3_Evaluate ROC curves

October 10, 2021

1 Exercise: Good and Bad ROC curves

In this exercise, we will make some ROC curves to explain what good and bad ROC curves might look like.

The goal of our models is to identify whether each item detected on the mountain is a hiker (`true`) or a tree (`false`). Let's take a look at the dataset.

[ ]:
```
import numpy
import pandas
import graphings
import sklearn.model_selection

# Load our data from disk
df = pandas.read_csv("hiker_or_tree.csv",delimiter="\t")

# Split into train and test
train, test =  sklearn.model_selection.train_test_split(df, test_size=0.5,␣
 ↪random_state=1)

# Graph our three features
graphings.histogram(test, label_x="height", label_colour="is_hiker", show=True)
graphings.multiple_histogram(test, label_x="motion", label_group="is_hiker",␣
 ↪nbins=12, show=True)
graphings.multiple_histogram(test, label_x="texture", label_group="is_hiker",␣
 ↪nbins=12)
```

```
C:\Users\aduzo\Anaconda3\lib\importlib\_bootstrap.py:219: RuntimeWarning:
numpy.ufunc size changed, may indicate binary incompatibility. Expected 192 from
C header, got 216 from PyObject
  return f(*args, **kwds)
```

We have three visual features - `height`, `motion`, and `texture`. Our goal here is not to optimize a model, but to explore ROC curves, so we'll work with just one at a time.

Before diving into it, take a look at the distributions above. We can see that we should be able to use height to separate hikers from trees quite easily. Motion will be slightly more difficult, presumably because trees blow in the wind, and some hikers are found sitting down. Texture seems much the same for hikers and trees.

1.1 A perfect model

What would a perfect ROC look like? If we had a perfectly designed model, it would calculate "0% chance of hiker" when it saw any tree and "100% of hiker" when it saw any hiker. This means that, so long as the decision threshold was $> 0\%$ and $< 100\%$, it would have perfect performance. Between these thresholds, the true positive rate would always be 1, and the false positive rate would always be 0.

Don't worry about the thresholds of exactly 0 and 1 (100%). At 0 we are forcing a model to return a False value and at 1 we are forcing it to return True.

It's almost impossible to train a model that is so perfect, but for the sake of learning, let's pretend we have done so, predicting the `is_hiker` label based on `height`.

[ ]:
```
import statsmodels.api

# Create a fake model that is perfect at predicting labels
class PerfectModel:
    def predict(self, x):
        # The perfect model believes that hikers are all
        # under 4m tall
        return 1/(1 + numpy.exp(80*(x-4)))

model =PerfectModel()

# Plot the model
import graphings
graphings.scatter_2D(test, trendline =model.predict)
```

```
C:\Users\aduzo\Anaconda3\lib\importlib\_bootstrap.py:219: RuntimeWarning:

numpy.ufunc size changed, may indicate binary incompatibility. Expected 192 from
C header, got 216 from PyObject
```

Our red line is our model, and our blue dots are our datapoints. On the y-axis `0` means tree, and `1` means hiker. We can see our perfect model is passing through every single point.

Now we want to make an ROC curve for this perfect model. There are automated ways to do this, but we're here to learn! It's not so hard to do ourselves. We just need to break it down into steps.

Remember than an ROC curve plots the *true positive rate* (TPR) against the *false positive rate* (FPR). Let's make a function that can calculate these for us. If you're rusty on what these terms mean, pay attention to the code comments:

[ ]:
```
numpy.equal([1,0,1],1)
```

[ ]:
```
array([ True, False,  True])
```

[ ]:
```
numpy.equal([1,0,1],0)
```

```
[ ]: array([False,  True, False])
```

```
[ ]: numpy.sum(numpy.equal([1,0,1],1) & numpy.equal([1,0,1],1))
```

```
[ ]: 2
```

notice and operator.

```
[ ]: def calculate_tpr_fpr(prediction, actual):
         '''
         Calculate true positive rate and false positive rate

         prediction: the labels predicted by the model
         actual: the correct labels we hope the model predicts
         '''

         # To calculate the true positive rate and true negative rate we need to know
         # TP - how many true positives (where the model predicts hiker, and it is a␣
      ↪hiker)
         # TN - how many true negatives (where the model predicts tree, and it is a␣
      ↪tree)
         # FP - how many false positives (where the model predicts hiker, but it was␣
      ↪a tree)
         # FN - how many false negatives (where the model predicts tree, but it was␣
      ↪a hiker)

         # First, make a note of which predictions were 'true' and which were 'false'
         prediction_true = numpy.equal(prediction, 1)
         prediction_false = numpy.equal(prediction, 0)

         # Now, make a note of which correct results were 'true' and which were␣
      ↪'false'
         actual_true = numpy.equal(actual, 1)
         actual_false = numpy.equal(actual, 0)

         # Calculate TP, TN, FP, and FN
         # The combination of sum and '&' counts the overlap
         # For example, TP calculates how many 'true' predictions
         # overlapped with 'true' labels (correct answers)
         TP = numpy.sum(prediction_true  & actual_true)
         TN = numpy.sum(prediction_false & actual_false)
         FP = numpy.sum(prediction_true  & actual_false)
         FN = numpy.sum(prediction_false & actual_true)

             # Calculate the true positive rate
         # This is the proportion of 'hiker' labels that are identified as hikers
         tpr = TP / (TP + FN)
```

```
    # Calculate the false positive rate
    # This is the proportion of 'tree' labels that are identified as hikers
    fpr = FP / (FP + TN)

    # Return both rates
    return tpr, fpr

print("Ready!")
```

```
Ready!
```

Now remember that to make an ROC curve, we calculate TPR and FPR for a wide range of thresholds. We then plot the TPR on the y-axis and the FPR on the x-axis.

First, lets make a short method that can calculate the TPR and FPR for just one decision threshold.

[ ]:
```
test.columns
```

[ ]:
```
Index(['height', 'is_hiker', 'motion', 'texture'], dtype='object')
```

[ ]:
```
def assess_model(model_predict, feature_name, threshold):
    '''
    Calculates the true positive rate and false positive rate of the model
    at a particular decision threshold

    model_predict: the model's predict function
    feature_name: the feature the model is expecting
    threshold: the decision threshold to use
    '''

    # Make model predictions for every sample in the test set
    # What we get back is a probability that the sample is a hiker
    # For example, if we had two samples in the test set, we might
    # get 0.45 and 0.65, meaning the model says there is a 45% chance
    # the first sample is a hiker, and 65% chance the second is a
    # hiker
    probability_of_hiker = model_predict(test[feature_name])

    # See which predictions at this threshold would say hiker
    predicted_is_hiker = probability_of_hiker > threshold

    # calculate the true and false positives rates using our
    # handy method
    return calculate_tpr_fpr(predicted_is_hiker, test.is_hiker)

print("Ready!")
```

```
Ready!
```

Now we can use it in a loop to create an ROC curve:

```
[ ]: numpy.linspace(0,1,10)
```

```
[ ]: array([0.        , 0.11111111, 0.22222222, 0.33333333, 0.44444444,
            0.55555556, 0.66666667, 0.77777778, 0.88888889, 1.        ])
```

```
[ ]: def create_roc_curve(model_predict, feature="height"):
         '''
         This function creates a ROC curve for a given model by testing it
         on the test set for a range of decision thresholds. An ROC curve has

         model_predict: The model's predict function
         feature: The feature to provide the model's predict function
         '''

         #  Make a list of thresholds to try
         thresholds = numpy.linspace(0,1,101)

         false_positive_rates = []
         true_positive_rates = []

         # Loop through all thresholds
         for threshold in thresholds:
             # calculate the true and false positives rates using our
             # handy method
             tpr, fpr =assess_model(model_predict, feature, threshold)

             # save the results
             true_positive_rates.append(tpr)
             false_positive_rates.append(fpr)


         # Graph the result
         # You don't need to understand this code, but essentially we are plotting
         # TPR versus FPR as a line plot
         # -- Prepare a dataframe, required by our graphing code
         df_for_graphing = pandas.DataFrame(dict(fpr=false_positive_rates,␣
       ↪tpr=true_positive_rates, threshold=thresholds))
         # -- Generate the plot
         fig = graphings.scatter_2D(df_for_graphing, x_range=[-0.05,1.05])
         #fig.update_traces(mode='lines') # Comment our this line if you would like␣
       ↪to see points rather than lines
         fig.update_yaxes(range=[-0.05, 1.05])

         # Display the graph
```

```
    fig.show()


# Create an roc curve for our model
create_roc_curve(model.predict)
```

What are we seeing here?

Except for at a threshold of 0, the model always has a true positive rate of 1. It also always has a false positive rate of 0, unless the threshold has been set to 1. Note that because we've drawn a line, it appears that there are intermediate values (such as a FPR of 0.5) but the line is simply deceiving. If you would like, comment out `fig.update_traces(mode='lines')` in the above cell and re-run to see points, rather than lines.

Think about it - our model is perfect. Using it, we will always gets all answers correct, putting all points are in top left corner (unless the threshold is 0 or 1, which effectively mean that we are discarding the model results completely).

1.2 Worse-than-chance

As a counter example to understand the ROC curve, lets consider a model that is worse than chance. In fact, this model gets every single answer wrong.

This doesn't happen often in the real world, so again, we will have to fake this model as well. Let's plot this fake model against our data:

[ ]:
```
# Create a fake model that gets every single answer incorrect
class VeryBadModel:
    def predict(self, x):
        # This model thinks that all people are over 4m tall
        # and all trees are shorter
        return 1 / (1 + numpy.exp(-80*(x - 4)))

model = VeryBadModel()

# Plot the model
graphings.scatter_2D(test, trendline=model.predict)
```

As you can see, the red line (model) goes the wrong direction! How will this look on an ROC curve?

[ ]:
```
# run our code to create the ROC curve
create_roc_curve(model.predict)
```

It is the opposite of the perfect model. Rather than the line reaching the top left of the graph, it reaches the bottom right. This means that the TPR is always 0 - it gets nothing right at all. In this particular example, it also always has a false positive rate of 1, so long as the threshold is less than 1.

1.3 A model no better than chance

The previous two models we've seen are very unusual. We've learned though, that we would like the curve to be as close to the top left of the graph as possible.

What would a model be like that does no better than chance? Let's have a look by trying to fit a model to our texture feature. We know this won't work well, because we've seen that hikers and trees have the same range of image textures.

```
[ ]: train.texture.head()
```

```
[ ]: 789    4.870652
     179    5.263048
     27     4.583643
     33     4.605835
     334    5.463910
     Name: texture, dtype: float64
```

```
[ ]: numpy.column_stack((numpy.full(train.texture.shape,1),train.texture))
```

```
[ ]: array([[1.        , 4.87065228],
            [1.        , 5.26304755],
            [1.        , 4.58364324],
            [1.        , 4.60583487],
            [1.        , 5.46390993],
            [1.        , 5.27030112],
            [1.        , 4.57524061],
            [1.        , 5.12120538],
            [1.        , 5.4626463 ],
            [1.        , 5.30397329],
            [1.        , 4.45310478],
            [1.        , 4.54785568],
            [1.        , 5.15813464],
            [1.        , 5.50584797],
            [1.        , 4.68596549],
            [1.        , 5.25863616],
            [1.        , 4.96553749],
            [1.        , 4.58427746],
            [1.        , 4.59419499],
            [1.        , 4.9811454 ],
            [1.        , 5.67878324],
            [1.        , 5.7059053 ],
            [1.        , 4.82415406],
            [1.        , 4.92069662],
            [1.        , 5.12372693],
            [1.        , 5.28920527],
            [1.        , 3.98502747],
            [1.        , 4.8204897 ],
            [1.        , 4.89606008],
            [1.        , 4.38316342],
            [1.        , 5.46612656],
            [1.        , 4.41868928],
            [1.        , 5.60102968],
            [1.        , 4.96342605],
            [1.        , 5.36141134],
            [1.        , 5.29008882],
            [1.        , 4.97739677],
            [1.        , 4.56272974],
            [1.        , 5.58215432],
            [1.        , 5.26168647],
            [1.        , 5.219281  ],
            [1.        , 4.97115912],
            [1.        , 5.67013596],
            [1.        , 4.99345229],
            [1.        , 4.72108302],
            [1.        , 4.69896469],
            [1.        , 4.55293152],
            [1.        , 4.21711258],
            [1.        , 5.30417769],
            [1.        , 4.93737288],
            [1.        , 4.80342892],
            [1.        , 5.26460565],
            [1.        , 4.71858548],
            [1.        , 5.37117079],
            [1.        , 6.04487046],
            [1.        , 4.84126691],
            [1.        , 4.63359628],
            [1.        , 5.33601428],
            [1.        , 4.26658843],
            [1.        , 4.56137657],
            [1.        , 5.85399309],
            [1.        , 5.06017121],
            [1.        , 5.56757069],
            [1.        , 5.34479641],
            [1.        , 4.4976177 ],
            [1.        , 5.17138671],
            [1.        , 5.43417019],
            [1.        , 6.05631106],
            [1.        , 4.42210417],
            [1.        , 5.26962691],
            [1.        , 5.02263128],
            [1.        , 4.58919976],
            [1.        , 4.11084094],
            [1.        , 4.89658845],
            [1.        , 5.33726411],
            [1.        , 4.69267954],
            [1.        , 5.25015976],
            [1.        , 5.59516162],
            [1.        , 5.32800759],
            [1.        , 5.07484376],
            [1.        , 4.86225946],
            [1.        , 5.04455877],
            [1.        , 4.9197824 ],
            [1.        , 3.7088826 ],
            [1.        , 5.12803972],
            [1.        , 5.9932696 ],
            [1.        , 4.17668261],
            [1.        , 4.34045438],
            [1.        , 4.76950881],
            [1.        , 5.07044365],
            [1.        , 5.03319635],
            [1.        , 5.0028892 ],
            [1.        , 4.99250479],
            [1.        , 4.3312992 ],
            [1.        , 4.6600281 ],
            [1.        , 4.73358402],
            [1.        , 4.85847357],
            [1.        , 4.2207029 ],
            [1.        , 5.20337986],
            [1.        , 4.18176299],
            [1.        , 5.45338319],
            [1.        , 4.09027558],
            [1.        , 5.62744344],
            [1.        , 4.57119984],
            [1.        , 4.47743051],
            [1.        , 4.68873127],
            [1.        , 4.81402003],
            [1.        , 5.64306312],
            [1.        , 4.5682367 ],
            [1.        , 5.28487565],
            [1.        , 4.66603497],
            [1.        , 4.85621219],
            [1.        , 4.88369856],
            [1.        , 4.57740229],
            [1.        , 5.14941751],
            [1.        , 4.34055159],
            [1.        , 5.17454184],
            [1.        , 4.85243234],
            [1.        , 5.00787462],
            [1.        , 4.99808503],
            [1.        , 5.56143487],
            [1.        , 5.26637945],
            [1.        , 4.66555225],
            [1.        , 5.23050798],
            [1.        , 5.44139953],
            [1.        , 5.41114854],
            [1.        , 4.94289087],
            [1.        , 5.00363024],
            [1.        , 4.5704198 ],
            [1.        , 4.95914887],
            [1.        , 4.20646559],
            [1.        , 5.56418927],
            [1.        , 5.08796395],
            [1.        , 5.76071702],
            [1.        , 5.03986098],
            [1.        , 5.13856325],
            [1.        , 5.29372478],
            [1.        , 4.87855581],
            [1.        , 4.67033677],
            [1.        , 5.3831131 ],
            [1.        , 5.13762444],
            [1.        , 5.12342664],
            [1.        , 5.11939793],
            [1.        , 5.12502134],
            [1.        , 4.31457323],
            [1.        , 4.64238393],
            [1.        , 4.67308774],
            [1.        , 4.42214202],
            [1.        , 5.37718153],
            [1.        , 5.5347505 ],
            [1.        , 5.21538882],
            [1.        , 5.45604343],
            [1.        , 4.81544404],
            [1.        , 5.55436896],
            [1.        , 4.06973463],
            [1.        , 5.37638073],
            [1.        , 5.48196846],
            [1.        , 5.43052312],
            [1.        , 5.22435018],
            [1.        , 4.54359495],
            [1.        , 5.18433285],
            [1.        , 4.89217437],
            [1.        , 5.38566765],
            [1.        , 5.62505611],
            [1.        , 4.60586177],
            [1.        , 4.34946696],
            [1.        , 4.58275083],
            [1.        , 5.26092242],
            [1.        , 4.4218122 ],
            [1.        , 4.4585112 ],
            [1.        , 5.42225922],
            [1.        , 4.82785543],
            [1.        , 5.19311194],
            [1.        , 5.32487235],
            [1.        , 4.71169656],
            [1.        , 5.3886044 ],
            [1.        , 5.69863663],
            [1.        , 4.66299135],
            [1.        , 5.38760306],
            [1.        , 5.52652119],
            [1.        , 5.5791909 ],
            [1.        , 5.15871358],
            [1.        , 5.74996258],
            [1.        , 4.8476282 ],
            [1.        , 4.71401001],
            [1.        , 5.3969652 ],
            [1.        , 5.00145414],
            [1.        , 5.08269095],
            [1.        , 5.2102967 ],
            [1.        , 4.4791236 ],
            [1.        , 4.76907525],
            [1.        , 4.46383394],
            [1.        , 5.11799499],
            [1.        , 4.61966892],
            [1.        , 4.64830247],
            [1.        , 4.92760344],
            [1.        , 4.65836464],
            [1.        , 5.00915863],
            [1.        , 4.56291614],
            [1.        , 4.59902646],
            [1.        , 5.79706869],
            [1.        , 5.59056936],
            [1.        , 4.34708352],
            [1.        , 5.48916377],
            [1.        , 5.33814076],
            [1.        , 4.60146805],
            [1.        , 5.347653  ],
            [1.        , 4.62401128],
            [1.        , 4.68331267],
            [1.        , 5.09787296],
            [1.        , 5.54197599],
            [1.        , 4.54352729],
            [1.        , 4.87848813],
            [1.        , 5.33319053],
            [1.        , 4.42934584],
            [1.        , 5.21809076],
            [1.        , 4.18325652],
            [1.        , 5.5887272 ],
            [1.        , 4.63739808],
            [1.        , 4.29925191],
            [1.        , 5.52333069],
            [1.        , 4.70467731],
            [1.        , 4.55848913],
            [1.        , 5.39246886],
            [1.        , 4.58754774],
            [1.        , 5.23937257],
            [1.        , 4.52562537],
            [1.        , 4.08205048],
            [1.        , 5.24129898],
            [1.        , 5.13970068],
            [1.        , 5.37996605],
            [1.        , 4.4057877 ],
            [1.        , 4.34905258],
            [1.        , 5.84723001],
            [1.        , 4.90538565],
            [1.        , 4.29998856],
            [1.        , 4.73210217],
            [1.        , 4.99794847],
            [1.        , 4.87610125],
            [1.        , 5.63733424],
            [1.        , 5.75692037],
            [1.        , 5.08684014],
            [1.        , 4.30640974],
            [1.        , 5.40146833],
            [1.        , 4.29738623],
            [1.        , 5.47125657],
            [1.        , 5.23634516],
            [1.        , 4.3536313 ],
            [1.        , 5.44816082],
            [1.        , 4.82603167],
            [1.        , 3.82156755],
            [1.        , 4.39107697],
            [1.        , 5.3331219 ],
            [1.        , 4.50163011],
            [1.        , 4.47750091],
            [1.        , 6.10433081],
            [1.        , 4.29737117],
            [1.        , 5.25794336],
            [1.        , 4.2231087 ],
            [1.        , 5.56680673],
            [1.        , 6.15346364],
            [1.        , 4.87916811],
            [1.        , 4.82592209],
            [1.        , 5.84550501],
            [1.        , 5.25805844],
            [1.        , 4.08539018],
            [1.        , 5.0409251 ],
            [1.        , 5.67379979],
            [1.        , 4.94072912],
            [1.        , 4.39776128],
            [1.        , 4.67120193],
            [1.        , 5.5019982 ],
            [1.        , 5.45357193],
            [1.        , 5.23814191],
            [1.        , 4.3081607 ],
            [1.        , 4.44530709],
            [1.        , 4.50618964],
            [1.        , 5.30019725],
            [1.        , 4.74375869],
            [1.        , 6.42196928],
            [1.        , 5.79354126],
            [1.        , 4.39706654],
            [1.        , 5.04708658],
            [1.        , 4.10144195],
            [1.        , 5.005659  ],
            [1.        , 4.54370662],
            [1.        , 4.70567695],
            [1.        , 5.1268874 ],
            [1.        , 4.36644635],
            [1.        , 4.9187514 ],
            [1.        , 5.31949298],
            [1.        , 4.60264867],
            [1.        , 5.41640937],
            [1.        , 4.42890185],
            [1.        , 4.73934178],
            [1.        , 4.5743689 ],
            [1.        , 5.0417166 ],
            [1.        , 4.83783409],
            [1.        , 5.23606796],
            [1.        , 5.27015276],
            [1.        , 3.77354007],
            [1.        , 4.4512451 ],
            [1.        , 5.53618094],
            [1.        , 4.86876615],
            [1.        , 4.97638121],
            [1.        , 4.75911181],
            [1.        , 4.43040484],
            [1.        , 4.87284464],
            [1.        , 5.27349911],
            [1.        , 4.73453402],
            [1.        , 4.91566886],
            [1.        , 4.98944809],
            [1.        , 5.27946637],
            [1.        , 4.02845029],
            [1.        , 5.54140654],
            [1.        , 5.52411528],
            [1.        , 5.35004976],
            [1.        , 4.60228053],
            [1.        , 4.76785295],
            [1.        , 5.18728806],
            [1.        , 4.89780759],
            [1.        , 3.99802912],
            [1.        , 4.37884974],
            [1.        , 4.23478963],
            [1.        , 5.04836968],
            [1.        , 5.56952665],
            [1.        , 4.98313524],
            [1.        , 5.08371663],
            [1.        , 5.37912972],
            [1.        , 5.14889952],
            [1.        , 5.2829866 ],
            [1.        , 4.23264997],
            [1.        , 6.00732444],
            [1.        , 5.16243428],
            [1.        , 5.24183237],
            [1.        , 5.64356347],
            [1.        , 4.82751294],
            [1.        , 4.18782052],
            [1.        , 4.72651404],
            [1.        , 6.21079465],
            [1.        , 4.45334805],
            [1.        , 5.09167988],
            [1.        , 5.21895907],
            [1.        , 4.93351711],
            [1.        , 4.9996935 ],
            [1.        , 5.7384068 ],
            [1.        , 4.45801993],
            [1.        , 4.98383104],
            [1.        , 4.23123258],
            [1.        , 4.70769458],
            [1.        , 4.5875862 ],
            [1.        , 4.00343173],
            [1.        , 4.76369344],
            [1.        , 5.17655783],
            [1.        , 5.50849784],
            [1.        , 5.05768516],
            [1.        , 4.92247509],
            [1.        , 5.45662943],
            [1.        , 4.85088068],
            [1.        , 5.29656178],
            [1.        , 4.97715004],
            [1.        , 4.45106365],
            [1.        , 4.72433843],
            [1.        , 5.05190384],
            [1.        , 4.95502078],
            [1.        , 3.95582648],
            [1.        , 5.2626801 ],
            [1.        , 5.49099683],
            [1.        , 5.52743423],
            [1.        , 4.86845937],
            [1.        , 5.10792984],
            [1.        , 4.31641614],
            [1.        , 5.11655632],
            [1.        , 5.13070102],
            [1.        , 5.17758634],
            [1.        , 4.76924351],
            [1.        , 4.65763015],
            [1.        , 4.84003871],
            [1.        , 4.89716114],
            [1.        , 4.39238729],
            [1.        , 4.35386169],
            [1.        , 5.85333092],
            [1.        , 4.56107423],
            [1.        , 5.07269399],
            [1.        , 5.43022722],
            [1.        , 4.36361218],
            [1.        , 4.33861879],
            [1.        , 4.39101331],
            [1.        , 4.5932641 ],
            [1.        , 4.69977473],
            [1.        , 4.62448453],
            [1.        , 5.26546061],
            [1.        , 4.98704195],
            [1.        , 5.13040003],
            [1.        , 4.90752532],
            [1.        , 5.27758499],
            [1.        , 5.23866833],
            [1.        , 5.41617591],
            [1.        , 4.66362873],
            [1.        , 5.13034424],
            [1.        , 4.98459934],
            [1.        , 4.95482782],
            [1.        , 5.49549309],
            [1.        , 5.62956676],
            [1.        , 5.31124747],
            [1.        , 4.68560029],
            [1.        , 5.23894728],
            [1.        , 4.84832444],
            [1.        , 4.79093804],
            [1.        , 4.93510914],
            [1.        , 4.91840821],
            [1.        , 4.65592035],
            [1.        , 4.46986309],
            [1.        , 3.91393864],
            [1.        , 4.86793053],
            [1.        , 5.07514015],
            [1.        , 5.65178773],
            [1.        , 4.72222144],
            [1.        , 5.29344237],
            [1.        , 5.15900843],
            [1.        , 5.25621707],
            [1.        , 5.18493024],
            [1.        , 4.56374878],
            [1.        , 4.83350297],
            [1.        , 4.86331143],
            [1.        , 4.74120974],
            [1.        , 5.51007632],
            [1.        , 5.89616149],
            [1.        , 5.30222354],
            [1.        , 5.36333998],
            [1.        , 4.82208385],
            [1.        , 4.86910915],
            [1.        , 4.67520323],
            [1.        , 5.46759577],
            [1.        , 4.8886938 ],
            [1.        , 4.07640624],
            [1.        , 4.77981903],
            [1.        , 5.14892896],
            [1.        , 5.61514967],
            [1.        , 5.16584703],
            [1.        , 4.81708167],
            [1.        , 5.3923946 ],
            [1.        , 5.43562414],
            [1.        , 4.92440709],
            [1.        , 4.64868986],
            [1.        , 5.95930976],
            [1.        , 5.61395435],
            [1.        , 5.72566756],
            [1.        , 5.27328455],
            [1.        , 5.62742438],
            [1.        , 5.03369646],
            [1.        , 4.6054296 ],
            [1.        , 5.60206376],
            [1.        , 4.93096267],
            [1.        , 4.04688358],
            [1.        , 4.87318981],
            [1.        , 4.29501443],
            [1.        , 4.00162723],
            [1.        , 5.47609106],
            [1.        , 5.07629075],
            [1.        , 4.69077939],
            [1.        , 3.91986529],
            [1.        , 4.92769034],
            [1.        , 4.9541624 ],
            [1.        , 4.81476157],
            [1.        , 4.37430462],
            [1.        , 4.48190572],
            [1.        , 4.60423312],
            [1.        , 3.99082247],
            [1.        , 3.9087873 ],
            [1.        , 4.77061251],
            [1.        , 4.95785287],
            [1.        , 4.86065933],
            [1.        , 5.30148951],
            [1.        , 5.47903426],
            [1.        , 5.3982654 ],
            [1.        , 3.91088745],
            [1.        , 4.11183833],
            [1.        , 4.28289314],
            [1.        , 4.3869887 ],
            [1.        , 5.4462942 ],
            [1.        , 4.51230573],
            [1.        , 4.9596965 ],
            [1.        , 4.54011301],
            [1.        , 5.52007637],
            [1.        , 5.03723855],
            [1.        , 4.50863051],
            [1.        , 5.22479885],
            [1.        , 5.55617433],
            [1.        , 4.73164729],
            [1.        , 5.09715659],
            [1.        , 5.51820851],
            [1.        , 5.93240767],
            [1.        , 5.26242547],
            [1.        , 5.24738112],
            [1.        , 4.55437118],
            [1.        , 5.0169644 ],
            [1.        , 4.96496794],
            [1.        , 6.21248874],
        [1.        , 5.65507962]])
```

[ ]:
```
import statsmodels.api

# This is a helper method that reformats the data to be compatible
# with this particular logistic regression model
prep_data = lambda x: numpy.column_stack((numpy.full(x.shape, 1), x))

# Train a logistic regression model to predict hiker based on texture
model = statsmodels.api.Logit(train.is_hiker, prep_data(train.texture)).fit()

# Plot the model
graphings.scatter_2D(test, label_x="texture", label_y="is_hiker",␣
 ↪trendline=lambda x: model.predict(prep_data(x)))
```

```
Optimization terminated successfully.
         Current function value: 0.693068
         Iterations 3
```

Our model is not very good - it doesn't pass through a single data point and probably will do no better than chance. This seems extreme but when we work with more complicated problems, sometimes it can be hard to find any real pattern in the data. What does this look like on an ROC curve?

[ ]:
```
# run our code to create the ROC curve
create_roc_curve(lambda x: model.predict(prep_data(x)), "texture")
```

It's a diagonal line! Why?

Remember that the model could not find a way to reliably predict the label from the feature. It is making a range of predictions but they are essentially guesswork.

If we have a threshold of 0.5, about half of our probabilities will be above the threshold, meaning that about half of our predictions are `hiker`. Half of the labels are also hiker, but there is no correlation between the two. This means we'll get about half the predicted `hiker` labels correct (TPR = 0.5). We will also get about half the predicted `hiker` labels wrong (FPR = 0.5).

If we increased the threshold to 0.8, it would predict *hiker* 80% of the time. Again, because this is random, it would identify about 80% of the hikers correctly (by chance), and but also 80% of the trees as hikers.

If we continued this approach for all thresholds, we would achieve a diagonal line.

1.4 A realistic model

In the real world, we typically achieve models that perform somewhere between between pure chance (a diagonal line) and perfectly (a line that touches the top left corner).

Let's finally build a more realistic model. We'll try to predict whether a sample is a hiker or not based on motion. Our model should do OK, but it won't be perfect. This is because hikers sometimes sit still (like trees) and trees sometimes blow in the wind (moving, like a hiker).

[ ]:
```python
import statsmodels.api

# Train a logistic regression model to predict hiker based on motion
model = statsmodels.api.Logit(train.is_hiker, prep_data(train.motion),␣
 ↪add_constant=True).fit()

# Plot the model
graphings.scatter_2D(test, label_x="motion", label_y="is_hiker",␣
 ↪trendline=lambda x: model.predict(prep_data(x)))
```

The model (red line) seems sensible, though we know sometimes it will get answers wrong.

Now let's look at the ROC curve:

[ ]:
```python
create_roc_curve(lambda x: model.predict(prep_data(x)), "motion")
```

We can see the curve bulging toward the top left corner, meaning it is working much better than chance.

This is a fairly typical ROC curve for an 'easy' machine-learning problem like this. Harder problems often see the line much more similar to a diagonal line.

By contrast, if we ever came across a line that bulged the opposite direction - to the bottom right - we would know we're doing worse than chance, and something is deeply wrong.

1.5 Summary

We got through it! ROC curves can seem difficult at first, particularly due to the terminology with respect to True and False positives. We built one from scratch though, here to get a feel for how they are working inside. If you found that tough, read through again slowly, and experiment with some of the functions we made. Don't fret - we normally can use existing libraries to do most of this work for us.

The mode important thing to remember with these curves is that we would like our line to be as close to the top left of the graph as possible. A model that can do this is correctly identifying the target (such as hikers) most of the time, without falsely identifying the target (labelling trees as hikers) very often.