Open in Colab

T81-558: Applications of Deep Neural Networks

Module 4: Training for Tabular Data

- Instructor: Jeff Heaton, McKelvey School of Engineering, Washington University in St. Louis
- For more information visit the class website.

Module 4 Material

- Part 4.1: Encoding a Feature Vector for Keras Deep Learning [Video] [Notebook]
- **Part 4.2: Keras Multiclass Classification for Deep Neural Networks with ROC and AUC** [Video] [Notebook]
- Part 4.3: Keras Regression for Deep Neural Networks with RMSE [Video] [Notebook]
- Part 4.4: Backpropagation, Nesterov Momentum, and ADAM Neural Network Training [Video] [Notebook]
- Part 4.5: Neural Network RMSE and Log Loss Error Calculation from Scratch [Video] [Notebook]

Google CoLab Instructions

The following code ensures that Google CoLab is running the correct version of TensorFlow.

In [1]:
```
try:
    %tensorflow_version 2.x
    COLAB = True
    print("Note: using Google CoLab")
except:
    print("Note: not using Google CoLab")
    COLAB = False
```

```
Note: using Google CoLab
```

Part 4.2: Multiclass Classification with ROC and AUC

The output of modern neural networks can be of many different forms. However, classically, neural network output has typically been one of the following:

- **Binary Classification** - Classification between two possibilities (positive and negative). Common in medical testing, does the person has the disease (positive) or not (negative).
- **Classification** - Classification between more than 2. The iris dataset (3-way classification).
- **Regression** - Numeric prediction. How many MPG does a car get? (covered in next video)

We will look at some visualizations for all three in this section.

It is important to evaluate the false positives and negatives in the results produced by a neural network. We will now look at assessing error for both classification and regression neural networks.

Binary Classification and ROC Charts

Binary classification occurs when a neural network must choose between two options: true/false, yes/no, correct/incorrect, or buy/sell. To see how to use binary classification, we will consider a classification system for a credit card company. This system will either "issue a credit card" or "decline a credit card." This classification system must decide how to respond to a new potential customer.

When you have only two classes that you can consider, the objective function's score is the number of false-positive predictions versus the number of false negatives. False negatives and false positives are both types of errors, and it is essential to understand the difference. For the previous example, issuing a credit card would be positive. A false positive occurs when a model decides to issue a credit card to someone who will not make payments as agreed. A false negative happens when a model denies a credit card to someone who would have made payments as agreed.

Because only two options exist, we can choose the mistake that is the more serious type of error, a false positive or a false negative. For most banks issuing credit cards, a false positive is worse than a false negative. Declining a potentially good credit card holder is better than accepting a credit card holder who would cause the bank to undertake expensive collection activities.

Consider the following program that uses the wcbreast_wdbc dataset to classify if a breast tumor is cancerous (malignant) or not (benign).

```
In [2]: import pandas as pd

        df = pd.read_csv(
            "https://data.heatonresearch.com/data/t81-558/wcbreast_wdbc.csv",
            na_values=['NA','?'])

        pd.set_option('display.max_columns', 5)
        pd.set_option('display.max_rows', 5)

        display(df)
```

	id	diagnosis	...	worst_symmetry	worst_fractal_dimension
0	842302	M	...	0.4601	0.11890
1	842517	M	...	0.2750	0.08902
...	...	...	...	...	...
567	927241	M	...	0.4087	0.12400
568	92751	B	...	0.2871	0.07039

569 rows × 32 columns

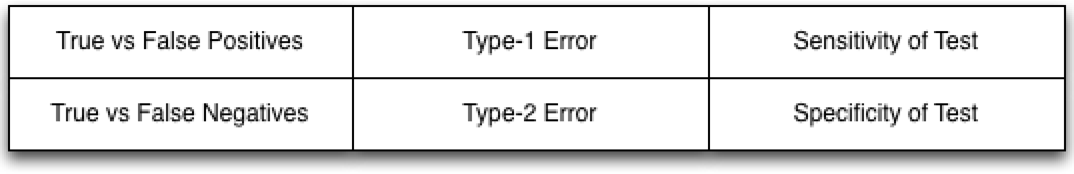
ROC curves can be a bit confusing. However, they are prevalent in analytics. It is essential to know how to read them. Even their name is confusing. Do not worry about their name; the receiver operating characteristic curve (ROC) comes from electrical engineering (EE).

Binary classification is common in medical testing. Often you want to diagnose if someone has a disease. This diagnosis can lead to two types of errors, known as false positives and false negatives:

- **False Positive** - Your test (neural network) indicated that the patient had the disease; however, the patient did not.
- **False Negative** - Your test (neural network) indicated that the patient did not have the disease; however, the patient did have the disease.
- **True Positive** - Your test (neural network) correctly identified that the patient had the disease.
- **True Negative** - Your test (neural network) correctly identified that the patient did not have the disease.

Figure 4.ETYP shows you these types of errors.

Figure 4.ETYP: Type of Error

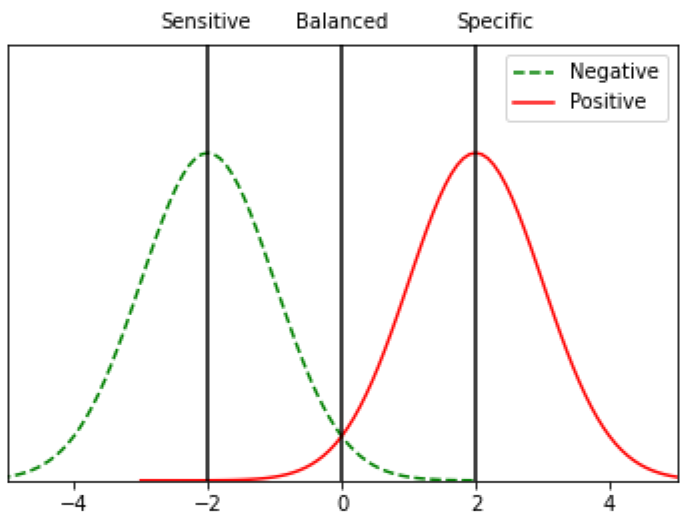


True vs False Positives	Type-1 Error	Sensitivity of Test
True vs False Negatives	Type-2 Error	Specificity of Test

Neural networks classify in terms of the probability of it being positive. However, at what possibility do you give a positive result? Is the cutoff 50%? 90%? Where you set, this cutoff is called the threshold. Anything above the cutoff is positive; anything below is negative. Setting this cutoff allows the model to be more sensitive or specific:

More info on Sensitivity vs. Specificity: Khan Academy

In [3]:
```
%matplotlib inline
import matplotlib.pyplot as plt
import numpy as np
import scipy.stats as stats
import math

mu1 = -2
mu2 = 2
variance = 1
sigma = math.sqrt(variance)
x1 = np.linspace(mu1 - 5*sigma, mu1 + 4*sigma, 100)
x2 = np.linspace(mu2 - 5*sigma, mu2 + 4*sigma, 100)
plt.plot(x1, stats.norm.pdf(x1, mu1, sigma)/1,color="green",
         linestyle='dashed')
plt.plot(x2, stats.norm.pdf(x2, mu2, sigma)/1,color="red")
plt.axvline(x=-2,color="black")
plt.axvline(x=0,color="black")
plt.axvline(x=+2,color="black")
plt.text(-2.7,0.55,"Sensitive")
plt.text(-0.7,0.55,"Balanced")
plt.text(1.7,0.55,"Specific")
plt.ylim([0,0.53])
plt.xlim([-5,5])
plt.legend(['Negative','Positive'])
plt.yticks([])
plt.show()
```



We will now train a neural network for the Wisconsin breast cancer dataset. We begin by preprocessing the data. Because we have all numeric data, we compute a z-score for each column.

In [4]:
```
from scipy.stats import zscore

x_columns = df.columns.drop('diagnosis').drop('id')
for col in x_columns:
    df[col] = zscore(df[col])

# Convert to numpy - Regression
x = df[x_columns].values
y = df['diagnosis'].map({'M':1,"B":0}).values # Binary classification,
                                              # M is 1 and B is 0
```

We can now define two functions. The first function plots a confusion matrix. The second function plots a ROC chart.

In [5]:
```
%matplotlib inline
import matplotlib.pyplot as plt
from sklearn.metrics import roc_curve, auc

# Plot a confusion matrix.
# cm is the confusion matrix, names are the names of the classes.
def plot_confusion_matrix(cm, names, title='Confusion matrix',
                            cmap=plt.cm.Blues):
    plt.imshow(cm, interpolation='nearest', cmap=cmap)
    plt.title(title)
    plt.colorbar()
    tick_marks = np.arange(len(names))
    plt.xticks(tick_marks, names, rotation=45)
    plt.yticks(tick_marks, names)
    plt.tight_layout()
    plt.ylabel('True label')
    plt.xlabel('Predicted label')
```

```
# Plot an ROC. pred - the predictions, y - the expected output.
def plot_roc(pred,y):
    fpr, tpr, _ = roc_curve(y, pred)
    roc_auc = auc(fpr, tpr)

    plt.figure()
    plt.plot(fpr, tpr, label='ROC curve (area = %0.2f)' % roc_auc)
    plt.plot([0, 1], [0, 1], 'k--')
    plt.xlim([0.0, 1.0])
    plt.ylim([0.0, 1.05])
    plt.xlabel('False Positive Rate')
    plt.ylabel('True Positive Rate')
    plt.title('Receiver Operating Characteristic (ROC)')
    plt.legend(loc="lower right")
    plt.show()
```

ROC Chart Example

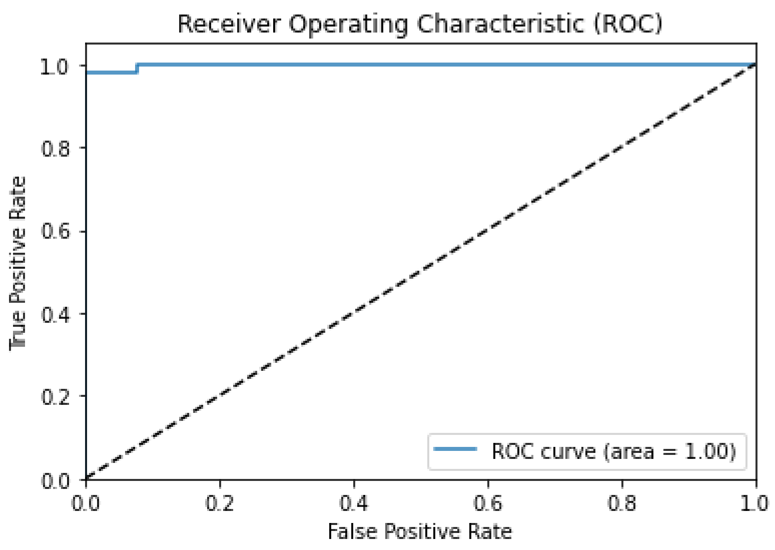
The following code demonstrates how to implement a ROC chart in Python.

In [6]:
```
# Classification neural network
import numpy as np
import tensorflow.keras
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Activation
from tensorflow.keras.callbacks import EarlyStopping
from sklearn.model_selection import train_test_split

# Split into train/test
x_train, x_test, y_train, y_test = train_test_split(
    x, y, test_size=0.25, random_state=42)

model = Sequential()
model.add(Dense(100, input_dim=x.shape[1], activation='relu',
                kernel_initializer='random_normal'))
model.add(Dense(50,activation='relu',kernel_initializer='random_normal'))
model.add(Dense(25,activation='relu',kernel_initializer='random_normal'))
model.add(Dense(1,activation='sigmoid',kernel_initializer='random_normal'))
model.compile(loss='binary_crossentropy',
              optimizer=tensorflow.keras.optimizers.Adam(),
              metrics =['accuracy'])
monitor = EarlyStopping(monitor='val_loss', min_delta=1e-3,
    patience=5, verbose=1, mode='auto', restore_best_weights=True)

model.fit(x_train,y_train,validation_data=(x_test,y_test),
          callbacks=[monitor],verbose=2,epochs=1000)
```

```
Epoch 1/1000
14/14 - 2s - loss: 0.6850 - accuracy: 0.8427 - val_loss: 0.6673 - val_accura
cy: 0.9441 - 2s/epoch - 164ms/step
Epoch 2/1000
14/14 - 0s - loss: 0.6261 - accuracy: 0.9319 - val_loss: 0.5393 - val_accura
cy: 0.9720 - 130ms/epoch - 9ms/step
Epoch 3/1000
14/14 - 0s - loss: 0.4265 - accuracy: 0.9413 - val_loss: 0.2536 - val_accura
cy: 0.9720 - 148ms/epoch - 11ms/step
Epoch 4/1000
14/14 - 0s - loss: 0.2112 - accuracy: 0.9437 - val_loss: 0.1067 - val_accura
cy: 0.9720 - 145ms/epoch - 10ms/step
Epoch 5/1000
14/14 - 0s - loss: 0.1171 - accuracy: 0.9624 - val_loss: 0.0644 - val_accura
cy: 0.9790 - 110ms/epoch - 8ms/step
Epoch 6/1000
14/14 - 0s - loss: 0.0852 - accuracy: 0.9789 - val_loss: 0.0552 - val_accura
cy: 0.9860 - 114ms/epoch - 8ms/step
Epoch 7/1000
14/14 - 0s - loss: 0.0744 - accuracy: 0.9789 - val_loss: 0.0541 - val_accura
cy: 0.9860 - 115ms/epoch - 8ms/step
Epoch 8/1000
14/14 - 0s - loss: 0.0662 - accuracy: 0.9812 - val_loss: 0.0473 - val_accura
cy: 0.9930 - 154ms/epoch - 11ms/step
Epoch 9/1000
14/14 - 0s - loss: 0.0602 - accuracy: 0.9812 - val_loss: 0.0493 - val_accura
cy: 0.9860 - 90ms/epoch - 6ms/step
Epoch 10/1000
14/14 - 0s - loss: 0.0548 - accuracy: 0.9859 - val_loss: 0.0468 - val_accura
cy: 0.9860 - 225ms/epoch - 16ms/step
Epoch 11/1000
14/14 - 0s - loss: 0.0491 - accuracy: 0.9836 - val_loss: 0.0484 - val_accura
cy: 0.9860 - 133ms/epoch - 10ms/step
Epoch 12/1000
14/14 - 0s - loss: 0.0458 - accuracy: 0.9836 - val_loss: 0.0486 - val_accura
cy: 0.9860 - 119ms/epoch - 8ms/step
Epoch 13/1000
Restoring model weights from the end of the best epoch: 8.
14/14 - 0s - loss: 0.0417 - accuracy: 0.9883 - val_loss: 0.0477 - val_accura
cy: 0.9860 - 124ms/epoch - 9ms/step
Epoch 13: early stopping
```

```
Out[6]: <keras.callbacks.History at 0x7f6a8aee46d0>
```

```
In [7]: pred = model.predict(x_test)
        plot_roc(pred,y_test)
```



Multiclass Classification Error Metrics

If you want to predict more than one outcome, you will need more than one output neuron. Because a single neuron can predict two results, a neural network with two output neurons is somewhat rare. If there are three or more outcomes, there will be three or more output neurons. The following sections will examine several metrics for evaluating classification error. We will assess the following classification neural network.

In [8]:
```
import pandas as pd
from scipy.stats import zscore

# Read the data set
df = pd.read_csv(
    "https://data.heatonresearch.com/data/t81-558/jh-simple-dataset.csv",
    na_values=['NA','?'])

# Generate dummies for job
df = pd.concat([df,pd.get_dummies(df['job'],prefix="job")],axis=1)
df.drop('job', axis=1, inplace=True)

# Generate dummies for area
df = pd.concat([df,pd.get_dummies(df['area'],prefix="area")],axis=1)
df.drop('area', axis=1, inplace=True)

# Missing values for income
med = df['income'].median()
df['income'] = df['income'].fillna(med)

# Standardize ranges
df['income'] = zscore(df['income'])
df['aspect'] = zscore(df['aspect'])
df['save_rate'] = zscore(df['save_rate'])
df['age'] = zscore(df['age'])
df['subscriptions'] = zscore(df['subscriptions'])
```

```
# Convert to numpy - Classification
x_columns = df.columns.drop('product').drop('id')
x = df[x_columns].values
dummies = pd.get_dummies(df['product']) # Classification
products = dummies.columns
y = dummies.values
```

In [9]:

```
# Classification neural network
import numpy as np
import tensorflow.keras
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Activation
from tensorflow.keras.callbacks import EarlyStopping
from sklearn.model_selection import train_test_split

# Split into train/test
x_train, x_test, y_train, y_test = train_test_split(
    x, y, test_size=0.25, random_state=42)

model = Sequential()
model.add(Dense(100, input_dim=x.shape[1], activation='relu',
                kernel_initializer='random_normal'))
model.add(Dense(50,activation='relu',kernel_initializer='random_normal'))
model.add(Dense(25,activation='relu',kernel_initializer='random_normal'))
model.add(Dense(y.shape[1],activation='softmax',
                kernel_initializer='random_normal'))
model.compile(loss='categorical_crossentropy',
              optimizer=tensorflow.keras.optimizers.Adam(),
              metrics =['accuracy'])
monitor = EarlyStopping(monitor='val_loss', min_delta=1e-3, patience=5,
                        verbose=1, mode='auto', restore_best_weights=True)
model.fit(x_train,y_train,validation_data=(x_test,y_test),
          callbacks=[monitor],verbose=2,epochs=1000)
```

```
Epoch 1/1000
47/47 - 2s - loss: 1.4456 - accuracy: 0.4753 - val_loss: 1.1348 - val_accura
cy: 0.4980 - 2s/epoch - 45ms/step
Epoch 2/1000
47/47 - 0s - loss: 1.1477 - accuracy: 0.4720 - val_loss: 1.0890 - val_accura
cy: 0.4980 - 249ms/epoch - 5ms/step
Epoch 3/1000
47/47 - 0s - loss: 1.0847 - accuracy: 0.5040 - val_loss: 1.0205 - val_accura
cy: 0.5380 - 482ms/epoch - 10ms/step
Epoch 4/1000
47/47 - 0s - loss: 0.9608 - accuracy: 0.5920 - val_loss: 0.9546 - val_accura
cy: 0.5740 - 309ms/epoch - 7ms/step
Epoch 5/1000
47/47 - 0s - loss: 0.8508 - accuracy: 0.6480 - val_loss: 0.8616 - val_accura
cy: 0.6600 - 290ms/epoch - 6ms/step
Epoch 6/1000
47/47 - 0s - loss: 0.7942 - accuracy: 0.6660 - val_loss: 0.8018 - val_accura
cy: 0.6900 - 298ms/epoch - 6ms/step
Epoch 7/1000
47/47 - 0s - loss: 0.7581 - accuracy: 0.6927 - val_loss: 0.8044 - val_accura
cy: 0.6740 - 271ms/epoch - 6ms/step
Epoch 8/1000
47/47 - 0s - loss: 0.7434 - accuracy: 0.6893 - val_loss: 0.7885 - val_accura
cy: 0.6660 - 246ms/epoch - 5ms/step
Epoch 9/1000
47/47 - 0s - loss: 0.7522 - accuracy: 0.6867 - val_loss: 0.7835 - val_accura
cy: 0.6720 - 281ms/epoch - 6ms/step
Epoch 10/1000
47/47 - 0s - loss: 0.7158 - accuracy: 0.6987 - val_loss: 0.7727 - val_accura
cy: 0.6840 - 327ms/epoch - 7ms/step
Epoch 11/1000
47/47 - 0s - loss: 0.7129 - accuracy: 0.6887 - val_loss: 0.7966 - val_accura
cy: 0.6820 - 231ms/epoch - 5ms/step
Epoch 12/1000
47/47 - 0s - loss: 0.7105 - accuracy: 0.6947 - val_loss: 0.7700 - val_accura
cy: 0.6620 - 239ms/epoch - 5ms/step
Epoch 13/1000
47/47 - 0s - loss: 0.7119 - accuracy: 0.6940 - val_loss: 0.7680 - val_accura
cy: 0.6700 - 254ms/epoch - 5ms/step
Epoch 14/1000
47/47 - 0s - loss: 0.6934 - accuracy: 0.7047 - val_loss: 0.7743 - val_accura
cy: 0.6600 - 289ms/epoch - 6ms/step
Epoch 15/1000
47/47 - 0s - loss: 0.6904 - accuracy: 0.7093 - val_loss: 0.7564 - val_accura
cy: 0.6860 - 266ms/epoch - 6ms/step
Epoch 16/1000
47/47 - 0s - loss: 0.6837 - accuracy: 0.7007 - val_loss: 0.7423 - val_accura
cy: 0.7000 - 297ms/epoch - 6ms/step
Epoch 17/1000
47/47 - 0s - loss: 0.6783 - accuracy: 0.7120 - val_loss: 0.7519 - val_accura
cy: 0.6840 - 258ms/epoch - 5ms/step
Epoch 18/1000
47/47 - 0s - loss: 0.6665 - accuracy: 0.7153 - val_loss: 0.7582 - val_accura
cy: 0.6660 - 259ms/epoch - 6ms/step
Epoch 19/1000
47/47 - 0s - loss: 0.6702 - accuracy: 0.7000 - val_loss: 0.7504 - val_accura
```

```
cy: 0.6880 - 271ms/epoch - 6ms/step
Epoch 20/1000
47/47 - 0s - loss: 0.6624 - accuracy: 0.7147 - val_loss: 0.7527 - val_accura
cy: 0.6800 - 328ms/epoch - 7ms/step
Epoch 21/1000
Restoring model weights from the end of the best epoch: 16.
47/47 - 1s - loss: 0.6558 - accuracy: 0.7160 - val_loss: 0.7653 - val_accura
cy: 0.6720 - 527ms/epoch - 11ms/step
Epoch 21: early stopping
```

Out[9]:
```
<keras.callbacks.History at 0x7f6a8ad5db50>
```

Calculate Classification Accuracy

Accuracy is the number of rows where the neural network correctly predicted the target class. Accuracy is only used for classification, not regression.

$$accuracy = \frac{c}{N}$$

Where c is the number correct and N is the size of the evaluated set (training or validation). Higher accuracy numbers are desired.

As we just saw, by default, Keras will return the percent probability for each class. We can change these prediction probabilities into the actual iris predicted with **argmax**.

In [10]:
```
pred = model.predict(x_test)
pred = np.argmax(pred,axis=1)
# raw probabilities to chosen class (highest probability)
```

Now that we have the actual iris flower predicted, we can calculate the percent accuracy (how many were correctly classified).

In [11]:
```
from sklearn import metrics

y_compare = np.argmax(y_test,axis=1)
score = metrics.accuracy_score(y_compare, pred)
print("Accuracy score: {}".format(score))
```

```
Accuracy score: 0.7
```

Calculate Classification Log Loss

Accuracy is like a final exam with no partial credit. However, neural networks can predict a probability of each of the target classes. Neural networks will give high probabilities to predictions that are more likely. Log loss is an error metric that penalizes confidence in wrong answers. Lower log loss values are desired.

The following code shows the output of predict_proba:

In [12]:
```python
from IPython.display import display

# Don't display numpy in scientific notation
np.set_printoptions(precision=4)
np.set_printoptions(suppress=True)

# Generate predictions
pred = model.predict(x_test)

print("Numpy array of predictions")
display(pred[0:5])

print("As percent probability")
print(pred[0]*100)

score = metrics.log_loss(y_test, pred)
print("Log loss score: {}".format(score))

# raw probabilities to chosen class (highest probability)
pred = np.argmax(pred,axis=1)
```

```
Numpy array of predictions
array([[0.    , 0.1201, 0.7286, 0.1494, 0.0018, 0.    , 0.    ],
       [0.    , 0.6962, 0.3016, 0.0001, 0.0022, 0.    , 0.    ],
       [0.    , 0.7234, 0.2708, 0.0003, 0.0053, 0.0001, 0.    ],
       [0.    , 0.3836, 0.6039, 0.0086, 0.0039, 0.    , 0.    ],
       [0.    , 0.0609, 0.6303, 0.3079, 0.001 , 0.    , 0.    ]],
      dtype=float32)
As percent probability
[ 0.0001 12.0143 72.8578 14.9446  0.1823  0.0009  0.0001]
Log loss score: 0.7423401429280638
```

Log loss is calculated as follows:

$$\text{log loss} = -\frac{1}{N}\sum_{i=1}^{N}\left(y_i \log(\hat{y}_i) + (1 - y_i)\log(1 - \hat{y}_i)\right)$$

You should use this equation only as an objective function for classifications that have two outcomes. The variable y-hat is the neural network's prediction, and the variable y is the known correct answer. In this case, y will always be 0 or 1. The training data have no probabilities. The neural network classifies it either into one class (1) or the other (0).

The variable N represents the number of elements in the training set the number of questions in the test. We divide by N because this process is customary for an average. We also begin the equation with a negative because the log function is always negative over the domain 0 to 1. This negation allows a positive score for the training to minimize.

You will notice two terms are separated by the addition (+). Each contains a log function. Because y will be either 0 or 1, then one of these two terms will cancel out to 0. If y is 0, then the first term will reduce to 0. If y is 1, then the second term will be 0.

If your prediction for the first class of a two-class prediction is y-hat, then your prediction for the second class is 1 minus y-hat. Essentially, if your prediction for class A is 70% (0.7), then your prediction for class B is 30% (0.3). Your score will increase by the log of your prediction for the correct class. If the neural network had predicted 1.0 for class A, and the correct answer was A, your score would increase by log (1), which is 0. For log loss, we seek a low score, so a correct answer results in 0. Some of these log values for a neural network's probability estimate for the correct class:

- -log(1.0) = 0
- -log(0.95) = 0.02
- -log(0.9) = 0.05
- -log(0.8) = 0.1
- -log(0.5) = 0.3
- -log(0.1) = 1
- -log(0.01) = 2
- -log(1.0e-12) = 12
- -log(0.0) = negative infinity

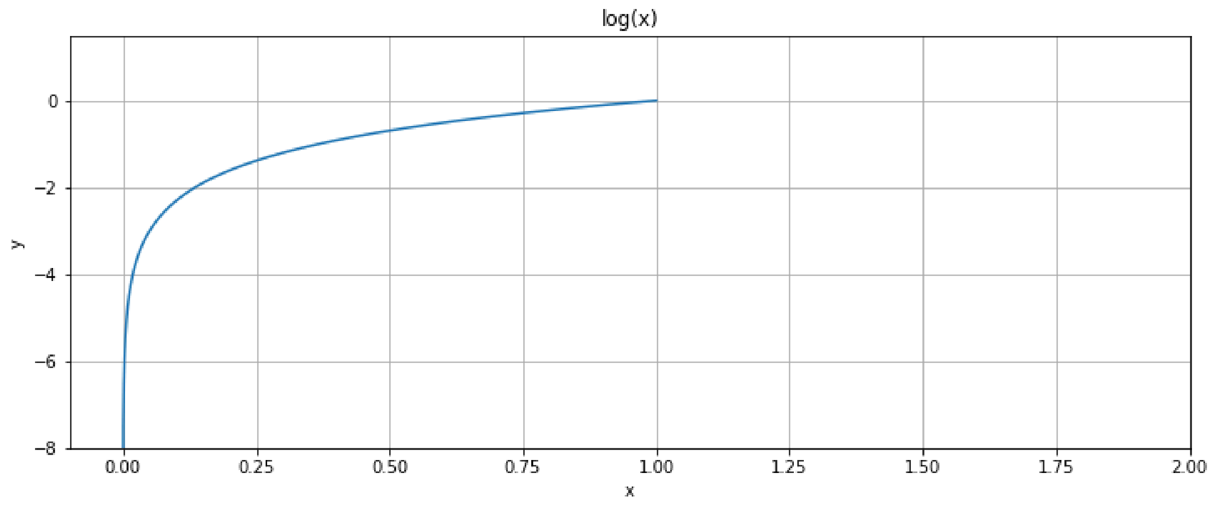
As you can see, giving a low confidence to the correct answer affects the score the most. Because log (0) is negative infinity, we typically impose a minimum value. Of course, the above log values are for a single training set element. We will average the log values for the entire training set.

The log function is useful to penalizing wrong answers. The following code demonstrates the utility of the log function:

In [13]:
```
%matplotlib inline
from matplotlib.pyplot import figure, show
from numpy import arange, sin, pi

#t = arange(1e-5, 5.0, 0.00001)
#t = arange(1.0, 5.0, 0.00001) # computer scientists
t = arange(0.0, 1.0, 0.00001)  # data     scientists

fig = figure(1,figsize=(12, 10))

ax1 = fig.add_subplot(211)
ax1.plot(t, np.log(t))
ax1.grid(True)
ax1.set_ylim((-8, 1.5))
ax1.set_xlim((-0.1, 2))
ax1.set_xlabel('x')
ax1.set_ylabel('y')
ax1.set_title('log(x)')
```

```
show()
```

```
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:12: RuntimeWarn
ing: divide by zero encountered in log
  if sys.path[0] == '':
```

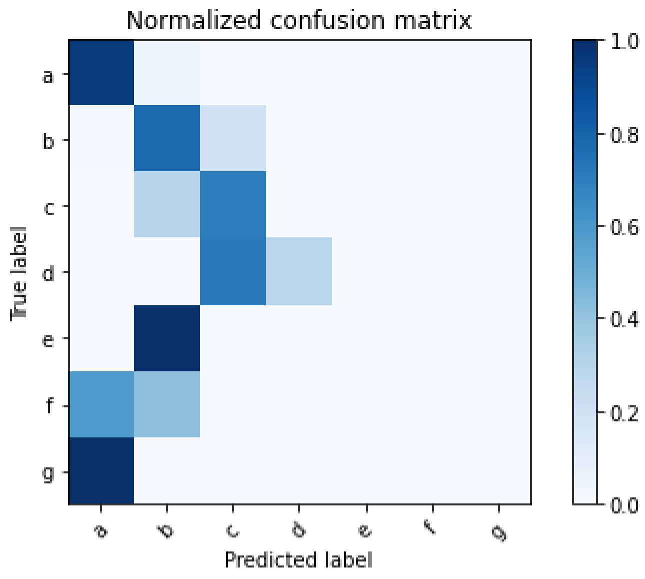


Confusion Matrix

A confusion matrix shows which predicted classes are often confused for the other classes. The vertical axis (y) represents the true labels and the horizontal axis (x) represents the predicted labels. When the true label and predicted label are the same, the highest values occur down the diagonal extending from the upper left to the lower right. The other values, outside the diagonal, represent incorrect predictions. For example, in the confusion matrix below, the value in row 2, column 1 shows how often the predicted value A occurred when it should have been B.

In [14]:
```
import numpy as np
from sklearn import svm, datasets
from sklearn.model_selection import train_test_split
from sklearn.metrics import confusion_matrix

# Compute confusion matrix
cm = confusion_matrix(y_compare, pred)
np.set_printoptions(precision=2)

# Normalize the confusion matrix by row (i.e by the number of samples
# in each class)
cm_normalized = cm.astype('float') / cm.sum(axis=1)[:, np.newaxis]
print('Normalized confusion matrix')
print(cm_normalized)
plt.figure()
plot_confusion_matrix(cm_normalized, products,
        title='Normalized confusion matrix')

plt.show()
```

```
Normalized confusion matrix
[[0.95 0.05 0.   0.   0.   0.   0.  ]
 [0.02 0.78 0.2  0.   0.   0.   0.  ]
 [0.   0.29 0.7  0.01 0.   0.   0.  ]
 [0.   0.   0.71 0.29 0.   0.   0.  ]
 [0.   1.   0.   0.   0.   0.   0.  ]
 [0.59 0.41 0.   0.   0.   0.   0.  ]
 [1.   0.   0.   0.   0.   0.   0.  ]]
```



```
In [14]:
```