Python Data Products

Course 3: Making Meaningful Predictions from Data

Lecture: Evaluating classifiers for ranking

Learning objectives

In this lecture we will...

- Extend our classifier from the previous lecture in order to evaluate its ranking performance
- Demonstrate the precision, recall, and F1 ranking measures

Code example: Precision and Recall

Let's start where we left off in the previous lecture. Previously, we had computed values for the number of **True Positives (TP), False Positives, True Negatives, and False Negatives:**

In [44]:
```
TP = sum([(p and l) for (p,l) in zip(predictions, y_class)])
FP = sum([(p and not l) for (p,l) in zip(predictions, y_class)])
TN = sum([(not p and not l) for (p,l) in zip(predictions, y_class)])
FN = sum([(not p and l) for (p,l) in zip(predictions, y_class)])
```

Code example: Precision and Recall

First, we can use these values to compute the precision and recall:

```
In [50]: precision = TP / (TP + FP)

In [51]: recall = TP / (TP + FN)

In [52]: precision, recall

Out[52]: (0.9688901639458971, 0.9921113722343231)
```

$$\text{precision} = \frac{|\{\text{relevant documents}\} \cap \{\text{retrieved documents}\}|}{|\{\text{retrieved documents}\}|}$$

$$\text{recall} = \frac{|\{\text{relevant documents}\} \cap \{\text{retrieved documents}\}|}{|\{\text{relevant documents}\}|}$$

Code example: Precision and Recall

And the F1-score:

```
In [53]: F1 = 2 * (precision*recall) / (precision + recall)

In [54]: F1
Out[54]: 0.9803632810702313
```

$$F_1 = 2 \cdot \frac{\mathrm{precision} \cdot \mathrm{recall}}{\mathrm{precision} + \mathrm{recall}}$$

Code example: Sorting scores by confidence

Next we want to sort our predictions by confidence.
First we obtain the **confidences** from the model:

```
In [55]: help(model)
```

```
 |      list of weak references to the object (if defined)
 |
 |  ----------------------------------------------------------------------
 |  Methods inherited from sklearn.linear_model.base.LinearClassifierMixin:
 |
 |  decision_function(self, X)
 |      Predict confidence scores for samples.
 |
 |      The confidence score for a sample is th
 |      sample to the hyperplane.
 |
 |      Parameters
 |      ----------
 |      X : {array-like, sparse matrix}, shape = (n_samples, n_features)
 |          Samples.
 |
 |      Returns
 |      -------
 |      array, shape=(n_samples,) if n_classes == 2 else (n_samples, n_classes)
 |          Confidence scores per (sample, class) combination. In the binary
 |          case, confidence score for self.classes_[1] where >0 means this
```

Note: confidence scores are equivalent to $X_i \cdot \theta$.

Code example: Sorting scores by confidence

Then we sort them along with the labels:

In [56]:
```
confidences = model.decision_function(X)
```

In [57]:
```
confidences
```

Out[57]:
```
array([4.27180659, 5.34068692, 7.88047918, ..., 6.77652788, 5.7457588 ,
       1.72125511])
```

In [58]:
```
confidencesAndLabels = list(zip(confidences,y_class))
```

In [59]:
```
confidencesAndLabels
```

Out[59]:
```
[(4.271806590458448, True),
 (5.340686923174397, True),
 (7.880479181532099, True),
 (5.224256954963243, True),
 (6.436088979353579, True),
 (12.960106079412048, True),
 (5.318764178069046, True),
 (6.235572902486643, True),
 (5.305301082711418, True),
```

In [60]:
```
confidencesAndLabels.sort()
confidencesAndLabels.reverse()
```

Code example: Sorting scores by confidence

At this point we can **discard** the confidences:

In [62]:
```
labelsRankedByConfidence = [z[1] for z in confidencesAndLabels]
```

In [63]:
```
labelsRankedByConfidence
```

Out[63]:
```
[True,
 True,
 True,
 True,
 True,
 True,
 True,
 True,
 True,
 True,
 True,
 True,
 True,
 True,
 True,
 True,
 True,
 True,
 True,
 True
```

Code example: Precision@K and Recall@K

Now we can compute Precision@K and Recall@K values:

```
In [64]: def precisionAtK(K, y_sorted):
             return sum(y_sorted[:K]) / K

In [65]: def recallAtK(K, y_sorted):
             return sum(y_sorted[:K]) / sum(y_sorted)

In [66]: precisionAtK(50, labelsRankedByConfidence)

Out[66]: 1.0

In [67]: precisionAtK(1000, labelsRankedByConfidence)

Out[67]: 1.0

In [68]: precisionAtK(10000, labelsRankedByConfidence)

Out[68]: 0.998
```

Code example: Precision@K and Recall@K

Now we can compute Precision@K and Recall@K values:

```
In [69]: recallAtK(50, labelsRankedByConfidence)
Out[69]: 0.0003582483090679812

In [70]: recallAtK(1000, labelsRankedByConfidence)
Out[70]: 0.007164966181359624

In [71]: recallAtK(10000, labelsRankedByConfidence)
Out[71]: 0.07150636248996904
```

Summary of concepts

- Showed how to compute the precision, recall, and F1-score on our sentiment classification example

On your own...

- Adapt the code to compute a **precision-recall curve**, i.e., plot precision@k and recall@k values for each value of k