 5.

Executive Summary

The dataset was preprocessed using Normalizer from org.apache.spark.ml.feature API. Clusters were created successfully using K-Means and optimal number of clusters were calculated to 5 by using Within Set Sum of Squared Errors (WSSSE). We did however, get a much smaller WSSSE without normalizing it, but in that scenario we did not get a distinct elbow graph (it was in fact a bell-curve). So we stuck the normalized model which gave a distinct elbow graph even if it had a high WSSSE.

Instructions to run the code:

Datasets

The datasets are present in the 'Dataset and Code'->'Dataset Files' folder.

Dataset for Case 1: TVcombined.txt

This file can be directly fed to the scala script.

Dataset for Case 2: adult_withdummies.csv

This file can be directly fed to the scala script. This is however a preprocessed dataset, after converting the categorical values to numerical values using the Python Script which is also included. The file is ''

Dataset for Case 3: TVcombined.txt

This file can be directly fed to the scala script.

For Scala

Select Scala folder, open Scala files to and execute the files to run scala code. Make sure you set the dataset path before executing it.
We have it as .scala files and not as .jar as we faced issues in building them as an application using sbt (Spark ML dependency issue).

The list of code files included:

Case 1:
GBT_PCA.scala GBT_Pipeline.scala
Linear_Regression_PCA.scala
Linear_Regression_Pipeline.scala
Classification_PCA_Pipeline.scala
Classification_Pipeline.scala

Case 2:
Adult_GBT_Pipeline.scala
Adult_Logistic_Regression_Pipeline.scala

Case 3:
K-Means_Normalised.scala
Gaussian_Mixture_Normalized.scala

Performance Metrics used

For classification, we used the accuracy score as our metric.
For regression, we used were Mean Squared Error (MSE).
For K-Means, we used Within Set Sum of Squared Errors (WSSSE).
For Gaussian Mixture Model, we used mixture weights and individual density functions to find the most likelihood model.

Lessons learnt and challenges faced

- Many of the functions executed perfectly on the Linux/Mac environment but did not execute in Windows spark-shell because of " Out of Memory : Java Heap Space issue "
- There were also situations were certain modules worked in Apache Zepplin, but did not work in the spark-shell.
- A lot of the functionalities were still not support by Python in the Spark environment, so were forced to stick with Scala throughout the project.
- There are certain syntax (of Scala) that are not supported by Apache Zepplin like setting the parameter values with .(dot) operator.
- The documentation for a lot of the functionalities were sparse and did not provide concrete examples.
- Overall, in a way, it was a good learning experience where we tried to fix issues where there was not much scope for 'googling' (Since a lot of the code was recently published) and with sparse documentation for support.

Conclusion

The three cases were analyzed using Apache Spark successfully.