[redacted]

We predict flight delays in the year 2022, it's a classification problem.

there are 5 stages for delay: delay of maximum 15 minutes, delay more than 15 minutes and less than 30 minutes, delay more than 30 minutes and less than 45 minutes,   delay more than 45 minutes and less than 60 minutes and delay more than 60 minutes.

We train the model to predict the delay label.

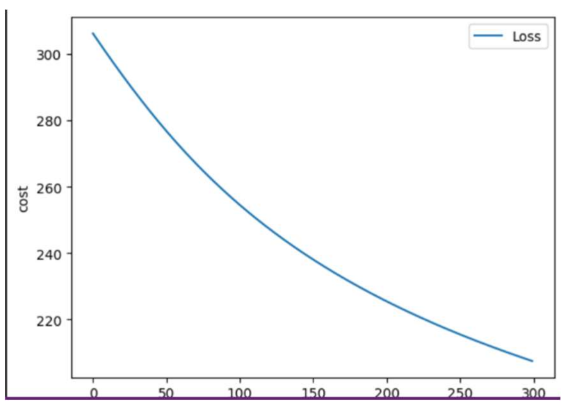
Our train results:

Our best result was when we used a classifier with 3 hidden layers .

First step

In the first step we used softmax without hidden layers to train the model.   With the softmax training we reached ***66 percent*** accuracy.

Below is the graph of the loss function that we got:



Second step:

In the second step we trained the model with several hidden layers. The first time with 3 hidden layers and the second time with 5 hidden layers.

- The result with 3 hidden layers:
  With 3 hidden layers we reached an accuracy of ***92 percent***, this is the highest accuracy we have reached. Here too we used Softmax.
- The result with 5 hidden layers:
  With 5 hidden layers we reached an accuracy of ***77 percent***. We see that the results are less good than 3 hidden layers. Here too we used Softmax.

The complete code of the **first step** (without hidden layers):

```python
smote = SMOTE()
X_data, Y_data = smote.fit_resample(X.to_numpy(), Y.to_numpy())
```

```python
scale = StandardScaler()
X_data = scale.fit_transform(X_data)
X_data
```

```
array([[ 1.09930271, -1.04546464,  0.12290052, ..., -1.05469336,
        -1.04088408, -0.41481666],
       [-1.52458816, -0.45201939, -1.3624324 , ...,  0.01079175,
        -0.06108706, -0.41481666],
       [-0.21264272,  1.32831636,  0.12290052, ...,  2.79188847,
         2.87616002, -1.21085198],
       ...,
       [-0.07572768,  0.14142586,  1.62325335, ...,  1.80964887,
         1.26288992,  0.03510215],
       [-1.22627892,  0.89735225,  1.12893618, ..., -0.04058314,
         0.0994909 ,  0.74135622],
       [-1.27713184, -0.06893349,  0.06882286, ..., -0.64139027,
        -0.6837012 , -0.13264493]])
```

```python
X_train, X_test, Y_train, Y_test = train_test_split(X_data, Y_data, test_size=0.2, random_state=1)
X_train, X_val, Y_train, Y_val = train_test_split(X_train, Y_train, test_size=0.25, random_state=1)
Y_train = Y_train.reshape((Y_train.shape[0],5))
Y_val = Y_val.reshape((Y_val.shape[0],5))
Y_test = Y_test.reshape((Y_test.shape[0],5))
```

```python
features = X_train.shape[1]
x = tf.placeholder(tf.float32, [None, features])
y_ = tf.placeholder(tf.float32, [None, 5])
W = tf.Variable(tf.zeros([features, 5]))
b = tf.Variable(tf.zeros([5]))
```

```python
pred = tf.nn.softmax(tf.add(tf.matmul(x, W), b))
loss = -tf.reduce_mean(y_ * tf.log(pred))
```

```python
update = tf.train.GradientDescentOptimizer(0.01).minimize(loss)
```

```python
correct_pred = tf.equal(tf.argmax(pred, 1), tf.argmax(y_, 1))
```

```python
accuracy = tf.reduce_mean(tf.cast(correct_pred, tf.float32))
```

```python
init = tf.global_variables_initializer()
```

```python
epoches = 300
batch_size = 300
errors = []
sess = tf.Session()
sess.run(tf.global_variables_initializer())
for i in range(epoches):
    for start, end in zip(range(0, len(X_train), batch_size),range(batch_size, len(X_train), batch_size)):
        sess.run(update, feed_dict = {x: X_train[start:end],y_: Y_train[start:end]})
    cost = sess.run(tf.nn.l2_loss(pred - Y_val),feed_dict = {x:X_val})
    errors.append(cost)
    if i%100 == 0:
        print("epoch %d, cost = %g" % (i, cost))
```

```
epoch 0, cost = 306.148
epoch 100, cost = 254.508
epoch 200, cost = 225.481
```

The result of the first step:

```
[ ]
    print("Accuracy: ", round(sess.run(accuracy, feed_dict={x:X_test, y_:Y_test}),2))
                                                                                  Python
...  Accuracy:  0.66
```

The complete code of the **second step** with **3** hidden layers:

3 Hidden layers:

```python
# Placeholder variables for the input features and labels
X = tf.placeholder(tf.float32, shape=[None, 17])
y = tf.placeholder(tf.float32, shape=[None, 5])

# Define the hidden layers
hidden1 = tf.layers.dense(X, 64, activation=tf.nn.relu)
hidden2 = tf.layers.dense(hidden1, 32, activation=tf.nn.relu)
hidden3 = tf.layers.dense(hidden2, 16, activation=tf.nn.relu)

# Define the output layer using the softmax function
logits = tf.layers.dense(hidden3, 5, activation=tf.nn.softmax)

# Define the loss function and optimizer
loss = tf.reduce_mean(tf.nn.softmax_cross_entropy_with_logits_v2(logits=logits, labels=y))
optimizer = tf.train.AdamOptimizer().minimize(loss)
```

```python
with tf.Session() as sess:
    # Initialize the variables
    sess.run(tf.global_variables_initializer())

    # Train the model on the training set
    for i in range(300):
        sess.run(optimizer, feed_dict={X: X_train, y: Y_train})

    # Evaluate the model on the test set
    accuracy = sess.run(loss, feed_dict={X: X_test, y: Y_test})
    print("Accuracy: ",round(accuracy, 2))
                                                                                  Python
```

```
Accuracy:  0.92
```

ⓘ Connecting to kernel: base (Python 3.8.3): Activating Python Environm...

The complete code of the **second step** with **5** hidden layers:

5 hidden layers:

```python
# Define the number of classes
num_classes = 5
# Define the number of features
num_features = 17
# Define the number of hidden layers
num_hidden_layers = 5
# Define the size of each hidden layer
hidden_layer_size = 32
# Placeholder for the input data
X = tf.placeholder(tf.float32, [None, num_features])
# Placeholder for the true labels
Y_true = tf.placeholder(tf.int32, [None, num_classes])
# Define the weights and biases of each hidden layer
weights = [
    tf.Variable(tf.random_normal([num_features, hidden_layer_size])),
    tf.Variable(tf.random_normal([hidden_layer_size, hidden_layer_size])),
    tf.Variable(tf.random_normal([hidden_layer_size, hidden_layer_size])),
    tf.Variable(tf.random_normal([hidden_layer_size, hidden_layer_size])),
    tf.Variable(tf.random_normal([hidden_layer_size, num_classes])),
]
biases = [
    tf.Variable(tf.random_normal([hidden_layer_size])),
    tf.Variable(tf.random_normal([hidden_layer_size])),
    tf.Variable(tf.random_normal([hidden_layer_size])),
    tf.Variable(tf.random_normal([hidden_layer_size])),
    tf.Variable(tf.random_normal([num_classes])),
]

# Define the neural network layers
hidden_layer = tf.add(tf.matmul(X, weights[0]), biases[0])
hidden_layer = tf.nn.relu(hidden_layer)

hidden_layer = tf.add(tf.matmul(hidden_layer, weights[1]), biases[1])
hidden_layer = tf.nn.relu(hidden_layer)

hidden_layer = tf.add(tf.matmul(hidden_layer, weights[2]), biases[2])
hidden_layer = tf.nn.relu(hidden_layer)

hidden_layer = tf.add(tf.matmul(hidden_layer, weights[3]), biases[3])
hidden_layer = tf.nn.relu(hidden_layer)

output_layer = tf.add(tf.matmul(hidden_layer, weights[4]), biases[4])

# Define the softmax function to generate the final prediction probabilities
predictions = tf.nn.softmax(output_layer)

# Define the cross-entropy loss function
loss = tf.reduce_mean(tf.nn.softmax_cross_entropy_with_logits_v2(logits=output_layer, labels=Y_true))

# Define the optimization method (i.e. gradient descent)
optimizer = tf.train.GradientDescentOptimizer(learning_rate=0.001).minimize(loss)
```

```python
correct_predictions = tf.equal(tf.argmax(predictions, 1), tf.argmax(Y_true, 1))
accuracy = tf.reduce_mean(tf.cast(correct_predictions, tf.float32))
```

Python

+ Code + Markdown

```python
sess = tf.Session()
sess.run(tf.global_variables_initializer())
```

Python

```python
# Calculate the accuracy on the training set
for i in range(1000):
    sess.run(accuracy, feed_dict={X: X_train, Y_true: Y_train})
```

Python

```python
test_accuracy = sess.run(accuracy, feed_dict={X: X_test, Y_true: Y_test})
print("Accuracy: ",round(test_accuracy,2))
```

Python

```
Accuracy:  0.77
```