Discussion Essay
Regularization

Submitted by:

Ishant Nayer

M10669373

Dataset

This dataset includes 5671 requests collected from the Reddit community Random Acts of Pizza between December 8, 2010 and September 29, 2013.

Following is the description of the variables:

"giver_username_if_known": Reddit username of giver if known, i.e. the person satisfying the request ("N/A" otherwise).

"number_of_downvotes_of_request_at_retrieval": Number of downvotes at the time the request was collected.

"number_of_upvotes_of_request_at_retrieval": Number of upvotes at the time the request was collected.

"post_was_edited": Boolean indicating whether this post was edited (from Reddit).

"request_id": Identifier of the post on Reddit, e.g. "t3_w5491".

"request_number_of_comments_at_retrieval": Number of comments for the request at time of retrieval.

"request_text": Full text of the request.

"request_text_edit_aware": Edit aware version of "request_text". We use a set of rules to strip edited comments indicating the success of the request such as "EDIT: Thanks /u/foo, the pizza was delicous".

"request_title": Title of the request.

"requester_account_age_in_days_at_request": Account age of requester in days at time of request.

"requester_account_age_in_days_at_retrieval": Account age of requester in days at time of retrieval.

"requester_days_since_first_post_on_raop_at_request": Number of days between requesters first post on RAOP and this request (zero if requester has never posted before on RAOP).

"requester_days_since_first_post_on_raop_at_retrieval": Number of days between requesters first post on RAOP and time of retrieval.

"requester_number_of_comments_at_request": Total number of comments on Reddit by requester at time of request.

"requester_number_of_comments_at_retrieval": Total number of comments on Reddit by requester at time of retrieval.

"requester_number_of_comments_in_raop_at_retrieval": Total number of comments in RAOP by requester at time of retrieval.

"requester_number_of_posts_at_request": Total number of posts on Reddit by requester at time of request.

"requester_number_of_posts_at_retrieval": Total number of posts on Reddit by requester at time of retrieval.

"requester_number_of_posts_on_raop_at_request": Total number of posts in RAOP by requester at time of request.

"requester_number_of_posts_on_raop_at_retrieval": Total number of posts in RAOP by requester at time of retrieval.

"requester_number_of_subreddits_at_request": The number of subreddits in which the author had already posted in at the time of request.

"requester_received_pizza": Boolean indicating the success of the request, i.e., whether the requester received pizza.

"requester_subreddits_at_request": The list of subreddits in which the author had already posted in at the time of request.

"requester_upvotes_minus_downvotes_at_request": Difference of total upvotes and total downvotes of requester at time of request.

"requester_upvotes_minus_downvotes_at_retrieval": Difference of total upvotes and total downvotes of requester at time of retrieval.

"requester_upvotes_plus_downvotes_at_request": Sum of total upvotes and total downvotes of requester at time of request.

"requester_upvotes_plus_downvotes_at_retrieval": Sum of total upvotes and total downvotes of requester at time of retrieval.

"requester_user_flair": Users on RAOP receive badges (Reddit calls them flairs) which is a small picture next to their username. In our data set the user flair is either None (neither given nor received pizza, N=4282), "shroom" (received pizza, but not given, N=1306), or "PIF" (pizza given after having received, N=83).

"requester_username": Reddit username of requester.

"unix_timestamp_of_request": Unix timestamp of request (supposedly in timezone of user, but in most cases it is equal to the UTC timestamp -- which is incorrect since most RAOP users are from the USA).

"unix_timestamp_of_request_utc": Unit timestamp of request in UTC.

Reading the Data

We have used the csv import feature of pandas.

Syntax:

Import pandas
train_data = pd.read_csv('PizzaTrain.csv')

Training and test datasets are given.

PizzaTrain = 4040 observations, 32 variables

PizzaTest = 1631 observations, 17 variables

PizzaTest dataset is not upto the mark. It has been divided unevenly as it does not contain all the variables. I will further divide my training dataset into 80-20% ratio for further analysis.

The new training data has 3232 observations. Also, as our objective is to compare the models with and without regularization, we will only consider numerical variables for simplification.

The response variable is requester_received_pizza it takes 1 when the requester recieved pizza and 0 otherwise.

Variable type:

```
>>> >>> >>> number_of_downvotes_of_request_at_retrieval              int64
number_of_upvotes_of_request_at_retrieval                 int64
request_number_of_comments_at_retrieval                   int64
requester_account_age_in_days_at_request                float64
requester_account_age_in_days_at_retrieval              float64
requester_days_since_first_post_on_raop_at_request      float64
requester_days_since_first_post_on_raop_at_retrieval    float64
requester_number_of_comments_at_request                   int64
requester_number_of_comments_at_retrieval                 int64
requester_number_of_comments_in_raop_at_request           int64
requester_number_of_comments_in_raop_at_retrieval         int64
requester_number_of_posts_at_request                      int64
requester_number_of_posts_at_retrieval                    int64
requester_number_of_posts_on_raop_at_request              int64
requester_number_of_posts_on_raop_at_retrieval            int64
requester_number_of_subreddits_at_request                 int64
requester_received_pizza                                   bool
requester_upvotes_minus_downvotes_at_request              int64
requester_upvotes_minus_downvotes_at_retrieval            int64
requester_upvotes_plus_downvotes_at_request               int64
requester_upvotes_plus_downvotes_at_retrieval             int64
unix_timestamp_of_request                                 int64
unix_timestamp_of_request_utc                             int64
dtype: object
```

Missing Values

train_data.describe()

A Sample indicating no missing values.

	...	requester_number_of_posts_at_retrieval	\
count	...	4040.000000	
mean	...	41.151733	
std	...	80.798543	
min	...	0.000000	
25%	...	2.000000	
50%	...	13.000000	
75%	...	46.000000	
max	...	999.000000	

	requester_number_of_posts_on_raop_at_request	\
count	4040.000000	
mean	0.063614	
std	0.325773	
min	0.000000	
25%	0.000000	
50%	0.000000	
75%	0.000000	
max	5.000000	

	requester_number_of_posts_on_raop_at_retrieval	\
count	4040.000000	
mean	1.239109	
std	0.603083	
min	0.000000	
25%	1.000000	
50%	1.000000	
75%	1.000000	
max	9.000000	

GBM Model

GBM was used for predicting whether requester received pizza or not. Further, two GBM were made, one with regularization and the other without regularization.

Regularization: It is a technique used to avoid overfitting problem. The models that overfit the data are complex models and have for example too many parameters. This technique helps in reducing model complexity by penalizing the objective function there by handling the problem of overfitting.

For GBM learning rate < 1 means that the model is regularized.

CODE:

```
X = train_data1.drop('requester_received_pizza', axis = 1)
Y = train_data1['requester_received_pizza']

X_train, X_test, Y_train, Y_test = train_test_split(X, Y,
test_size = 0.20, random_state = None)

print(X_train.shape)
print(Y_train.shape)

#Regularized model as learning rate <0.1
final_params = {'n_estimators': 1000, 'max_leaf_nodes': 4, 'max_depth': None,
'random_state': 2,
                'min_samples_split': 5}

model1 = ensemble.GradientBoostingClassifier(**final_params)
model1.fit(X_train, Y_train)

#Original model
original_params = {'n_estimators': 1000, 'max_leaf_nodes': 4, 'max_depth':
None, 'random_state': 2,
                'min_samples_split': 5, 'learning_rate':1}

model2 = ensemble.GradientBoostingClassifier(**original_params)
model2.fit(X_train, Y_train)
```

The default model is inclusive of regularization which means that learning rate is < 1. In this case, it was 0.1 as shown below.

```
GradientBoostingClassifier(criterion='friedman_mse', init=None,
              learning_rate=0.1, loss='deviance', max_depth=None,
              max_features=None, max_leaf_nodes=4,
              min_impurity_split=1e-07, min_samples_leaf=1,
              min_samples_split=5, min_weight_fraction_leaf=0.0,
              n_estimators=1000, presort='auto', random_state=2,
              subsample=1.0, verbose=0, warm_start=False)
```

Without regularization:

```
>>> >>> ... >>> >>> >>> GradientBoostingClassifier(criterion='friedman_mse', in
it=None,
              learning_rate=1, loss='deviance', max_depth=None,
              max_features=None, max_leaf_nodes=4,
              min_impurity_split=1e-07, min_samples_leaf=1,
              min_samples_split=5, min_weight_fraction_leaf=0.0,
              n_estimators=1000, presort='auto', random_state=2,
              subsample=1.0, verbose=0, warm_start=False)
```

Now the performance is compared:

```
#Scoring
model1.score(X_train, Y_train) #0.95977
model2.score(X_train, Y_train) #0.99876

#Generate class probabilities
probs1 = model1.predict_proba(X_test)
probs2 = model2.predict_proba(X_test)

from sklearn.metrics import roc_auc_score

roc_auc_score(Y_test, probs1[:, 1]) #0.89345
roc_auc_score(Y_test, probs2[:, 1]) #0.85054
```

Plotting:

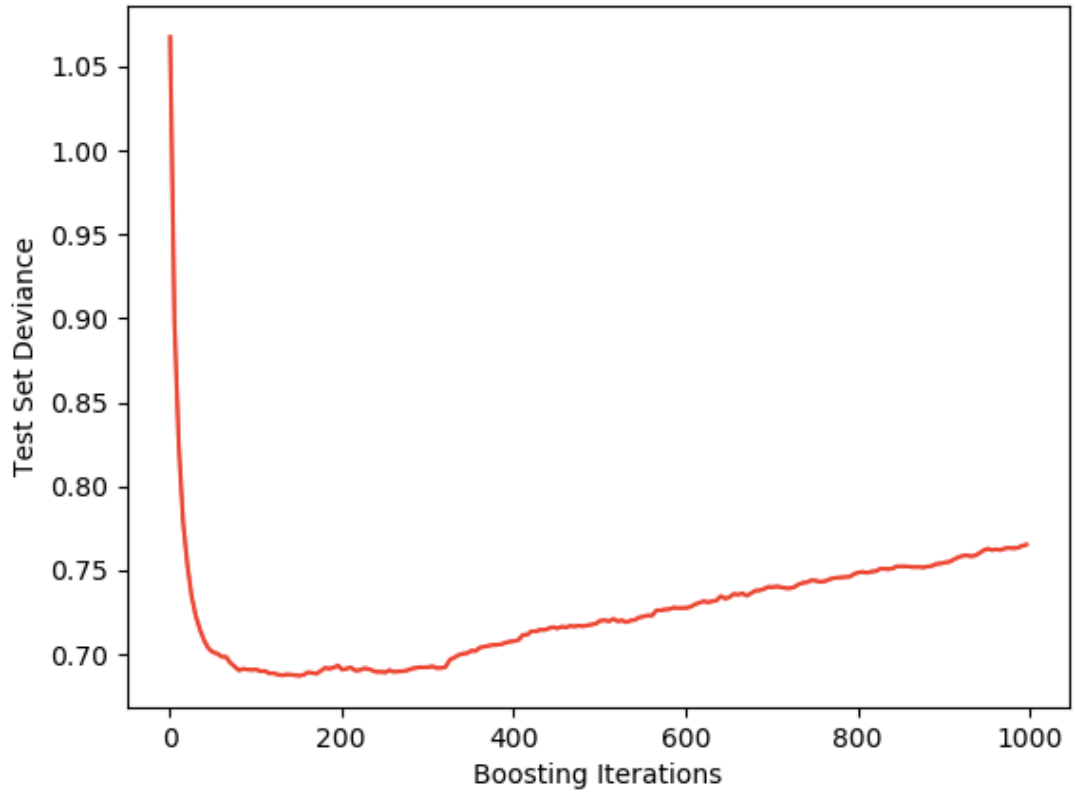
```
# compute test set deviance
test_deviance = np.zeros((final_params['n_estimators'],), dtype=np.float64)
for i, y_pred in enumerate(model1.staged_decision_function(X_test)):
    # clf.loss_ assumes that Y_test[i] in {0, 1}
    test_deviance[i] = model1.loss_(Y_test, y_pred)

#Plot
plt.plot((np.arange(test_deviance.shape[0]) + 1)[::5], test_deviance[::5],
        '-', color='red')

plt.legend(loc='upper left')
plt.xlabel('Boosting Iterations')
```

```
plt.ylabel('Test Set Deviance')
plt.show()
```

Plot:



Conclusion:

AUC values differ. AUC for final model is better than original model without regularization.

FULL CODE:

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
"""
Created on Sat Apr 22 11:44:15 2017
```

```
@author: IshantNayer
"""

from sklearn.model_selection import train_test_split
import pandas as pd
from sklearn import ensemble
import numpy as np
import matplotlib.pyplot as plt

train_data = pd.read_csv('PizzaTrain.csv')

print(train_data.head())

train_data.describe()
train_data.dtypes
print(train_data.head())

train_data.shape #(4040, 32)

train_data1 = train_data.select_dtypes(exclude=['object'])
train_data1.dtypes

print(train_data1.describe())

X = train_data1.drop('requester_received_pizza', axis = 1)
Y = train_data1['requester_received_pizza']

X_train, X_test, Y_train, Y_test = train_test_split(X, Y,
test_size = 0.20, random_state = None)

print(X_train.shape)
print(Y_train.shape)

#Regularized model as learning rate <0.1
final_params = {'n_estimators': 1000, 'max_leaf_nodes': 4, 'max_depth': None,
'random_state': 2,
                'min_samples_split': 5}
```

```
model1 = ensemble.GradientBoostingClassifier(**final_params)
model1.fit(X_train, Y_train)

#Original model
original_params = {'n_estimators': 1000, 'max_leaf_nodes': 4, 'max_depth':
None, 'random_state': 2,
                'min_samples_split': 5, 'learning_rate':1}

model2 = ensemble.GradientBoostingClassifier(**original_params)
model2.fit(X_train, Y_train)

#Scoring
model1.score(X_train, Y_train) #0.95977
model2.score(X_train, Y_train) #0.99876

#Generate class probabilities
probs1 = model1.predict_proba(X_test)
probs2 = model2.predict_proba(X_test)

from sklearn.metrics import roc_auc_score

roc_auc_score(Y_test, probs1[:, 1]) #0.89345
roc_auc_score(Y_test, probs2[:, 1]) #0.85054

# compute test set deviance
test_deviance = np.zeros((final_params['n_estimators'],), dtype=np.float64)
for i, y_pred in enumerate(model1.staged_decision_function(X_test)):
      # clf.loss_ assumes that Y_test[i] in {0, 1}
      test_deviance[i] = model1.loss_(Y_test, y_pred)

#Plot
plt.plot((np.arange(test_deviance.shape[0]) + 1)[::5], test_deviance[::5],
         '-', color='red')

plt.legend(loc='upper left')
plt.xlabel('Boosting Iterations')
```

```
plt.ylabel('Test Set Deviance')
plt.show()
```