3.
Multiclass Classification

Multiclass Classification with Binary Classifiers

- Some classification algorithms (e.g. SGD, Random Forest, and naive Bayes classifiers) handle multiple classes natively.
- Others (e.g. Logistic Regression or Support Vector Machine classifiers) are strictly binary classifiers.
- You can use binary classifiers to perform multiclass classification:
  - *One-versus-the-rest* (OvR) strategy (aka *one-vs-all*): train N binary classifiers, one for each class, get the decision score from each classifier and select the class with the highest score.
  - *One-versus-one* (OvO) strategy: train a binary classifier for every pair of classes ($N(N-1)/2$ classifiers), and see which class wins the most duels.

Multiclass Classification with SVM

- Scikit-Learn detects when you try to use a binary classification algorithm for a multiclass classification task, and it automatically runs OvR or OvO, depending on the algorithm.

```
from sklearn.svm import SVC

svm_clf = SVC(random_state=42)
svm_clf.fit(X_train[:2000], y_train[:2000])  # y_train, not y_train_5
```

- Under the hood, Scikit-Learn used the OvO strategy: it trained 45 binary classifiers, and selected the class that won the most duels.

```
svm_clf.predict([some_digit])
```

```
array(['5'], dtype=object)
```

Multiclass Classification with SVC

- If you call the `decision_function()` method, you will see that it returns 10 scores per instance (instead of just 1):

```
some_digit_scores = svm_clf.decision_function([some_digit])
some_digit_scores.round(2)
```

```
array([[ 3.79,  0.73,  6.06,  8.3 , -0.29,  9.3 ,  1.75,  2.77,  7.21,
         4.82]])
```

- You can force Scikit-Learn to use OvO or OvR:

```
from sklearn.multiclass import OneVsRestClassifier

ovr_clf = OneVsRestClassifier(SVC(random_state=42))
ovr_clf.fit(X_train[:2000], y_train[:2000])
```

```
len(ovr_clf.estimators_)
```

```
10
```

Multiclass Classification with SGD

- Multiclass classification with SGD:

```
sgd_clf = SGDClassifier(random_state=42)
sgd_clf.fit(X_train, y_train)
sgd_clf.predict([some_digit])
```

```
array(['3'], dtype='<U1')
```

```
sgd_clf.decision_function([some_digit]).round()
```

```
array([[-31893., -34420.,  -9531.,   1824., -22320.,  -1386., -26189.,
        -16148.,  -4604., -12051.]])
```

- You can find the list of OvO multiclass, OvR multiclass, and inherently multiclass estimators at:

https://scikit-learn.org/stable/modules/multiclass.html

Multiclass Classification with SGD

- Use the `cross_val_score()` function to evaluate the `SGDClassifier`'s accuracy:

```
cross_val_score(sgd_clf, X_train, y_train, cv=3, scoring="accuracy")
```

```
array([0.87365, 0.85835, 0.8689 ])
```

- If you simply scale the input:

```
from sklearn.preprocessing import StandardScaler

scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train.astype("float64"))
cross_val_score(sgd_clf, X_train_scaled, y_train, cv=3, scoring="accuracy")
```

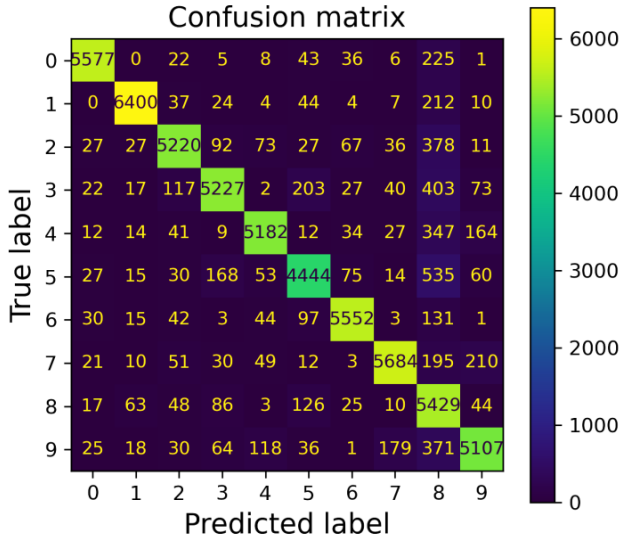
```
array([0.8983, 0.891 , 0.9018])
```

4.
Error Analysis

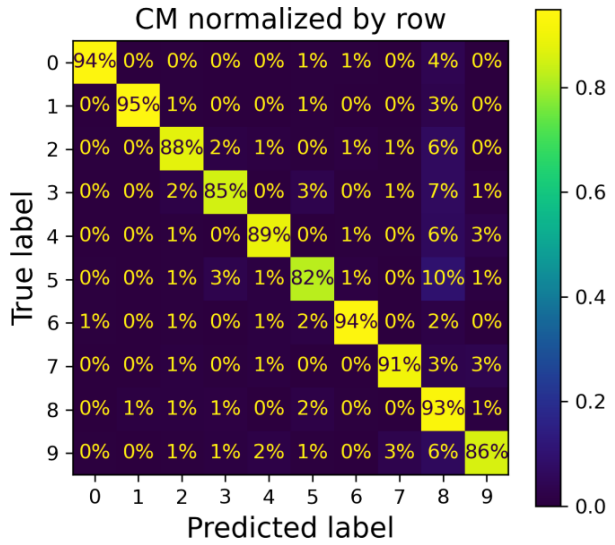
Display Confusion Matrix

```
from sklearn.metrics import ConfusionMatrixDisplay

y_train_pred = cross_val_predict(sgd_clf, X_train_scaled, y_train, cv=3)
ConfusionMatrixDisplay.from_predictions(y_train, y_train_pred)
plt.show()
```

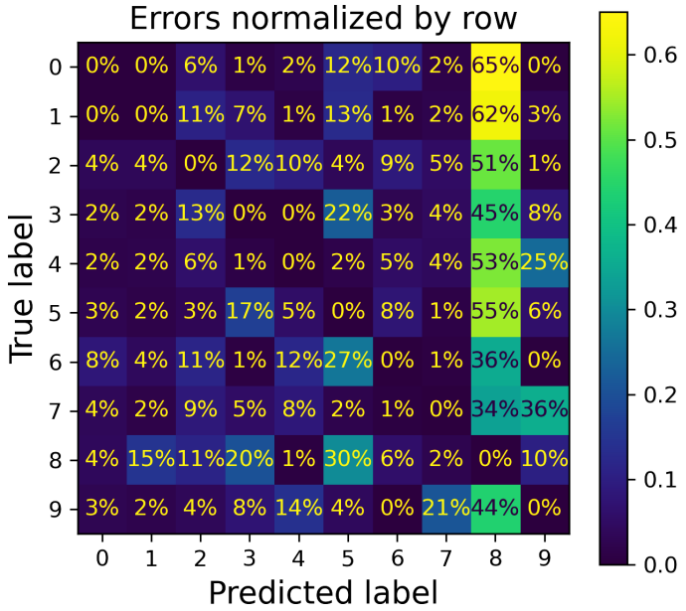



```
normalize="true", values_format=".0%"
```



Display Confusion Matrix

```
sample_weight = (y_train_pred != y_train)
ConfusionMatrixDisplay.from_predictions(y_train, y_train_pred,
                                        sample_weight=sample_weight,
                                        normalize="true", values_format=".0%")
plt.show()
```



Analyzing Confusion Matrix

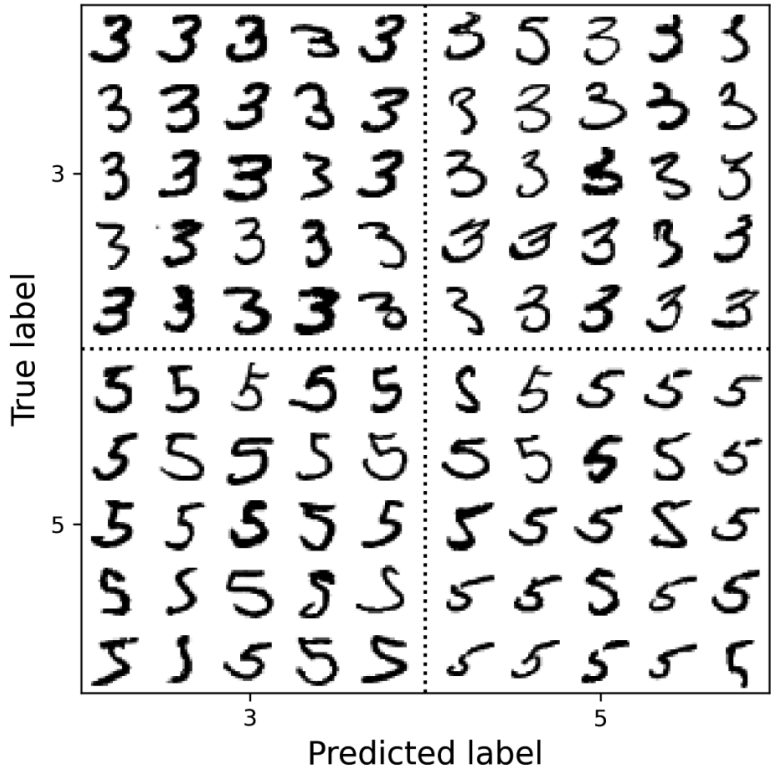
- Looking the confusion matrix plot, it seems that we should try to reduce the false 8s.
  - Gather more training data for digits that look like 8s (but are not) so that the classifier can learn to distinguish them from real 8s.
  - Engineer new features that would help the classifier—for example, writing an algorithm to count the number of closed loops (e.g., 8 has two, 6 has one, 5 has none).
  - Preprocess the images (e.g., using Scikit-Image, Pillow, or OpenCV) to make some patterns, such as closed loops, stand out more.

Analyzing Individual Errors

```
cl_a, cl_b = 3, 5
X_aa = X_train[(y_train == cl_a) & (y_train_pred == cl_a)]
X_ab = X_train[(y_train == cl_a) & (y_train_pred == cl_b)]
X_ba = X_train[(y_train == cl_b) & (y_train_pred == cl_a)]
X_bb = X_train[(y_train == cl_b) & (y_train_pred == cl_b)]

plt.figure(figsize=(8,8))
plt.subplot(221); plot_digits(X_aa[:25], images_per_row=5)
plt.subplot(222); plot_digits(X_ab[:25], images_per_row=5)
plt.subplot(223); plot_digits(X_ba[:25], images_per_row=5)
plt.subplot(224); plot_digits(X_bb[:25], images_per_row=5)
save_fig("error_analysis_digits_plot")
plt.show()
```

- SGD classifier is quite sensitive to image shifting and rotation.
- To reduce the 3/5 confusion we can preprocess the images to ensure that they are well centered and not too rotated.



5.
Multilabel Classification

Multilabel Classification

- *Multilabel Classification*: a classification system that outputs multiple binary tags.

- Consider a face-recognition classifier: what should it do if it recognizes several people in the same picture?
  - It should attach one tag per person it recognizes.
  - If the classifier has been trained to recognize three faces, Alice, Bob, and Charlie. Then when the classifier is shown a picture of Alice and Charlie, it should output [1, 0, 1].

K-Nearest Neighbors (KNN)

- Given integer K and a test observation x_0, the KNN classifier first identifies the K points in the training data that are closest to x_0, represented by N_0.
- It then estimates the conditional probability for class j as the fraction of points in N_0 whose target values equal j:

$$P(Y = j|X = x_0) = \frac{1}{K} \sum_{i \in N_0} I(y_i = j)$$

- Finally, KNN classifes the test observation x_0 to the class with the largest probability.
- KNN supports multilabel classification.

Example

- Create artificial labels (`y_multilabel`) from existing labels:

```
y_train_large = (y_train >= '7')
y_train_odd = (y_train.astype('int8') % 2 == 1)
y_multilabel = np.c_[y_train_large, y_train_odd]
```

- Apply K-Nearest-Neighbors (KNN) classifier:

```
from sklearn.neighbors import KNeighborsClassifier

knn_clf = KNeighborsClassifier()
knn_clf.fit(X_train, y_multilabel)
```

```
knn_clf.predict([some_digit])
```

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

Evaluate a Multilabel Classifier

- Measure the F_1 score (or any other binary classifier metric) for each individual label, then simply compute the average score.

```
y_train_knn_pred = cross_val_predict(knn_clf, X_train, y_multilabel, cv=3)
f1_score(y_multilabel, y_train_knn_pred, average="macro")
```

```
0.976410265560605
```

- If you have many more pictures of Alice than of Bob or Charlie, you may want to give more weight to the classifier's score on pictures of Alice.
- One simple option is to give each label a weight equal to its *support* (i.e., the number of instances with that target label).
  - To do this, set `average="weighted"` when calling the `f1_score()` function.

Chain Prediction

- To use a classifier that does not support multilabel classification, such as SVC, one possible strategy is to train one model per label.
- This won't capture the dependencies between the labels.
  - For example, a large digit (7, 8, or 9) is twice more likely to be odd than even, but the classifier for the "odd" label does not know what the classifier for the "large" label predicted.
- To solve this issue, the models can be organized in a chain: when a model makes a prediction, it uses the input features plus all the predictions of the models that come before it in the chain.
- Scikit-Learn has a class called `ChainClassifier` that does just that.