Lab_1_Loading_and_Understanding_Your_Data

April 24, 2019


```
In [0]: # Licensed under the Apache License, Version 2.0 (the "License");
        # you may not use this file except in compliance with the License.
        # You may obtain a copy of the License at
        #
        # https://www.apache.org/licenses/LICENSE-2.0
        #
        # Unless required by applicable law or agreed to in writing, software
        # distributed under the License is distributed on an "AS IS" BASIS,
        # WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
        # See the License for the specific language governing permissions and
        # limitations under the License.
```

1 Lab 1: Loading and Understanding Your Data

Learning Objectives: * Learn the basics of reading data with Pandas * Learn the basics of data cleaning and handling missing data using Pandas * Learning how to visualize data with a scatter plot * Use Numpy to generate the line minimizing squared loss * Explore visually the difference in the model when replacing missing items by 0s versus the mean value for that feature.

1.1 Data Set

This lab will use a data set from 1985 Ward's Automotive Yearbook that is part of the UCI Machine Learning Repository under Automobile Data Set. You can find a description of the data at `https://archive.ics.uci.edu/ml/datasets/automobile`.

1.2 Imports

In this first cell, we'll import some libraries, including Pandas that will be used later to read and load the data. Run this cell to execute the code.

```
In [0]: import math

        from IPython import display
        from matplotlib import cm
        from matplotlib import gridspec
```

```
from matplotlib import pyplot as plt
import numpy as np
import pandas as pd
from mpl_toolkits.mplot3d import Axes3D
from sklearn import metrics
import tensorflow as tf
from tensorflow.contrib.learn.python.learn import learn_io, estimator

# This line incrasing the amount of logging when there is an error.  You can
# remove it if you want less logging
tf.logging.set_verbosity(tf.logging.ERROR)

print "Done with the imports."
```

1.3 Pandas -- Python Data Analysis Library

We are using a package called Pandas for reading in our data, exploring our data and doing some basic processing. First we set up some options to control how items are displayed and the maximum number of rows to show when displaying a table. Feel free to change this setup to whatever you'd like.

As illustrated below, in colab you can define code cells that do not generate any output. In fact, the first cell would also have been that way except for the `print` statement at the end added just to help illustrate this aspect of colab. If at any point you are not sure if a cell is successfully running, you can add a print statement, but that should not be necessary since you can visually see when the cell is done running when the arrow is showing again. Also note that when you select the next cell the number showing the execution order of the cell appears.

```
In [0]: # Set the output display to have one digit for decimal places, for display
        # readability only and limit it to printing 15 rows.
        pd.options.display.float_format = '{:.2f}'.format
        pd.options.display.max_rows = 15
```

1.3.1 Loading the Data Set with Pandas

The car data set we will be using in this lab is provided as a comma separated file without a header row. In order, for each column to have a meaningful header name we must provide it. We get the information about the columns from Automobile Data Set

```
In [0]: # Provide the names for the columns since the CSV file with the data does
        # not have a header row.
        cols = ['symboling', 'losses', 'make', 'fuel-type', 'aspiration', 'num-doors',
                'body-style', 'drive-wheels', 'engine-location', 'wheel-base',
                'length', 'width', 'height', 'weight', 'engine-type', 'num-cylinders',
                'engine-size', 'fuel-system', 'bore', 'stroke', 'compression-ratio',
                'horsepower', 'peak-rpm', 'city-mpg', 'highway-mpg', 'price']


        # Load in the data from a CSV file that is comma seperated.
```

```
        car_data = pd.read_csv('https://storage.googleapis.com/ml_universities/cars_dataset/ca
                               sep=',', names=cols, header=None, encoding='latin-1')

        print "Data set loaded."
```

1.3.2 Examine the Data

It's a good idea to get to know your data a little bit before you work with it. Let's look at the header row and the first 10 rows of data.

```
In [0]: car_data[1:10]
```

1.3.3 Look at Some Basic Statistics About the Data

We'll print out a quick summary of a few useful statistics on each column. This will include things like mean, standard deviation, max, min, and various quantiles.

```
In [0]: car_data.describe()
```

1.3.4 Handling Missing Data Entries

Why are some columns such as the price and losses not showing in the summary column? If we look at the data more carefuly, we'll see that a "?" was used in this data set to indicate that a value is unknown.

Pandas provices a method to_numeric that converts a column to be numeric. There are three options about how to handle errors (entries that are not numeric) of which 'coerce' is what we want to use in this situation, since it converts those entries to pandas representation `NaN` for "not a number".

Notice now when you use `describe` to see the statistics for the entries that are a number, the count indicates how many entries had a number (not `NaN`) for that column.

```
In [0]: car_data['price'] = pd.to_numeric(car_data['price'], errors='coerce')
        car_data['horsepower'] = pd.to_numeric(car_data['horsepower'], errors='coerce')
        car_data['peak-rpm'] = pd.to_numeric(car_data['peak-rpm'], errors='coerce')
        car_data['city-mpg'] = pd.to_numeric(car_data['city-mpg'], errors='coerce')
        car_data['highway-mpg'] = pd.to_numeric(car_data['highway-mpg'], errors='coerce')
        car_data['losses'] = pd.to_numeric(car_data['losses'], errors='coerce')
        car_data.describe()
```

1.3.5 Replacing NAN by zero

When training a linear model using features that is numerical, we **cannot have `NaN` (doing so would cause overflow when training)**. Here we replace `NaN` (which corresponding to where we had missing entries) by 0.

```
In [0]: # Replace nan by the mean storing the solution in the same table (`inplace')
        car_data.fillna(0, inplace=True)
        car_data.describe()
```

1.3.6 Using a Scatter Plot to Visualize the Data

We will begin by trying to predict the price using the horsepower. Because we just have a single feature we can visualize the raw data using a scatter plot.

```
In [0]: INPUT_FEATURE = "horsepower"
        LABEL = "price"

        plt.ylabel(LABEL)
        plt.xlabel(INPUT_FEATURE)
        plt.scatter(car_data[INPUT_FEATURE], car_data[LABEL], c='black')
        plt.show()
```

1.3.7 Using numpy polyfit to find the line that minimizes RMSE

For the task of finding a line that minimizes the squared error with respect to a set of points, using SGD is not the most efficient method but it will be useful in that it can be applied to much more complex problems. As a tool to help see what the optimal solution looks like we wil use `polyfit` to compute the optimal solution and then add that to our plot.

```
In [0]: x = car_data[INPUT_FEATURE]
        y = car_data[LABEL]
        opt = np.polyfit(x, y, 1)
        y_pred = opt[0] * x + opt[1]
        opt_rmse = math.sqrt(metrics.mean_squared_error(y_pred, y))
        slope = opt[0]
        bias = opt[1]
        print "Optimal RMSE =", opt_rmse, "for solution", opt
```

1.3.8 Showing A Linear Regression Model in a Scatter Plot

To help provide intuition to what is being learned by a linear regression model and as a way to visualzize the quality, we provide a method to create the scatter plot of a single input feature with respect to the label feature. In addition, a set of lines (from different linear regression models) as provided as a list of slopes, biases and model_names for the legend can be provided.

```
In [0]: def make_scatter_plot(dataframe, input_feature, target,
                               slopes=[], biases=[], model_names=[]):
          """ Creates a scatter plot of input_feature vs target along with the models.

          Args:
            dataframe: the dataframe to visualize
            input_feature: the input feature to be used for the x-axis
            target: the target to be used for the y-axis
            slopes: list of model weight (slope)
            bias: list of model bias (same size as slopes)
            model_names: list of model_names to use for legend (same size as slopes)
          """
          # Define some colors to use that go from blue towards red
```

```
    colors = [cm.coolwarm(x) for x in np.linspace(0, 1, len(slopes))]

    # Generate the Scatter plot
    x = dataframe[input_feature]
    y = dataframe[target]
    plt.ylabel(target)
    plt.xlabel(input_feature)
    plt.scatter(x, y, color='black', label="")

    # Add the lines corresponding to the provided models
    for i in range (0, len(slopes)):
      y_0 = slopes[i] * x.min() + biases[i]
      y_1 = slopes[i] * x.max() + biases[i]
      plt.plot([x.min(), x.max()], [y_0, y_1],
               label=model_names[i], color=colors[i])
    if (len(model_names) > 0):
      plt.legend(bbox_to_anchor=(1.05, 1), loc=2, borderaxespad=0.)
```

```
In [0]: make_scatter_plot(car_data,INPUT_FEATURE, LABEL,
                          [slope], [bias], ["initial model"])
```

1.4 Task 1: Create a scatter plot for new features (1 Point)

Create a scatter plot with price as the input feature and losses as the label where the line optimizing the squared loss is shown. We've gotten you started. You just need to copy the appropriate lines from above. **Save the slope and bias from this line to be used in a later task.**

```
In [0]: INPUT_FEATURE = "price"
        LABEL = "losses"

        # Fill in the rest of this block.
```

1.5 Task 2: Explain what you see (1 point)

Explain why we are seeing so many points along the line `y=0`.

PUT YOUR ANSWER HERE

1.6 Task 3: Options to handle missing data (3 Points)

In this task you will explore alternate ways to handle missing data. When training a linear model using features that are numerical, we **cannot have `NaN` (doing so would cause overflow when training)**. One option is to just discard any rows with any missing entries but often this would not leave enough data. Here we explore ways to handle the missing data without just discarding it.

Note that when you get a column of a dataframe (e.g. car_data["price']), you get a `Series`. Read http://pandas.pydata.org/pandas-docs/version/0.18.1/api.html#computations-descriptive-stats and think about if you see any statistics for a column that

might make a better choice than 0 for filling in the missing entries.What do you think would work best?

Modify the code to use the function you selected to replace missings instead of just using 0 and both look at the scatter plot and the line minimizing RMSE both with the missings replaced by 0 and the missing entries replaced by the option you pick. Feel free to show multiple options as a tool to help you explain your choice of which you think is best.. What option do you think is best and why?

```
In [0]: # Load in the data from a CSV file that is comma seperated.
        car_data_v2 = pd.read_csv('https://storage.googleapis.com/ml_universities/cars_dataset/
                                  sep=',', names=cols, header=None, encoding='latin-1')
        car_data_v2['price'] = pd.to_numeric(car_data_v2['price'], errors='coerce')
        car_data_v2['losses'] = pd.to_numeric(car_data_v2['losses'], errors='coerce')

In [0]: # Fill in what you want to do with the nan here

In [0]: # Create a scatter plot with the model from above (when NA replaced by 0) and at least
```

Put your answer here to what option you think is best and why here