Topic - Ml Questions

Difficulty Med : Classify Trucks

Create and return a convolutional neural network (CNN) to classify images produced by a traffic-camera feed based on whether or not a truck is in the images. Note that: You should use TensorFlow 2 Keras APIs to create a CNN that expects images of dimensions 224x224 with 3 channels (Red, Green, and Blue). The CNN should consist of 8 ordered layers: A convolutional layer with 16 filters, each with a kernel of dimensions 3x3 and a Rectified Linear Unit (ReLU) activation function. A max pooling layer with dimensions 2x2. A convolutional layer with 32 filters, each with a kernel of dimensions 3x3 and a ReLU activation function. Another max pooling layer with dimensions 2x2. A convolutional layer with 64 ,filters each with a kernel of dimensions 3x3 and a ReLU activation function. A layer to flatten the values of the previous layer. A dense layer (fully connected) with 20 neurons and a ReLU activation function. Finally, a dense layer with a single neuron and sigmoid activation. This layer will output the prediction. Each convolutional and pooling layer should use valid padding, each layer with a ReLU activation should use He normal initialization, and the layer with a sigmoid activation should use Glorot normal initialization. The CNN should use Adam as the optimizer with a learning rate of .01. As well, binary cross-entropy should be used as the loss function, since the label of each frame is 1 when there's a truck in the image and 0 when there isn't a truck in the image.

The following two images are example images that will be used to train your model:


The first image would be labeled as 1, since a truck is in the image. Conversely, the second image would be labeled as 0, since there isn't a truck in the image. The training will be handled for you. You're only required to return the model with the above specifications. Be sure to include binary accuracy as the metric to monitor during training.

Hint:

First, we need to create a Sequential model proivided by TensorFlow's high-level API, Keras. We can provide the Sequential model with a list of layers that we want to include in the CNN. The first layer will be a convolutional layer, which is availble in Keras as Conv2D. We need to speciy the Rectified Linear Unit (ReLu) activation function which can be done by assigning the activation parameter the value of 'relu'. As well, we also have to specify an input_shape parameter, which will allow the Conv2D layer to know what to expect in terms of input. For this scenario, we want to specify a tuple (224, 224, 3), which contains the image dimensions followed by the three channels, RGB—one for each color channel in the images. We want to use He normal initialization for the first layer. To do that we need to set the kernel_initializer parameter as 'he_normal'. Next, we want a max pooling layer, specified by MaxPooling2D in Keras. When we get to the flatten layer, we need to use tf.keras.layers.Flatten().When we get to the first dense layer, we need to use Dense in Keras. Finally, the output layer will be a dense layer as well. It'll have a single neuron, a sigmoid activation, and use 'glorot_normal' initialization. Once the model is defined, we need to compile the model with an optimizer, a loss, and a metric that we want to monitor during training. To do that, we can use the compile function on the model. It takes in an optimizer (in our case, Adam), a BinaryCrossentropy() as its loss, and ['binary_accuracy'] as its metrics.

Difficulty Med : Predict Cancellations

Use the LogisticRegression library within PySpark to predict subscription cancellations based on user features. Return the prediction dataframe produced by the LogisticRegression model. This dataframe will contain the following columns: rawPrediction, which has the log-odds for each example, probability, which has the probabilities for each example, and prediction, which contains either a 0 or 1, depending on the probability and threshold hyperparameter.

The prediction dataframe should be trained and predicted on the following dataframe:

```
+--------+----------------------+--------------------+--------------------+--------------------+
|
user_id|month_interaction_count|week_interaction_count|day_interaction_count|cancelled_within_week|
+--------+----------------------+--------------------+--------------------+--------------------+
|66860ae6|                    41|                   9|                   0|                   1|
|249803f8|                    25|                   9|                   2|                   0|
|32ed74cc|                    21|                   2|                   1|                   1|
|7ed76e6a|                    22|                   5|                   2|                   0|
|46c81f43|                    32|                   8|                   2|                   0|
|cf0f185e|                    26|                   4|                   0|                   1|
|568275b3|                    29|                   5|                   1|                   1|
|86a060ec|                    33|                   7|                   1|                   1|
|c0c07290|                    35|                  10|                   0|                   0|
|709dc1da|                    36|                  11|                   1|                   0|
+--------+----------------------+--------------------+--------------------+--------------------+
```

The dataframe above, accessible as user_interaction_df in the code, contains features for each user_id, including month_interaction_count, week_interaction_count, and day_interaction_count. For each user_id, the binary label cancelled_within_week is 1 if the user cancelled their subscription within a week of the last recorded day_interaction_count and 0 otherwise. The model hyperparameters include 10 iterations, a decision threshold of 0.6, L1 regularization, and a regularization parameter of 0.1.

Sample Output Dataframe

```
+--------+-------------+-----------+----------+
|user_id |rawPrediction |probability |prediction|
+--------+-------------+-----------+----------+
|010b4076|         [x,x]|       [x,x]|       0.0|
|31c73683|         [x,x]|       [x,x]|       0.0|
|8173164f|         [x,x]|       [x,x]|       0.0|
|f77ad2d3|         [x,x]|       [x,x]|       0.0|
|25050522|         [x,x]|       [x,x]|       0.0|
|bfb27c75|         [x,x]|       [x,x]|       0.0|
|09663ea6|         [x,x]|       [x,x]|       1.0|
|ca7aacf2|         [x,x]|       [x,x]|       0.0|
|63f84e80|         [x,x]|       [x,x]|       0.0|
|cbb81ed7|         [x,x]|       [x,x]|       0.0|
+--------+-------------+-----------+----------+
```

Hint:

First, we need to create a "features" column that groups the user features together. We can do this by using a VectorAssembler. Then, we need to create a "label" column that contains all the labels. Next, we'll create the model with the correct hyperparameters. LogisticRegression uses elasticNetParam to indicate L1, L2, or a combination of regularization strategies. Use an elasticNetParam of 1 to specify L1 regularization. We need to assign the decision threshold to 0.6 with the threshold argument. We then need to specify maxIter as 10 to limit the number of iterations of optimization for the model. Lastly, our regularization parameter regParam will be assigned to 0.1. We then call fit on the LogisticRegression model. Finally, we call transform on the same dataframe we trained with to get the predictions. This will evaluate the model's performance on the training examples. Make sure to ultimately select the appropriate rows specified in the question prompt.

Difficulty Med : Neuron

Create a single neuron, to be used in a neural network. You'll have access to a list Neuron.examples of 100 examples, where each example is a dictionary with two keys: "features" and "label". The value of "features" is a list of 3 features, which have been min-max scaled for you, and the value of "label" is 0 or 1.

Below is an example portion of the Neuron.examples:

```
[

  {"features": [0.7737498370415932, 0.893981580520576,
0.7776116731845149], "label": 0},

  {"features": [0.8356527294792708, 0.7535044575176968,
0.7940884252881397], "label": 0},

  # ...

  # More examples.

  {"features": [0.25835793676162827, 0.2166447564607853,
0.5066866046843734], "label": 1},

  {"features": [0.34848185391755987, 0.15010261370695727,
0.3466287718524547], "label": 1},

  # ...

  # More examples.

]
```

Note that:

You shouldn't use regularization.

You should use mini-batch gradient descent.

You should use the sigmoid activation function.

The neuron should include a weight for the bias and one weight for each input feature.

You should use 0.01 for the learning rate, 10 examples for the mini-batch size, 100 epochs, and the log loss as the loss function.

The predict function should return the probability that the input should have the label of 1—not the corresponding 0 or 1 label.

Your output value will automatically be rounded to the fourth decimal.

Sample Input

```
features = [0.79, 0.89, 0.777]
```

Sample Output

```
0.1636 // The probability that the sample input should have a label of 1.
```

Hint:

First, we need to extract mini-batches from the examples. This means retrieving a portion of total examples, which will be used to complete one iteration of gradient descent. Now that we have a mini-batch of examples, we need to get the prediction for each example in the mini-batch. This means performing a forward pass through the neuron. A forward pass for a single example involves multiplying each feature in the example by each weight. Don't forget to include a bias in the features, to go with the corresponding weight. The bias will just be a 1 appended to the features, and the weight for the bias will be treated as just another weight. Then, the results of these multiplications are summed up and finally passed through a sigmoid to obtain the prediction. Sigmoids take the equation: 1/(1 + e^-(x)). In Python, you'll use the math.exp() function for e. After we have the prediction for each example in the mini-batch, we need to find the gradient of the loss function with respect to each weight in the neuron. The gradient with respect to any weight i is the difference between the prediction and the true label of an example, multiplied by feature_i. Since we're using a mini-batch, we sum each gradient produced by each example. Finally, we divide each gradient by the mini-batch size. Now that we have 4 gradients (3 for the inputs and 1 for the bias), we need to update the weights as follows: w_i(t+1) = w_i(t) - lr*gradient_i, where lr is the learning rate. Now that the weights are updated, we continue through the remaining mini-batches until a single epoch is completed; we then repeat the entire process for 99 more epochs. Now that the neuron is trained, we can get a prediction by performing a forward pass with any input features.

Difficulty Med: Regression Tree

Create a regression tree to predict the barrels per day (the bpd) produced by a particular drilling site, given some porosity, gamma, sonic, and density values. You'll have access to a list of training examples in the RegressionTree's root node, specifically in RegressionTree.root.examples. Each example is a dictionary with feature keys mapping to their respective values and with a bpd key mapping to the example's label.

Below is an example portion of the RegressionTree.root.examples:

```
[
  {
    "porosity": 0.24230826038076003,
    "gamma": 1.5600463819136288,
    "sonic": 2568.8231147730116,
    "density": -0.353639698833012,
    "bpd": 164.7544334411493, # The label for this example.
  },
  {
    "porosity": 0.4821959432320581,
    "gamma": 1.4953123610344377,
    "sonic": 2768.866560695128,
    "density": 1.1231264377284371,
    "bpd": 157.33821193599536, # The label for this example.
  },
  {
    "porosity": 0.058672948847231135,
    "gamma": 1.5384704880812365,
```

```
    "sonic": 3236.794545516582,
    "density": 1.2698807135982118,
    "bpd": 159.49129568528647, # The label for this example.
  },
  # ...
  # More examples.
]
```

Note that:

You should use the mean squared error (MSE) as the splitting criteria. You shouldn't evaluate "bpd" as a feature to split on, since it's what you're trying to predict. You shouldn't use any hyperparameters like max depth. There's no need to handle missing data or to scale the features. Recursive and iterative implementations are both fine. You shouldn't use any libraries that implement regression trees for you, such as scikit-learn. Your output values will automatically be rounded to the fourth decimal.

Sample Input

```
features = {
  "porosity": 0.70,
  "gamma": 1.57,
  "sonic": 3666.90,
  "density": 2.52
}
```

Sample Output

```
143.0698 # The regression tree prediction for the input's bpd.
```

Hint:

We first need to train the regression tree. The root node is populated with examples, which we need to split up and put into left and right TreeNodes. This process can recursively continue to the leaf nodes, at which point there

should only be one example left in a node. Finding the best split of an example involves evaluating all of its feature and split point values to see which split point produces the smallest MSE. Each split point value to evaluate is the average of two adjacent feature values when the feature values are sorted. After finding the split point that minimizes the MSE, an example in a node is sent to the left child node if the relevant feature value is less than or equal to the split point value. Otherwise, the example is sent to the right child node. Finding a prediction means traversing down the regression tree while comparing the relevant feature values of the input to each node's split point value. Upon reaching a leaf node in the regression tree, the prediction is just the average of the labels in that leaf node.