CISC 5950 Project 2

Chih-You Chen, William Cheung, Nicola Damian, Debarshi Dutta, Qiangwen Xu May 8^{th}

1 Introduction

In this project we seek to apply the machine learning library from Spark: pyspark.ml. There are various problems that we seek to solve including filtering toxic chat participants, predicting heart disease, and estimating income earnings.

2 Toxic Comment Classification

2.1 Overview

The anonymity of the online experience brought a new issue. Without the need to have accountability the darker side of humanity comes out. Toxic comments and trash talk become dominate and feeds into cyber bullying. The environment this perpetuates can infect and destroy online communities. Its important to filter out users that are toxic and block or ban them. Machine learning give an opportunity for the automation of this filtering.

2.2 Comment filtration using Linear Regression

We use the linear regression for a simple easy to train model that also is robust to large training data set by using Spark as the underlying structure. There are multiple types of toxic comments including threading (bodily harm) to identity hate (racial and sexual harm). Overall we have a toxic score that summarizes this and a snapshot can be seen in Fig. 1. The toxic score is measured as a probability from 0 to 1 of the user being extremely toxic and banning those who score a 0.5 or above. In the end we can see from Fig. 2 that about 3.5% of the population are toxic enough to ban.

id	toxic	severe_toxic	obscene	threat	insult	identity_hate
8cc3f83611bd567a	1.0	0.9473952	1.0	0.014250001	1.0	1.0
cfba4598252c5703	1.0	0.9999908	1.0	0.060498662	1.0	0.9220034
5034a94077a1a68e	1.0	1.0	1.0	1.0	1.0	1.0
c6d0510ce68e3b0a	1.0	5.2627725E-5	1.9866171E-11	0.3730006	8.2240414E-10	1.4096117E-8
1d583e75e1a2a9b4	1.0	1.4155376E-5	1.2250794E-8	1.0	1.0	1.0
fbf25ee49f4bf691	1.0	1.0	1.0	0.9999958	1.0	0.95701826
1791df1e288b1fde	1.0	0.7489234	0.9015351	0.96730214	1.0	4.59999E-5
8922ff8e65efbe38	1.0	0.99998933	1.0	0.024775784	1.0	0.8681414
2b83913e942383ea	1.0	0.013963143	1.0	0.0027850843	1.0	0.9975866
6798cdb49dceacc0	1.0	1.0	1.0	0.99796146	1.0	1.0
82478c0b09073312	1.0	0.9999638	0.99997264	1.0	1.0	1.109683E-8
ff4ae0cb03e213ca	1.0	1.0	1.0	1.0	1.0	1.0
23cd33e32ef5b1d4	1.0	8.171425E-5	0.9998504	0.0015062059	0.30396333	1.2832471E-4
aabadbfe6e3f39e4	1.0	6.3698695E-9	1.0	3.1004285E-5	1.0	1.6187292E-7
874f5a55e0ab9c0e	1.0	1.0	1.0	0.9999301	1.0	0.9067571
2756281cb6f50ab7	1.0	1.0	1.0	1.622552E-4	1.0	1.8558554E-5
9e79d3b930812e5d	1.0	0.01940128	1.0	3.8156944E-5	1.0	1.0
aba7a3f6bc3af8fc	1.0	1.0	1.0	9.195153E-8	1.0	7.497859E-10
b9ec9e3e9cacd9eb	1.0	1.0	1.0	3.2263043E-4	1.0	0.028768905
e880bf345714baf9	1.0	2.0914331E-11	1.0	1.0	1.0	9.274788E-9
4cfd17f169a50b49	1.0	1.0	1.0	5.2428177E-6	1.0	2.1311564E-12
3f7a3bc9fc30bf0a	1.0	3.4082343E-5	0.9988832	0.92131126	0.99988085	0.99997634
d98c115a0d025ec2	1.0	0.99999934	1.0	1.0	0.9999998	0.98014855
0b7efaaf471a53f2	1.0	7.941062E-7	1.0	8.527462E-6	1.0	3.97141E-9
54c6c1752f229a08	1.0	0.99975276	1.0	0.92140543	0.9999997	0.03747227
c00bfacb7253e466	1.0	0.55270857	1.0	0.009127972	1.0	0.045320973
3bda85df9f92966e	1.0	1.0	0.99999964	0.95232147	0.99358934	0.9995683
17f6f73ca52bca2a	1.0	0.0053269994	0.99999994	4.3854612E-4	0.99999815	0.90879226
0e87d795992e9329	1.0	1.0	1.0	0.012720456	1.0	0.9999916
352f365b432dbe3f	1.0	1.0	1.0	1.0	1.0	1.0
d3daf4d799fe4dac	1.0	2.278987E-11	1.546518E-19	2.286505E-6	2.4859E-41	1.7898452E-8
80dd1b73ec75f2c1	1.0	1.0	1.0	1.8336781E-4	1.0	1.0

Figure 1: Snapshot of the product of the classification.

```
+-------------------------------------------+
|sum(CASE WHEN (toxic > 0.5) THEN 1 ELSE 0 END)|
+-------------------------------------------+
|                                       5366|
+-------------------------------------------+

153164 total_people
```

Figure 2: Summary of number of toxic id persons in the community identified.

3 Heart Disease Prediction

3.1 Overview

Healthcare has been a active frontier for technological advancement. with the need to process so much medical information the assistance of machine learning is being used for things such a disease diagnostics. Here try to predict the possibility of heart disease based on measurable and self-reported health data. Some examples of these are BMI, heart rate, smoking status, and education level.

3.2 preprocessing

There are several things we have to do before training the model. First, we have to deal with the missing value. As we can see, there are 582 out of 3658 rows that contains missing value. Here, we just drop those 582 rows. Secondly, we split the data into training and testing by the percentage of 80:20. Lastly, to feed our data to the model, we have to transform it into the form of an 'vector of features' and the 'label' column. We apply Assembler.transform in this part. Example of the features used can be seen in Fig. 3

```
>>> data_assembled.show(5)
```

male	age	education	currentSmoker	cigsPerDay	BPMeds	prevalentStroke	prevalentHyp	diabetes	totChol	sysBP	diaBP	BMI	heartRate	glucose	TenYearCHD	features	label
1	39	4	0	0	0	0	0	0	195	106.0	70.0	26.97	80	77	0	[1.0,39.0,4.0,0.0...	0
0	46	2	0	0	0	0	0	0	250	121.0	81.0	28.73	95	76	0	(15,[1,2,9,10,11,...	0
1	48	1	1	20	0	0	0	0	245	127.5	80.0	25.34	75	70	0	[1.0,48.0,1.0,1.0...	0
0	61	3	1	30	0	0	1	0	225	150.0	95.0	28.58	65	103	1	[0.0,61.0,3.0,1.0...	1
0	46	3	1	23	0	0	0	0	285	130.0	84.0	23.1	85	85	0	[0.0,46.0,3.0,1.0...	0

Figure 3: features and label transformation

3.3 Model Training

We use LogisticRegression from pyspark.ml.classification here to train our model by default parameters.

3.4 Testing and Evaluation

After we get our trained model, we then apply this model to predict the label in testing data.By grouping the original label and prediction. We print out 3 metrices. Accuracy(0.86),Recall(0.08) and Precision(0.69).

3.5 Result Analysis

Although the accuracy of the model is pretty high(0.86), it mainly comes from the high TrueNegative as can seen in Fig. 4. From the extremely low recall rate, we can find that the model can't really identify the True positive cases. And from the precision, we can also see that only 0.69 of the prediction are truly positive. For the future work, we can try to improve the result by doing some feature engineering, dealing with data imbalance problem or trying to tune the parameters.

label	prediction	count
1	0.0	99
0	0.0	639
1	1.0	9
0	1.0	4

```
('testing data Accuracy:', 0.8628495339547271)
('testing data Recall:', 0.08333333333333333)
('testing data Precision:', 0.6923076923076923)
```

Figure 4: performance summary

4 Income Prediction

4.1 Overview

Estimating income is an extremely important tool for banks and governments. It can determine the about of loan to provide to the amount of financial assistance needed for a region. With the features looked into it cannot be used in private setting due to protection on judgment based on race and gender but from a government perspective it helpful. Fig. 5 show a sample of the data used.

```
   age   workclass   fnlwgt      education  education_num       marital_status  ...     sex  capital_gain  capital_loss  hours_per_week  native_country  la
1   25     Private  226802.0          11th            7.0        Never-married  ...    Male           0.0           0.0            40.0   United-States  <=
2   38     Private   89814.0       HS-grad            9.0   Married-civ-spouse  ...    Male           0.0           0.0            50.0   United-States  <=
3   28   Local-gov  336951.0    Assoc-acdm           12.0   Married-civ-spouse  ...    Male           0.0           0.0            40.0   United-States   >
4   44     Private  160323.0  Some-college           10.0   Married-civ-spouse  ...    Male        7688.0           0.0            40.0   United-States   >
5   18           ?  103497.0  Some-college           10.0        Never-married  ...  Female           0.0           0.0            30.0   United-States  <=

[5 rows x 15 columns]
```

Figure 5: Sample of feature types used

4.2 Model Training and Evaluation

For similar reasons as stated in Section 2.2 we use Spark and Linear Regression. In this simplified version of the income prediction question we are making a binary prediction of whether or not the individual earn more than $50,000 or not. Fig. 6 show prediction examples. The performance of the model is captured in Fig. 7

```
+--------------------+------+--------------------+--------------------+----------+
|            features|labelI|       rawPrediction|         probability|prediction|
+--------------------+------+--------------------+--------------------+----------+
|(114,[5,10,25,38,...|   1.0|[-1.3280373161575...|[0.20948420145479...|       1.0|
|(114,[0,10,25,40,...|   0.0|[1.17938237876484...|[0.76483673565065...|       0.0|
|(114,[0,10,25,38,...|   0.0|[-0.8566316180662...|[0.29804357724603...|       1.0|
|(114,[3,8,23,42,5...|   0.0|[4.03475647717149...|[0.98261750855203...|       0.0|
|(114,[0,10,25,40,...|   0.0|[2.12760223584631...|[0.89355716547904...|       0.0|
+--------------------+------+--------------------+--------------------+----------+
only showing top 5 rows
```

Figure 6: prediction examples

```
2020-05-06 15:48:27 INFO  TaskSchedulerImpl:54 - Adding task set 267.0 with 1 tasks
2020-05-06 15:48:27 INFO  TaskSetManager:54 - Starting task 0.0 in stage 267.0 (TID 152, localhost, executor driver, partition 0, ANY, 7662 bytes)
2020-05-06 15:48:27 INFO  Executor:54 - Running task 0.0 in stage 267.0 (TID 152)
2020-05-06 15:48:27 INFO  ShuffleBlockFetcherIterator:54 - Getting 1 non-empty blocks including 1 local blocks and 0 remote blocks
2020-05-06 15:48:27 INFO  ShuffleBlockFetcherIterator:54 - Started 0 remote fetches in 1 ms
2020-05-06 15:48:27 INFO  Executor:54 - Finished task 0.0 in stage 267.0 (TID 152). 1367 bytes result sent to driver
2020-05-06 15:48:27 INFO  TaskSetManager:54 - Finished task 0.0 in stage 267.0 (TID 152) in 29 ms on localhost (executor driver) (1/1)
2020-05-06 15:48:27 INFO  TaskSchedulerImpl:54 - Removed TaskSet 267.0, whose tasks have all completed, from pool
2020-05-06 15:48:27 INFO  DAGScheduler:54 - ResultStage 267 (countByValue at MulticlassMetrics.scala:42) finished in 0.034 s
2020-05-06 15:48:27 INFO  DAGScheduler:54 - Job 132 finished: countByValue at MulticlassMetrics.scala:42, took 0.261581 s
Test set accuracy: 0.853448805356
```

Figure 7: model test results

5 Decision Tree and Random Forest Experimentation

5.1 Overview

Throughout the three questions we stick to the linear regression as the base algorithm of our models. However its important to look at other possibilities in order to see if other algorithms perform better. We base question 3 to implement the other 2 algorithms, decision tree and random forest.

5.2 Performance

It's important to try different algorithms when testing your pipeline. Even with domain knowledge its difficult to know which algorithm performs best with your data and give more wright to the algorithms that preforms best. As shown in Fig. 8 by trying Random Forest we found a higher performing version of the model.

We combined 50 decision trees in random forest model. It obtained satisfying accuracy. Thus, we became curious that how would the single decision tree perform? With that idea, we implemented decision tree model as well.

While putting the evaluation of two models together, as shown in Fig. 9, we find that actually the decision tree got a little better accuracy than random forest in both training set and test set. However, when it comes to precision, the decision tree sharply drop to somewhat over 50% which is similar to random guessing. Meanwhile the random forest gave us great precision over 89% in both training and test.

```
***********
[+] Evaluation on test set:
>>> pred_test = rf.transform(testsetreg)
>>> rf_test_accuracy = MulticlassClassificationEvaluato
>>> print("RF's Test Accuracy is %.4f"%rf_test_accuracy)
RF's Test Accuracy is 0.8216
>>> rf_test_auc = BinaryClassificationEvaluator(labelCo
>>> print("RF's Test Precision is %.4f"%rf_test_auc)
RF's Test Precision is 0.8900
>>>
```

Figure 8: Accuracy of Random Forest

```
20/05/06 16:27:42 INFO DAGScheduler: Job 51 finished: aggregate at AreaUn
20/05/06 16:27:42 INFO MapPartitionsRDD: Removing RDD 328 from persistenc
20/05/06 16:27:42 INFO BlockManager: Removing RDD 328
[*] Random Forest:
[-] <Train> Accuracy is 0.8175
[-] <Train> Precision is 0.8929
[-] <Test> Accuracy is 0.8187
[-] <Test> Precision is 0.8916
[*] Decision Tree:
[-] <Train> Accuracy is 0.8374
[-] <Train> Precision is 0.5037
[-] <Test> Accuracy is 0.8357
[-] <Test> Precision is 0.5164
20/05/06 16:27:42 INFO SparkContext: Invoking stop() from shutdown hook
```

Figure 9: Evaluation of Random Forest and Decision Tree