Python: Getting Started with Data Science

This short tutorial will not only guide you through some basic data analysis methods but it will also show you how to implement some of the more sophisticated techniques available today. We will look into traffic accident data from the National Highway Traffic Safety Administration and try to predict fatal accidents using state-of-the-art statistical learning techniques. If you are interested, download the code at the bottom and follow along as we work through a real world data set. This post is in Python while a companion post (http://www.datarobot.com/blog/r-getting-started-with-data-science) covers the same techniques in R.

(http://www.datarobot.com/wp-content/uploads/2014/03/python-logo-master-v3-TM-flattened.png)

First things first

For those of you who are not familiar with Python and some of its most popular libraries for data science, please follow along with this blogpost (http://www.datarobot.com/blog/getting-up-and-running-with-python), which will get you set up with an environment similar to the one we will be using. There are instructions for Mac, Linux, and Windows environments, so hopefully we have all the bases covered.

IPython is awesome, as you will come to find out.

Get some data

Being able to play with data requires having the data itself, so let's take care of that right now. The National Highway Traffic Safety Administration (NHTSA) has some really cool data that they make public. The following code snippet will take care of downloading the data to a new directory, and extracting the files from that zipfile. The zip is 14.9 MB so it might take some time to run – it is worth the wait! This is really cool data.

In [1]:

```
import zipfile
import urllib2
import os

source_url = 'ftp://ftp.nhtsa.dot.gov/GES/GES12/GES12_Flatfile.zip'
zip_name = 'GES12_Flatfile.zip'
cwd = os.getcwd()
dir_path  = os.path.join(cwd, 'GES2012')
zip_path = os.path.join(dir_path, zip_name)

# We'll make a directory for you to play around with,
# then when you're done playing you can just delete the directory
if not os.path.exists(dir_path):
    os.makedirs(dir_path)

# Download the file from GES website if you haven't already
if not os.path.exists(zip_path):
    response = urllib2.urlopen(source_url)
    with open(zip_path, 'wb') as fh:
        x = response.read()
        fh.write(x)

# Extract all the files from that zipfile
with zipfile.ZipFile(os.path.join(dir_path, zip_name), 'r') as z:
    z.extractall(dir_path)
```

In [2]:

```
#See what we just unzipped
os.listdir(dir_path)
```

Out[2]:

```
['VIOLATN.TXT',
 'DRIMPAIR.TXT',
 'VEHICLE.TXT',
 'PERSON.TXT',
 'VSOE.TXT',
 'PARKWORK.TXT',
 '2012GESFlatFileTXT.sas',
 'GES12_Flatfile.zip',
 'NMPRIOR.TXT',
 'VEVENT.TXT',
 'DISTRACT.TXT',
 'CEVENT.TXT',
 'DAMAGE.TXT',
 'ACCIDENT.TXT',
 'SAFETYEQ.TXT',
 'VISION.TXT',
 'NMIMPAIR.TXT',
 'FACTOR.TXT',
 'MANEUVER.TXT',
 'NMCRASH.TXT']
```

Load the data into Python

With our data downloaded and readily accessible, we can start to play around and see what we can learn from the data. Many of the columns have an encoding that you will need to read the manual (ftp://ftp.nhtsa.dot.gov/GES/GES12/GES%20Analytical%20Users%20Manual%201988-2012_FINAL-2013-10-31.pdf) in order to understand, so it might be useful to download that PDF so you can easily refer to it. We will be looking at `PERSON.TXT`, which contains information at the level of the individuals involved in the accidents.

In [3]:

```
import pandas as pd
import numpy as np
import sklearn

cwd = os.getcwd()
dir_path  = os.path.join(cwd, 'GES2012')
input_file_path = os.path.join(dir_path, 'PERSON.TXT')

input_data = pd.read_csv(input_file_path, delimiter='\t')
```

In [4]:

```
sorted(input_data.columns)
```

Out[4]:

```
['AGE',
 'AGE_IM',
 'AIR_BAG',
 'ALC_RES',
 'ALC_STATUS',
 'ATST_TYP',
 'BODY_TYP',
 'CASENUM',
 'DRINKING',
 'DRUGRES1',
 'DRUGRES2',
 'DRUGRES3',
 'DRUGS',
 'DRUGTST1',
 'DRUGTST2',
 'DRUGTST3',
 'DSTATUS',
 'EJECTION',
 'EJECT_IM',
 'EMER_USE',
 'FIRE_EXP',
 'HARM_EV',
 'HOSPITAL',
 'HOUR',
 'IMPACT1',
 'INJSEV_IM',
 'INJ_SEV',
 'LOCATION',
 'MAKE',
 'MAN_COLL',
 'MINUTE',
 'MOD_YEAR',
 'MONTH',
 'PERALCH_IM',
 'PER_NO',
 'PER_TYP',
 'PJ',
 'PSU',
 'PSUSTRAT',
 'P_SF1',
 'P_SF2',
 'P_SF3',
 'REGION',
 'REST_MIS',
 'REST_USE',
 'ROLLOVER',
 'SCH_BUS',
 'SEAT_IM',
 'SEAT_POS',
 'SEX',
 'SEX_IM',
 'SPEC_USE',
 'STRATUM',
 'STR_VEH',
 'TOW_VEH',
 'VEH_NO',
 'VE_FORMS',
 'WEIGHT']
```

Clean up the data

One prediction task you might find interesting is predicting whether or not a crash was fatal. The column `INJSEV_IM` contains imputed values for the severity of the injury, but there is still one value that might complicate analysis – level 6 indicates that the person died prior to the crash.

In [5]:

```
input_data.INJSEV_IM.value_counts()
```

Out[5]:

```
0    100840
2     20758
1     19380
3      9738
5      1179
4      1178
6         4
dtype: int64
```

Fortunately, there are only four of those cases within the dataset, so it is not unreasonable to ignore them during our analysis. However, we will find that a few of the columns in the data have missing values:

In [6]:

```
# Drop those odd cases
input_data = input_data[input_data.INJSEV_IM != 6]

for column_name in input_data.columns:
    n_nans = input_data[column_name].isnull().sum()
    if n_nans > 0:
        print column_name, n_nans
```

```
MAKE 5162
BODY_TYP 5162
MOD_YEAR 5162
TOW_VEH 5162
SPEC_USE 5162
EMER_USE 5162
ROLLOVER 5162
IMPACT1 5162
FIRE_EXP 5162
```

For this analysis, we will just drop these rows (they are all the same rows) – but you certainly don't have to do that. In fact, maybe there is a systematic data entry error that is causing them to be interpreted incorrectly. Regardless of the way you cleanup this data, we will most assuredly want to drop the column `INJ_SEV` , as it is the non-imputed version of `INJSEV_IM` and is a pretty severe data leak – there are others as well.

In [7]:

```
print input_data.shape
data = input_data[~input_data.MAKE.isnull()]
discarded = data.pop('INJ_SEV')
target = data.pop('INJSEV_IM')
print data.shape
```

```
(153073, 58)
(147911, 56)
```

One more preprocessing step we'll do is to transform the response. If you flip to the manual it shows that category `4` is a fatal injury – so we will encode our target as such.

In [8]:

```
target = (target == 4).astype('float')
```

Now we model!

We have predictors, we have a target, now it is time to build a model. We will be using ordinary least squares, a Ridge Regression (http://en.wikipedia.org/wiki/Ridge_regression) and Lasso Regression (http://en.wikipedia.org/wiki/Lasso_%28statistics%29#Lasso_method), both being forms of regularized Linear Regression, Gradient Boosting Machine (http://en.wikipedia.org/wiki/Gradient_boosting) (GBM) and a CART (http://en.wikipedia.org/wiki/Predictive_analytics#Classification_and_regression_trees) to have some variety in modeling methods. These are just some representatives from the scikit-learn (http://scikit-learn.org/stable/modules/classes.html) library, which gives access to quite a few machine learning techniques.

Don't be alarmed if these cell blocks take quite a bit of time to run – the data is of non-negligible size. Additionally, some of the models perform a search over several parameters to find a best fit, and the gradient boosting classifier is building many trees in order to produce its ensembled decisions. There is a lot of computation going on under the hood, so get up and take a break if you need.

In [9]:

```
from sklearn.linear_model import LinearRegression
from sklearn.metrics import roc_auc_score
from sklearn.grid_search import GridSearchCV

# Train on half of the data while reserving the other half for
# model comparisons
xtrain, xtest, ytrain, ytest = sklearn.cross_validation.train_test_split(
    data.values, target.values, train_size=0.5)

linreg = LinearRegression()
linreg.fit(xtrain, ytrain)

lr_preds = linreg.predict(xtest)
lr_perf = roc_auc_score(ytest, lr_preds)
print 'OLS: Area under the ROC curve = {}'.format(lr_perf)
```

```
OLS: Area under the ROC curve = 0.935139729907
```

In [10]:

```
from sklearn.linear_model import Ridge

ridge = GridSearchCV(Ridge(),
                     {'alpha': np.logspace(-10, 10, 10)})
ridge.fit(xtrain, ytrain)
ridge_preds = ridge.predict(xtest)
ridge_performance = roc_auc_score(ytest, ridge_preds)
print 'Ridge: Area under the ROC curve = {}'.format(ridge_performance)
```

```
Ridge: Area under the ROC curve = 0.935221465912
```

In [11]:

```
from sklearn.linear_model import Lasso
from sklearn.metrics import roc_auc_score
from sklearn.grid_search import GridSearchCV

lasso = GridSearchCV(Lasso(),
                     {'alpha': np.logspace(-10, -8, 5)})
lasso.fit(xtrain, ytrain)
lasso_preds = lasso.predict(xtest)
lasso_performance = roc_auc_score(ytest, lasso_preds)
print 'Lasso: Area under the ROC curve = {}'.format(lasso_performance)
```

```
Lasso: Area under the ROC curve = 0.939851198289
```

In [12]:

```
from sklearn.ensemble import GradientBoostingClassifier
from sklearn.metrics import roc_auc_score
from sklearn.grid_search import GridSearchCV

gbm = GradientBoostingClassifier(n_estimators=500)

gbm.fit(xtrain, ytrain)
gbm_preds = gbm.predict_proba(xtest)[:, 1]
gbm_performance = roc_auc_score(ytest, gbm_preds)

print 'GBM: Area under the ROC curve = {}'.format(gbm_performance)
```

```
GBM: Area under the ROC curve = 0.970431372668
```

In [13]:

```
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import roc_auc_score

tree = GridSearchCV(DecisionTreeClassifier(),
                    {'max_depth': np.arange(3, 10)})

tree.fit(xtrain, ytrain)
tree_preds = tree.predict_proba(xtest)[:, 1]
tree_performance = roc_auc_score(ytest, tree_preds)

print 'DecisionTree: Area under the ROC curve = {}'.format(tree_performance)
```

```
DecisionTree: Area under the ROC curve = 0.913468225877
```

As one final morsel for you to chew on, it would be good to understand which variables the GBM model thinks are most useful for classification. Spoiler alert: data leaks ahead.

In [14]:

```
importances = pd.Series(gbm.feature_importances_, index=data.columns)
print importances.order(ascending=False)[:10]
```

```
STRATUM       0.261623
REST_USE      0.123142
EJECT_IM      0.071433
AGE_IM        0.056080
ALC_RES       0.050908
AIR_BAG       0.048473
HARM_EV       0.044688
HOUR          0.043379
DRUGS         0.042505
PERALCH_IM    0.035529
dtype: float64
```

Now what should I do?

We have a great blogpost (http://www.datarobot.com/blog/regularized-linear-regression-with-scikit-learn) that goes into more detail about regularized linear regression, if that is what you are interested in. It would also be good to look into all the models that are offered by scikit-learn (http://scikit-learn.org) – you might find some you have never heard of! Beyond that, here are a few challenges that you can undertake to help you hone your data science skills.

Data Prep

If it wasn't obvious in the blog post, the column `STRATUM` is a data leak (it encodes the severity of the crash). Which other columns contain data leaks? Can you come up with a rigorous method to generate candidates for deletion without having to read the entire GES manual?

And while we are considering data preparation, consider the column `REGION` . Any regression model will consider the West region to be 4 times more `REGION` -y than the Northeast – that just doesn't make sense. Which columns could benefit from a one-hot encoding (http://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.OneHotEncoder.html)?

Which is the best model?

How good of a model can you build for predicting fatalities from car crashes? First you will need to settle on a metric of "good" – and be prepared to reason why it is a good metric. How bad is it to be wrong? How good is it to be right?

In order to avoid overfitting (http://en.wikipedia.org/wiki/Overfitting) you will want to separate some of the data and hold it in reserve for when you evaluate your models – some of these models are expressive enough to memorize all the data!

Which is the best story?

Of course, data science is more than just gathering data and building models – it's about telling a story backed-up by the data. Do crashes with alcohol involved tend to lead to more serious injuries? When it is late at night, are there more convertibles involved in crashes than other types of vehicles (this one involves looking at a different dataset within the GES data)? Which is the safest seat in a car? And how sure can you be that your findings are statistically relevant?

Good luck coming up with a great story!

Download Notebook (https://s3.amazonaws.com/datarobotblog/notebooks/Getting Started with Data Science.ipynb) View on NBViewer (http://nbviewer.ipython.org/urls/s3.amazonaws.com/datarobotblog/notebooks/Getting Started with Data Science.ipynb)

This post was written by Dallin Akagi and Mark Steadman. Please post any feedback, comments, or questions below or send us an email at <firstname>@datarobot.com.

2 Responses to "Python: Getting Started with Data Science"

Jason

First off, thanks for such an informative article! I just had some noob question. How would you go about building a confusion matrix for say, the ridge regression? I tried using the confusion_matrix function but quickly realized that the ridge_preds variable was a continuous array.

Reply (http://www.datarobot.com/blog/python-getting-started-with-data-science/?replytocom=72375#respond)

Ted Kwartler (http://www.datarobot.com)

The confusion matrix approach to evaluating a model doesn't directly apply to regression problems due to the continuous nature of the target variable. However, You could in principle bin both the actual values and the predicted values into deciles (or any other approach, really), and visualize them in that manner. Keep in mind that your approach for binning can affect the ability to interpret the confusion matrix, especially if either or both of the distributions is not a normal distribution.

Reply (http://www.datarobot.com/blog/python-getting-started-with-data-science/?replytocom=73814#respond)

Leave a Reply

Your email address will not be published. Required fields are marked *

Name *

Email *

Website

Comment

Post Comment

Connect with Us.

(https://twitter.com/DataRobot)

(http://www.linkedin.com/company/datarobot)

(https://plus.google.com/108404356130400194703)