Click Here for Makeene Learning

Spencer Chan, *Millionth Visitor*, and Kelvin Lin, *Internet Prize Winner*

***Abstract*—In this project, we entered the Avazu click-through-rate (CTR) prediction contest. The aim was to examine a list of advertisements and their features, ranging from click, ad type, placement, website, and device model. We were to use the data of ads collected over 10 days, and predict the CTR of one day of ads. We used multiple methods, such as KNN classification, stochastic gradient descent (SGD) classication, and logistic regression to attempt to properly predict the CTR. This contest was sponsored by the company Avazu, for a grand prize of $5000.**

***Keywords*—Click-Through-Rate (CTR), Stochastic Gradient Descent (SGD), K-Nearest-Neighbors (KNN)**

I. Introduction

In this project, we experimented with the CTR of advertisements in many forms, and output our expected probability of garnering a click. Advertisement is one of the biggest areas of the working world which have endless amounts of money poured into it. If you can introduce your product to more people, you can gain popularity over the competition and become even more successful. One 30-second commercial at the most recent Super Bowl XLVIII cost $4 millions dollars. It is extremely important to have your ads appeal to the customer, so figuring out which ads perform the best and receive the most clicks or views is a high concern for many companies.

II. The Data

The data that was provided to us was an accumulation of 10-days worth of click-through data. The training set consisted of 40 million points, each with an ID number and 23 features. The test data provided contains 4.7 million data points collected over one day's time. The features include click, hour, site information, app information, device information and various anonymized variables. These variables were mostly integers, but categorical data such as device type, ad category, and ad placement were 6 character hex strings. In order to reduce dimensionality and deal with the problem of processing the hex strings, these features were discarded.

The benchmark for this data is a constant value prediction, assigning a probability of 0.5 to all IDs. This equates to a log loss of 0.6931472. This benchmark is not too difficult to overcome, as the actual probability of an ad being clicked is far lower than 50 percent.

III. Software

Due to the large data size, MATLAB prt is a poor choice due to its higher memory requirements compared to other machine learning suites. For this dataset, the python packages sklearn and numpy were used instead. While numpy has its own functions for reading in data, it was unable to process the IDs of each data point being read in. There were always some discrepancies between the input IDs and output IDs, probably due to the fact that numpy insists on converting the IDs to floats before writing it as a string.

Using the sklearn package, reading in the full training and test datasets utilize approximately 700MB of memory. Training on our dataset however, takes much more memory. Often the training of our model would use anywhere up to 10.2GB of memory at a time. This was particularly difficult as some classifiers such as KNN can take more than 24 hours to train, if they complete at all.

IV. Experimentation

For this data set, 14 percent of the ads resulted in a click. Our first submission was a constant value for all IDs. We chose to utilize the mean due to the fact that the mean minimizes the log loss for a constant submission. The constant submission returned a log loss of 0.4436101.

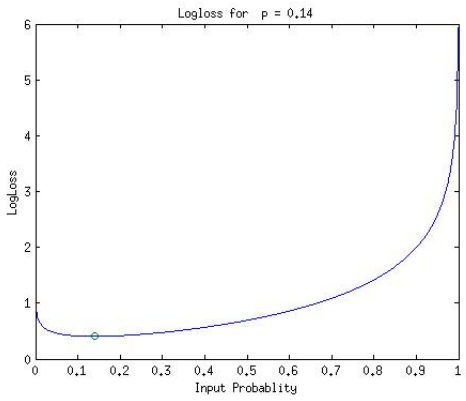

Fig. 1. Logarithmic loss for p = 0.14

Our first attempt was to utilize KNN-regression to create a probability of click-through. Compared to a normal KNN classifier, a KNN regressor takes the average of the nearest neighbors, allowing discrete data points to be mapped into continuous space. Using only the numeric data points, we implemented the sklearn package in Python to process the data, and perform the classification. this resulted in a log loss of about 0.45, slightly worse than a constant mean probability.

KNN was also especially difficult to work with due to the large memory requirements. Given that we were utilizing 10 features over 40 million training points and 4.7 million test points, the matrix required to store the neighbor matrix is very large. As a result, we were not able to train a KNN regressor on the full dataset. The largest training set we submitted included only 500 thousand training points. The small training set caused many problems as well. Given the sparseness of the

training set compared to the test set, many test points included many successful ads within their nearest neighbors - far more than the average. As a result, many of the unsuccessful KNN outputs we obtained had a log loss greater than 2.

Logistic regression was our second attempt. Logistic regression provided much better results for our dataset, likely due to the fact that our data is simply a series of bernoulli trials. Python numpy was unable to handle the full size of our dataset. When we tried casting an array from 2d to 1d in order to send it as input to the logistic regressor, our python script would crash - likely due to memory constraints. In the end, our final submission was trained on only 8 million data points. Even with this restriction, we were able to beat the 0.14 constant value submission with a log loss of 0.4408581.

The logistic regressor was trained with multiple cost values, ranging from 0 to 10^5. We found that the lowest log loss resulted from having no regularization. With 8 million data points however, it must be noted that this is likely the result of an overfit, and may not actually generalize well for all datasets.

The last technique utilized was Stochastic Gradient Descent (SGD). SGD is an optimization method technique for minimizing an objective function modeling the system in the form of a sum. SGD is a popular method of training for other machine learning techniques such as support vector machines (SVM) and the aforementioned logistic regression. However, on its own, it did not perform as well, as logistic regression.

V. Results

Results for this contest were evaluated based on the average logarithmic loss for the submission. Logarithmic loss allows for a simple method of evaluating error in Bernoulli trials as well as heavily penalizing incorrect predictions. Any incorrect prediction with 100 percent confidence will result in an infinite amount of log loss.

Classifier	Log Loss Score
Logistic Regression C = 10^3	0.4408581
Logistic Regression C = 0	0.4415889
Logistic Regression C = 10^5	0.4415889
Constant 0.14	0.4436101
KNN - 10 Neighbors[1]	0.4815247
Benchmark 0.5	0.6931472
SGD[2]	1.1496145
KNN - 5 Neighbors	2.4559108

TABLE I. Different techniques for classification, by score

As you can see, logistic regression performed the best, while the others, especially SGD, seemed to fail. We attempted multiple iterations of SGD, but none reached past the mark of 1.0 log loss. KNN was extremely computationally intensive, and performed worse than logistic regression, with more variance in results.

[1]Entries equal to zero were changed to 0.1

[2]Zeroes were changed to 0.14 and ones were changed to 0.86

VI. Conclusion

While we were able to get slightly better results compared to the constant mean input, these improvements were not significant. The reason for the lack of improvment from the mean benchmark is likely due to the method of feature reduction. In this case, data related to ad and device category were discarded, but it is likely that this data is more relevant to the actual behaviour of the data than the features we are currently using.

In the future, we could implement a method of mapping the categorical data into a form which is easier to manipulate. This would involve iterating through the data to figure out how many categories for each feature there are, and possibly implementing a one-hot code to separate the categories. A simple method of doing this would be to use a hash table to map each category into numerical values and then count the number of table elements which are filled.

APPENDIX
PYTHON SOURCE CODE

```
import csv
import numpy as np
import timeit
from sklearn import linear_model
from sklearn import neighbors
import time

# train_rev2.csv has 47686351 data points
# train_small.csv has 199229 data points
# train_smallest.csv has 99 data points

# test_rev2 has 4769401 data points
# test_small has 999 data points

# read in test data
f_len = 4577464

classified = 0.14*np.ones((f_len,1))
#print(classified)
print(classified.shape)
#print(ID)
np.savetxt("./min_log_loss.csv",classified,fmt='%s',delimiter=',')

import csv
import numpy as np
import timeit
from sklearn import linear_model
from sklearn import neighbors
import time

# file io
fname = "train_small.csv"
if fname == "train_small.csv":
    f_len = 500000
else:
    if fname ==  "train_smallest.csv":
        f_len = 99
    else:
        if fname == "train":
            f_len = 40428967
        else:
            f_len = 0

file_in = open(fname)
csv_reader = csv.reader(file_in)
header = next(csv_reader)
targets = np.zeros((f_len,1))
features = np.zeros((f_len,13))
index = 0

# iterate over each row and read in data
start = time.time()
for row in csv_reader:
    if index == f_len:
        print("index has exceeded matrix dimensions \nwriting data to file")
```

```
            break
        targets[index] = np.array(row[1]).astype(np.float)
        features[index,:]=np.array(row[2:5]+row[14:24]).astype(np.float)
        index = index + 1
end = time.time()
file_in.close()
out = np.hstack((targets,features))
print("Feature matrix dimensions: ",out.shape)
#np.savetxt("./features_rev2.csv",out,fmt='%s',delimiter=',')
print("It took ",end-start," seconds to read in training data")



# test_rev2 has 4769401 data points
# test_small has 999 data points

# read in test data
fname = "test"
if fname == "test":
    f_len = 4577464
else:
    if fname == "test_small.csv":
        f_len = 199999
    else:
        f_len = 0

file_in = open(fname)
csv_reader = csv.reader(file_in)
header = next(csv_reader)
ID = np.array((f_len,),dtype='object')
test_features = np.zeros((f_len,13))
index = 0

# iterate over each row and read in data
start = time.time()
for row in csv_reader:
    if index == f_len:
        print("index has exceeded matrix dimensions \nwriting data to file")
        break
    #ID[index] = row[0]
  #  t = np.array(row[1:4]+row[15:17]+row[18:26])
 #   print(t.shape,row[1:4])
    test_features[index,:]=np.array(row[1:4]+row[13:23]).astype(np.float)
    index = index + 1
end = time.time()
file_in.close()
print("Test feature matrix dimensions: ",test_features.shape)
#np.savetxt("./test_features_smallest.csv",out,fmt='%s',delimiter=',')
print("It took ",end-start," seconds to read in test data")

# create and train model
# KNN regression
start = time.time()
n_neighbors = 10
classifier = neighbors.KNeighborsRegressor(n_neighbors,weights='uniform')

classifier = classifier.fit(features,targets)
end = time.time()
```

```
print("It took ",end-start," seconds to train the model")

# use model to preduct outputs
start = time.time()
classified = classifier.predict(test_features)
end = time.time()
print("It took ",end-start," seconds to classify the data")
#print(classified)
print(classified.shape)
#print(ID)

print("correcting zeros")
for x in classified:
    if x == 0:
        x[...] = 0.1

np.savetxt("./test_classified.csv",classified,fmt='%s',delimiter=',')

import csv
import numpy as np
import timeit
from sklearn import linear_model
from sklearn import neighbors
import time

# file io
fname = "train"
if fname == "train_small.csv":
    f_len = 500000
else:
    if fname ==  "train_smallest.csv":
        f_len = 99
    else:
        if fname == "train":
            f_len = 40428967
        else:
            f_len = 0

file_in = open(fname)
csv_reader = csv.reader(file_in)
header = next(csv_reader)
targets = np.zeros((f_len,1))
features = np.zeros((f_len,13))
index = 0

# iterate over each row and read in data
start = time.time()
for row in csv_reader:
    if index == f_len:
        print("index has exceeded matrix dimensions \nwriting data to file")
        break
    targets[index] = np.array(row[1]).astype(np.float)
    features[index,:]=np.array(row[2:5]+row[14:24]).astype(np.float)
    index = index + 1
end = time.time()
file_in.close()
out = np.hstack((targets,features))
print("Feature matrix dimensions: ",out.shape)
```

```
#np.savetxt("./features_rev2.csv",out,fmt='%s',delimiter=',')
print("It took ",end-start," seconds to read in training data")



# test_rev2 has 4769401 data points
# test_small has 999 data points

# read in test data
fname = "test"
if fname == "test":
    f_len = 4577464
else:
    if fname == "test_small.csv":
        f_len = 199999
    else:
        f_len = 0

file_in = open(fname)
csv_reader = csv.reader(file_in)
header = next(csv_reader)
ID = np.array((f_len,),dtype='object')
test_features = np.zeros((f_len,13))
index = 0

# iterate over each row and read in data
start = time.time()
for row in csv_reader:
    if index == f_len:
        print("index has exceeded matrix dimensions \nwriting data to file")
        break
    #ID[index] = row[0]
  #  t = np.array(row[1:4]+row[15:17]+row[18:26])
 #   print(t.shape,row[1:4])
    test_features[index,:]=np.array(row[1:4]+row[13:23]).astype(np.float)
    index = index + 1
end = time.time()
file_in.close()
print("Test feature matrix dimensions: ",test_features.shape)
#np.savetxt("./test_features_smallest.csv",out,fmt='%s',delimiter=',')
print("It took ",end-start," seconds to read in test data")

# create and train model
# logistic regression
start = time.time()
n_neighbors = 10
#classifier = neighbors.KNeighborsRegressor(n_neighbors,weights='uniform')
classifier = linear_model.LogisticRegression(C=1e3)
classifier = classifier.fit(features,np.ravel(targets))
end = time.time()
print("It took ",end-start," seconds to train the model")

# use model to preduct outputs
start = time.time()
classified = classifier.predict_proba(test_features)
end = time.time()
print("It took ",end-start," seconds to classify the data")
#print(classified)
```

```
print(classified[:,1].shape)
#print(ID)

'''
print("correcting zeros")
for x in classified:
    if x == 0:
        x[...] = 0.1
'''
np.savetxt("./test_classified.csv",classified[:,1],fmt='%s',delimiter=',')

import csv
import numpy as np
import timeit
from sklearn import linear_model
from sklearn import neighbors
import time

# file io
fname = "train"
if fname == "train_small.csv":
    f_len = 500000
else:
    if fname ==  "train_smallest.csv":
        f_len = 99
    else:
        if fname == "train":
            f_len = 40428967
        else:
            f_len = 0

file_in = open(fname)
csv_reader = csv.reader(file_in)
header = next(csv_reader)
targets = np.zeros((f_len,1))
features = np.zeros((f_len,13))
index = 0

# iterate over each row and read in data
start = time.time()
for row in csv_reader:
    if index == f_len:
        print("index has exceeded matrix dimensions \nwriting data to file")
        break
    targets[index] = np.array(row[1]).astype(np.float)
    features[index,:]=np.array(row[2:5]+row[14:24]).astype(np.float)
    index = index + 1
end = time.time()
file_in.close()
out = np.hstack((targets,features))
print("Feature matrix dimensions: ",out.shape)
#np.savetxt("./features_rev2.csv",out,fmt='%s',delimiter=',')
print("It took ",end-start," seconds to read in training data")



# test_rev2 has 4769401 data points
# test_small has 999 data points
```

```
# read in test data
fname = "test"
if fname == "test":
    f_len = 4577464
else:
    if fname == "test_small.csv":
        f_len = 199999
    else:
        f_len = 0

file_in = open(fname)
csv_reader = csv.reader(file_in)
header = next(csv_reader)
ID = np.array((f_len,),dtype='object')
test_features = np.zeros((f_len,13))
index = 0

# iterate over each row and read in data
start = time.time()
for row in csv_reader:
    if index == f_len:
        print("index has exceeded matrix dimensions \nwriting data to file")
        break
    #ID[index] = row[0]
  #  t = np.array(row[1:4]+row[15:17]+row[18:26])
 #   print(t.shape,row[1:4])
    test_features[index,:]=np.array(row[1:4]+row[13:23]).astype(np.float)
    index = index + 1
end = time.time()
file_in.close()
print("Test feature matrix dimensions: ",test_features.shape)
#np.savetxt("./test_features_smallest.csv",out,fmt='%s',delimiter=',')
print("It took ",end-start," seconds to read in test data")

# create and train model
# logistic regression
start = time.time()
n_neighbors = 10
#classifier = neighbors.KNeighborsRegressor(n_neighbors,weights='uniform')
classifier = linear_model.SGDClassifier(loss='log')
classifier = classifier.fit(features,np.ravel(targets))
end = time.time()
print("It took ",end-start," seconds to train the model")

# use model to preduct outputs
start = time.time()
classified = classifier.predict_proba(test_features)
end = time.time()
print("It took ",end-start," seconds to classify the data")
#print(classified)
print(classified.shape)
#print(ID)

for x in classified:
    if x[1] == 0:
        x[0] = .14
    elif x[1] == 1:
```

```
        x[0] = .86
'''
print("correcting zeros")
for x in classified:
    if x == 0:
        x[...] = 0.1
'''
np.savetxt("./test_classified.csv",classified[:,0],fmt='%s',delimiter=',')
```

REFERENCES

[1] C, Bishop, *Pattern Recognition and Machine Learning (Information Science and Statistics)*, ISBN:0387310738, Springer-Verlag New York, Inc 2006.