Learning Insurance Claims

Hardik Shah
Northeastern University
shah.hard@husky.neu.edu

Abstract

The project is to create a machine learning model that trains to predict the cost of given insurance claims given a certain claims details.

1. Introduction

When you've been devastated by a serious car accident, your focus is on the things that matter the most: family, friends, and other loved ones. Pushing paper with your insurance agent is the last place you want your time or mental energy spent. Hence to give customer a better customer service in their time of need, quoting the claims amount as quick as possible is pivotal. In this project I try to employ machine models on given claims dataset to try to make an initial accurate claims amount prediction.

1.1. Dataset

Data is given in csv format. It has all the personal information scrubbed and to make it simple the feature names are changed to generic cat# for categorical features and cont# for continuous features, and the last column is the label. Following is a sample record.

cat1	cat3	cat116	cont1	cont2	cont14	loss
A	A	LB	0.7263	0.245921	0.714843	2213.18
A	A	DP	0.330514	0.737068	0.304496	1283.6
A	A	GK	0.261841	0.358319	0.774425	3005.09
B	A	DJ	0.321594	0.555782	0.602642	939.85

Each record has 130 features, of which 116 are categorical, cat1 to cat116, and 14 are continuous, cont1 to cont14. "loss" here gives the claims amount. I have sampled around 5000 records from training and 1000 records from testing.

1.2. Models

This clearly looks like a regression problem. Hence to baseline the result. I use a simple Linear Regression approach with a ridge regularization. I also compare the Results with a two layer feed forward neural network regressor.

2. Proposed Approach

Used the Google's Tensorflow to implement machine learning algorithms. It is a low-level machine learning package where it is possible to create models with high flexibility. All the training was done in batches.

2.1. Data Pre-Processing

As described earlier, the data has 116 categorical features, to handle those I chose to do one-hot encoding to all the categorical feature. e.g. cat1 has A, B categories. Hence it was changed to (1, 0) for A and (0, 1) for B. The loss, i.e., the label of the data was very large as compared to the features given. Hence, to keep the learning parameters small, I normalized the labels by Z-score normalizing. [refer script: file_proc.py]. Data can be found in the data folder of the zip file. _orig is the original data and _prcsd is the processed data. After processing the data each records ends up having 4327 features and the last column as the label.

2.2. Ridge Regression

Used Ridge regression algorithm to train a model against the given data. Following is the code snippets for the model.

```
W = tf.Variable(tf.zeros([c.NUM_FEATURES, 1], dtype=tf.float32), name="W")
b = tf.Variable(tf.zeros([1], dtype=tf.float32), name="b")
logits = tf.matmul(features_placeholder, W) + b

#cost for simple linear regression
#cost_op = tf.reduce_mean(tf.pow(logits - labels_placeholder, 2))

ridge_param = tf.constant(1.)
ridge_loss = tf.reduce_mean(tf.square(W))
#cost for ridge regression
cost_op = tf.expand_dims(tf.add(tf.reduce_mean(tf.square(labels_placeholder -
logits))
                                , tf.mul(ridge_param, ridge_loss)),0)

train_op = tf.train.GradientDescentOptimizer(learning_rate).minimize(cost_op)
```

I use the inbuilt gradient descent to train the model and use the Ridge regression cost to run the gradient descent on. Each batch is trained max_steps time or till the cost for that batch has reduced to certain tolerance.

```
while (loss_value > c.TOLERENCE and step < c.MAX_STEPS):
   feed_dict = fill_feed_dict(data_set,
data_set.shape[0],features_placeholder, labels_placeholder)
   _, loss_value = sess.run([train_op, cost_op], feed_dict=feed_dict)
   if step % (c.MAX_STEPS / 2) == 0:
        # Print status to stdout.
        print('Step %d: loss = %.2f' % (step, loss_value))
   step += 1
```

2.3 Feed Forward Neural Network

Used a 4-layer Feed Forward Neural Network, with 2 hidden layers with rectified linear (Relu) activation function. Like in Linear Regression, I used batches to train the model.

Following is the code snippets to create the network architecture.

```
# Hidden 1
  with tf.name_scope('hidden1'):
    weights = tf.Variable(
        tf.truncated_normal([c.NUM_FEATURES, hidden1_units],
                            stddev=1.0 / math.sqrt(float(c.NUM_FEATURES))),
        name='weights')
    biases = tf.Variable(tf.zeros([hidden1_units]), name='biases')
    hidden1 = tf.nn.relu(tf.matmul(data, weights) + biases)

# Hidden 2
  with tf.name_scope('hidden2'):
    weights = tf.Variable(
        tf.truncated_normal([hidden1_units, hidden2_units],
                            stddev=1.0 / math.sqrt(float(hidden1_units))),
        name='weights')
    biases = tf.Variable(tf.zeros([hidden2_units]), name='biases')
    hidden2 = tf.nn.relu(tf.matmul(hidden1, weights) + biases)

# Linear
  with tf.name_scope('linear-regression'):
    weights = tf.Variable(
        tf.truncated_normal([hidden2_units, c.NUM_OUTPUT],
                            stddev=1.0 / math.sqrt(float(hidden2_units))),
        name='weights')
    biases = tf.Variable(tf.zeros([c.NUM_OUTPUT]), name='biases')
    logits = tf.matmul(hidden2, weights) + biases
```

The layer weights are initialized with N(0,1/sqrt(features)) and biases are initialized with zero. The training uses the backpropagation to train the model, using the tensorflow inbuilt gradient descent function, as shown in the following snippet

```
# Create the gradient descent optimizer with the given learning rate.
optimizer = tf.train.GradientDescentOptimizer(learning_rate)
# Create a variable to track the global step.
global_step = tf.Variable(0, name='global_step', trainable=False)
# Use the optimizer to apply the gradients that minimize the loss
# (and also increment the global step counter) as a single training step.
train_op = optimizer.minimize(cost, global_step=global_step)
```

Again, all the batches are trained max_steps times or till the cost comes down to tolerance or less.

2.4. Accuracy

I calculate the accuracy by finding the absolute difference between the predicted label and the target label. Following snippet demonstrates the calculation of the accuracy in the testing data.

```
y_pred = lr.run_test_wihthout_label(model, test_data[cur_batch: cur_batch +
batch_size, :])
y_tar = np.reshape(test_data[cur_batch: cur_batch + batch_size:, -1],
(batch_size, c.NUM_OUTPUT))

error_avg = np.mean(np.abs(y_pred - y_tar))
```

3. Experiments

3.1. Linear Regression

Batch Size	Learning Rate	Average Testing Error
1000	0.001	0.562
1000	0.01	0.462
1000	0.02	NAN
1000	0.1	NAN

It is observed at the learning rate 0.001 is too small while 0.02 is large enough to not converge.

3.2. Feed Forward Neural Network (2 Hidden Layers)

Following are the experiment results with the neural network model.

Hidden1	Hidden2	Batch Size	Learning Rate	Avg. test Error
100	20	100	0.01	0.450
100	20	500	0.01	0.446
100	50	500	0.01	0.449
120	20	500	0.01	0.436
120	60	100	0.01	0.492
120	20	100	0.01	0.495
120	1	500	0.01	0.526

Here the observation is when the number of neurons in the second layer is set to 1, it gives an effect of a 1 hidden layer neural network and the average test error is considerably high.

3.3. Comparison

The linear regression model does a decent job in doing a preliminary modelling, while the neural network actually brings in a better modelling of the data and decreases the error considerably.

4. Final Inference

Neural Networks are very flexible in terms of types of problem they can model while Linear Regression gives a very good first look at the data and simpler to implement. Neural Networks are good at learning

irregularities, non-linearity etc. in the data as compared to Linear regression algorithm.

5. Future Work

Use Dimension Reduction (PCA, Autoencoder) to reduce the dimension of the data. Make the architecture of the neural network more flexible, i.e., to take user defined number of layers and activation function in each layer. Make the algorithm run faster and memory efficient. Make use of Tensorboard for inbuilt graphics and data flow imaging.

References

[1] https://www.tensorflow.org/.
[2] https://www.coursera.org/learn/machine-learning.