Chapter 3: Classfication

K Abhiroop Tejomay

2019:09:22

1. Performance Measures

1.1 Accuracy

- Accuracy is generally not the preferred performance measure for classifiers, especially when you are dealing with skewed datasets (i.e., when some classes are much more frequent than others).

```
from sklearn.model_selection import cross_val_score
cross_val_score(sgd_clf, X_train, y_train_5, cv=3, scoring="accuracy")
```

1.2 Confusion Matrix

- The general idea is to count the number of times instances of class A are classified as class B.
- To compute the confusion matrix, you first need to have a set of predictions so that they can be compared to the actual targets.
- You can get predictions on the training set using the `cross_val_predict` function:

```
from sklearn.model_selection import cross_val_predict

y_train_pred = cross_val_predict(sgd_clf, X_train, y_train_5, cv=3)
```

- Use `confusion_matrix` method to compute the confusion matrix

```
from sklearn.metrics import confusion_matrix
confusion_matrix(y_train_5, y_train_pred)
```

- Correctly classified negative instances are called True Negatives (1st row, 1st col).
- Incorrectly classified negative instances as positive are called False Positives (1st row, 2nd col).
- Incorrectly classified positive instances as negative are called False Negatives (2nd row, 1st col).
- Correctly classified positive instances are called True Positives (2nd row, 2nd col).

1.3 Precision

- Precision is given by:

$$\text{precision} = \frac{TP}{TP + FP} \tag{1}$$

- TP is the number of true positives, and FP is the number of false positives.
- Precision can be thought of as: Of all the instances that the **classifier** predicts as the positive class, how many are actually positive?
- Precision is typically used by another metric named Recall, also called Sensitivity or True Positive Rate.

1.4 Recall

- Recall is given by:

$$\text{recall} = \frac{TP}{TP + FN} \tag{2}$$

- Recall can be thought of as: Of all the positive instances that are actually present in the **dataset**, how many are predicted as positive by the classfier?
- Below is a figure showing the Confusion Matrix with Precision and Recall.

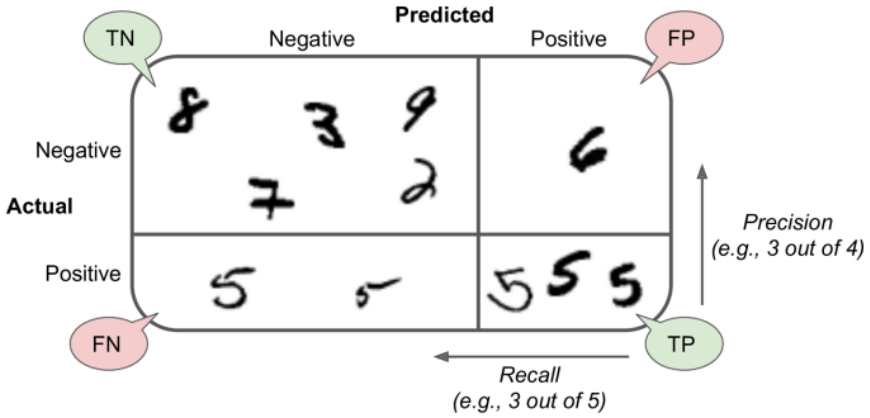

Figure 1: Illustrated Confusion Matrix

```
from sklearn.metrics import precision_score, recall_score
print(precision_score(y_train_5, y_train_pred))
print(recall_score(y_train_5, y_train_pred))
```

- It is often convenient to combine precision and recall into a single metric called the F1 score, in particular if you need a simple way to compare two classifiers.

- The F1 score is the harmonic mean of precision and recall.
- Whereas the regular mean treats all values equally, the harmonic mean gives much more weight to low values.
- As a result, the classifier will only get a high F1 score if both recall and precision are high.

$$F_1 = \frac{2}{\frac{1}{\text{precision}} + \frac{1}{\text{recall}}} = 2 \times \frac{\text{precision} \times \text{recall}}{\text{precision} + \text{recall}} = \frac{TP}{TP + \frac{FN+FP}{2}} \quad (3)$$

- The F1 score favors classifiers that have similar precision and recall. This is not always what you want.
- In some contexts you mostly care about precision, and in other contexts you really care about recall.

1.5 Precision/Recall Tradeoff

- Increasing precision reduces recall, and vice versa. This is called the precision/recall trade-off.
- To understand this trade-off, let's look at how the `SGDClassifier` makes its classification decisions. For each instance, it computes a score based on a decision function. If that score is greater than a threshold, it assigns the instance to the positive class; otherwise it assigns it to the negative class. Figure 2 shows a few digits positioned from the lowest score on the left to the highest score on the right. Suppose the decision threshold is positioned at the central arrow (between the two 5s): you will find 4 true positives (actual 5s) on the right of that threshold, and 1 false positive (actually a 6). Therefore, with that threshold, the precision is 80% (4 out of 5). But out of 6 actual 5s, the classifier only detects 4, so the recall is 67% (4 out of 6). If you raise the threshold (move it to the arrow on the right), the false positive (the 6) becomes a true negative, thereby increasing the precision (up to 100% in this case), but one true positive becomes a false negative, decreasing recall down to 50%. Conversely, lowering the threshold increases recall and reduces precision.

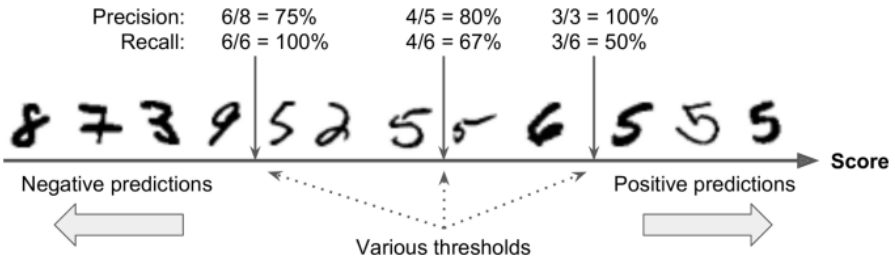

Figure 2: Precision/Recall Tradeoff

- Scikit-Learn does not let you set the threshold directly, but it does give you access to the decision scores that it uses to make predictions.
- Instead of calling the classifier's `predict()` method, you can call its `decision_function()` method.

```
y_scores = sgd_clf.decision_function([some_digit])
y_scores
threshold = 0
y_some_digit_pred = (y_scores > threshold)
```

- How do you decide which threshold to use?
- First, use the `cross_val_predict()` function to get the scores of all instances in the training set, but this time specify that you want to return decision scores instead of predictions:

```
y_scores = cross_val_predict(sgd_clf, X_train, y_train_5, cv=3,
                             method="decision_function")
```

- With these scores, use the $\text{precision}_{\text{recallcurve}}()$ function to compute precision and recall for all possible thresholds:

```
from sklearn.metrics import precision_recall_curve

precisions, recalls, thresholds = precision_recall_curve(y_train_5, y_scores)
```

- Finally, use Matplotlib to plot precision and recall as functions of the threshold value

```
def plot_precision_recall_vs_threshold(precisions, recalls, thresholds):
    plt.plot(thresholds, precisions[:-1], "b--", label="Precision")
    plt.plot(thresholds, recalls[:-1], "g-", label="Recall")
    [...] # highlight the threshold and add the legend, axis label, and grid

plot_precision_recall_vs_threshold(precisions, recalls, thresholds)
plt.show()
```

- Precision may sometimes go down when you raise the threshold (although in general it will go up).
- On the other hand, recall can only go down when the threshold is increased.
- Another way to select a good precision/recall trade-off is to plot precision directly against recall.

- Choose the threshold where the curve starts falling sharply.
- If you just increase the threshold, the precision will increase.
- A high-precision classifier is not very useful if its recall is too low!

1.6 ROC Curve

- Instead of plotting precision versus recall, the ROC curve plots the true positive rate (another name for recall) against the false positive rate (FPR).

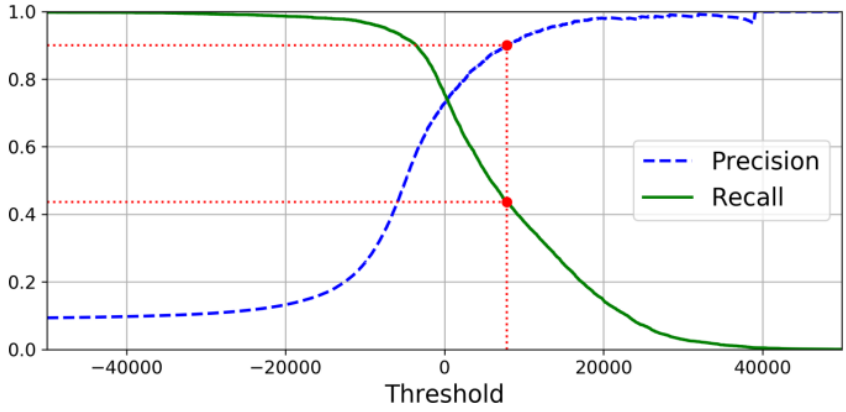

Figure 3: Precision and Recall versus Decision Threshold

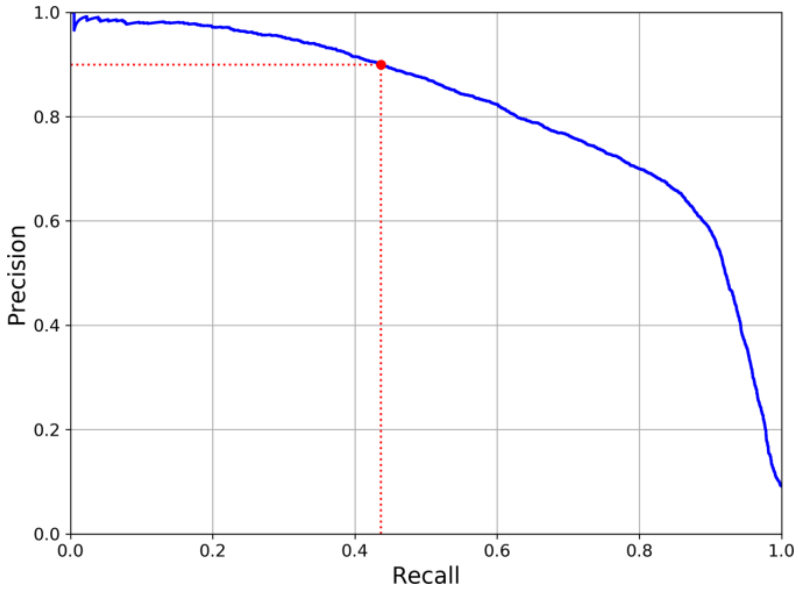

Figure 4: Precision Vs Recall

- The FPR is the ratio of negative instances that are incorrectly classified as positive. It is equal to 1 – the true negative rate (TNR), which is the ratio of negative instances that are correctly classified as negative.
- The TNR is also called specificity.
- Hence, the ROC curve plots sensitivity (recall) versus 1 – specificity.
- Below is code to plot the ROC curve:

```
from sklearn.metrics import roc_curve

fpr, tpr, thresholds = roc_curve(y_train_5, y_scores)

def plot_roc_curve(fpr, tpr, label=None):
    plt.plot(fpr, tpr, linewidth=2, label=label)
    plt.plot([0, 1], [0, 1], 'k--') # Dashed diagonal
    [...] # Add axis labels and grid

plot_roc_curve(fpr, tpr)
plt.show()
```

- Once again there is a trade-off: the higher the recall (TPR), the more false positives (FPR) the classifier produces.
- The dotted line represents the ROC curve of a purely random classifier; a good classifier stays as far away from that line as possible (toward the top-left corner).

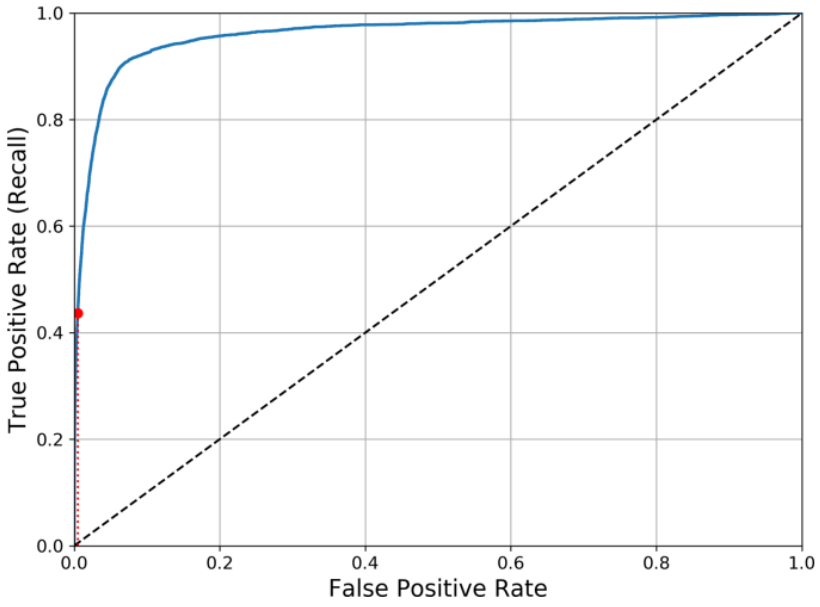

Figure 5: ROC Curve

- As a rule of thumb, you should prefer the PR curve whenever the positive class is rare or when you care more about the false positives than the false negatives. Otherwise, use the ROC curve.
- One way to compare classifiers is to measure the area under the curve (AUC). A perfect classifier will have a ROC AUC equal to 1, whereas a purely random classifier will have a ROC AUC equal to 0.5.

```
from sklearn.metrics import roc_auc_score
roc_auc_score(y_train_5, y_scores)
```

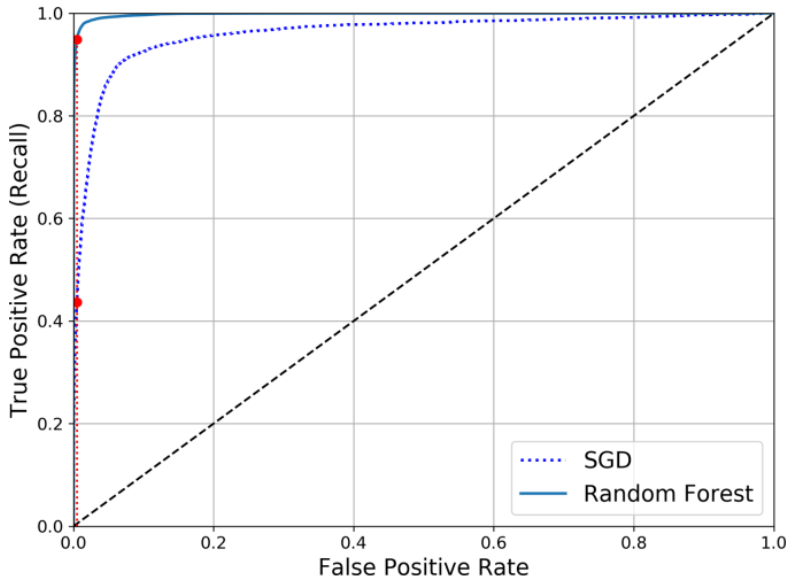

Figure 6: Comparing ROC Curves of Random Forest and SGD Classifiers

2. Multiclass Classification

- Some algorithms (such as SGD classifiers, Random Forest classifiers, and naive Bayes classifiers) are capable of handling multiple classes natively.
- Others (such as Logistic Regression or Support Vector Machine classifiers) are strictly binary classifiers. However, there are various strategies that you can use to perform multiclass classification with multiple binary classifiers.
- If there are 10 classes, you could train 10 binary classifiers and choose the class that outputs the highest score. This is called **One Vs All** Strategy.
- Another strategy is to train a binary classifier for every pair of classes.
- If there are N classes, you need to train N $\times$ (N – 1) / 2 classifiers.
- This is called **One Vs One** Strategy.
- You choose the class which wins the most duels.

- The main advantage of OvO is that each classifier only needs to be trained on the part of the training set for the two classes that it must distinguish.

3. Error Analysis

- Analyzing the confusion matrix often gives you insights into ways to improve your classifier.
- Plot the confusion matrix using `plt.matshow()`. Divide each value in the confusion matrix by the number of images in the corresponding class so that you can compare error rates instead of absolute numbers of errors (which would make abundant classes look unfairly bad):

```
row_sums = conf_mx.sum(axis=1, keepdims=True)
norm_conf_mx = conf_mx / row_sums
```

- Fill the diagonal with zeros to keep only the errors, and plot the result:

```
np.fill_diagonal(norm_conf_mx, 0)
plt.matshow(norm_conf_mx, cmap=plt.cm.gray)
plt.show()
```

- Analyzing individual errors can also be a good way to gain insights on what your classifier is doing and why it is failing, but it is more difficult and time-consuming.

4. Multilabel Classification

- In some cases you may want your classifier to output multiple classes for each instance.
- Such a classification system that outputs multiple binary tags is called a multilabel classification system.
- KNeighborsClassifier instance supports multilabel classification, though not all classifiers do.

5. Multioutput Classification

- It is simply a generalization of multilabel classification where each label can be multiclass (i.e., it can have more than two possible values).

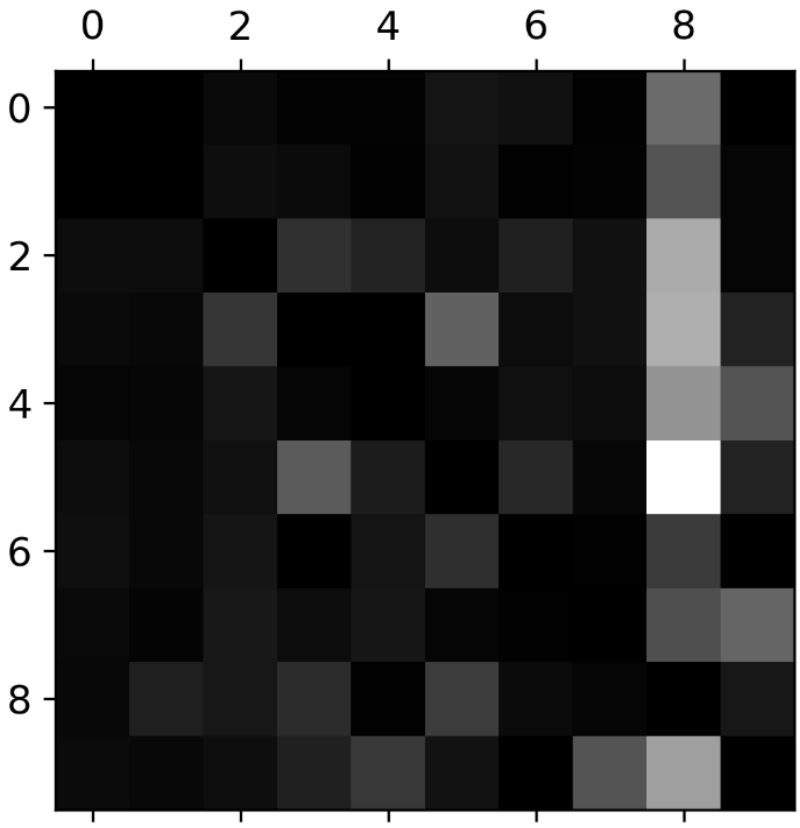

Figure 7: Confusion Matrix Focused on Errors