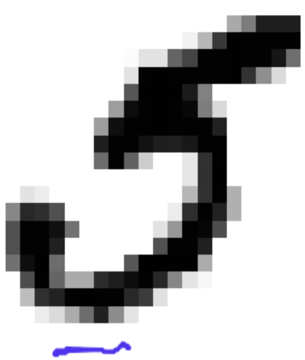
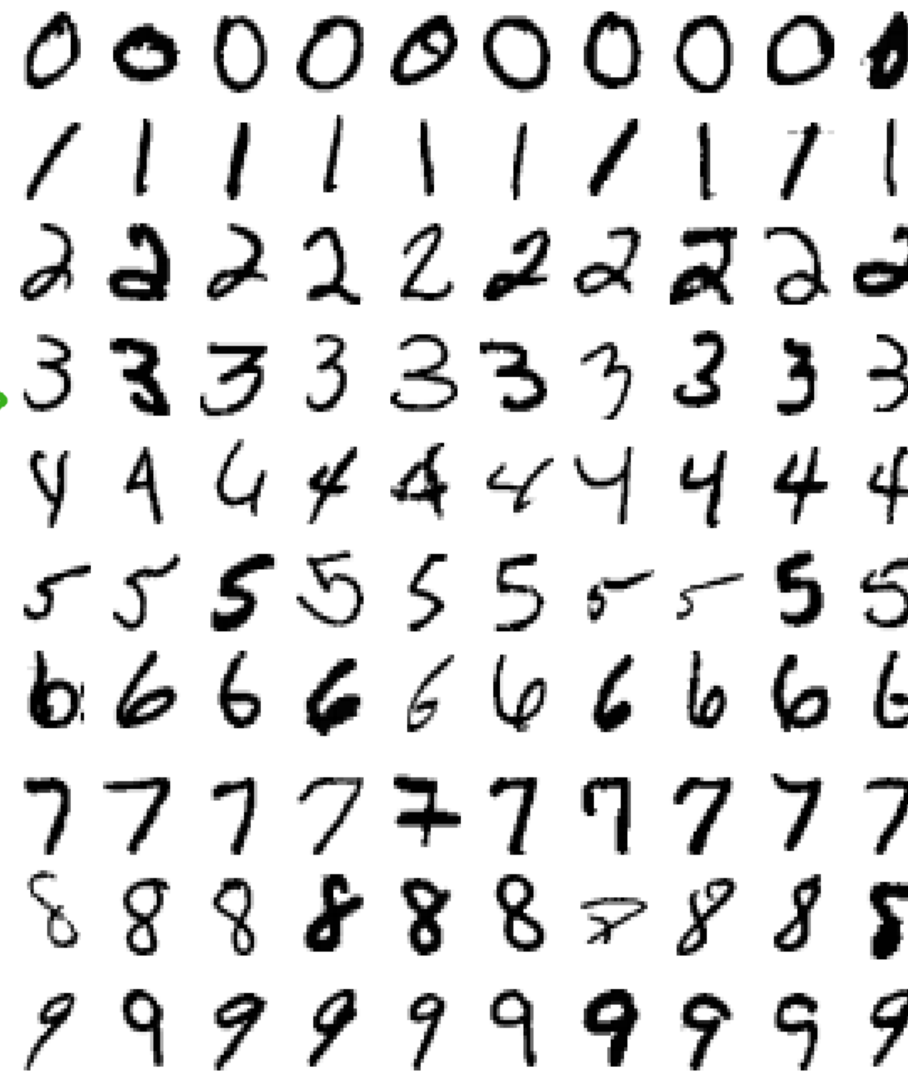
Error Based Learning

Error based learning:MNIST

```
fl1 = fileloader("classification/mnist-original.mat")
fl1.fetch_ml_data()
mnist_raw=fl1.load_ml_data()
mnist = {
    "data": mnist_raw["data"].T,
    "target": mnist_raw["label"][0],
    "COL_NAMES": ["label", "data"],
    "DESCR": "mldata.org dataset: mnist-original",
}
print(mnist)
X, y = mnist["data"], mnist["target"]
print("X.shape:",X.shape)
print("y.shape:",y.shape)
some_digit = X[36000]
some_digit_image = some_digit.reshape(28, 28)
plt.imshow(some_digit_image, cmap = matplotlib.cm.binary,
interpolation="nearest")
plt.axis("off")
plt.show()
print(y[36000])

plt.figure(figsize=(9,9))
#[:12000:600] startindex, endindex , leap
example_images = np.r_[X[:12000:600], X[13000:30600:600], X[30600:60000:590]]
plot_digits(example_images, images_per_row=10)
fl1.save_fig("more_digits_plot")
plt.show()
```



• There are approximately 6000 images of each digit

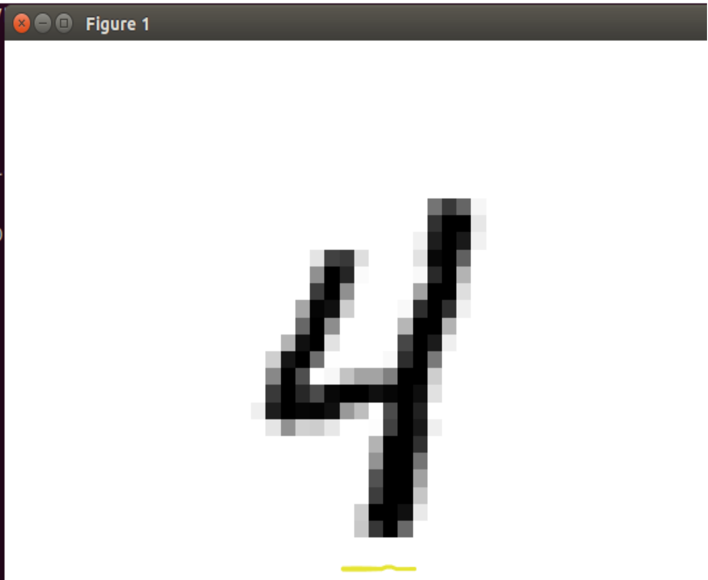


Error based learning:MNIST

```python
X, y = mnist["data"], mnist["target"]
X_train, X_test, y_train, y_test = X[:60000], X[60000:], y[:60000], y[60000:]

shuffle_index = rnd.permutation(60000)
print(shuffle_index)
X_train, y_train = X_train[shuffle_index], y_train[shuffle_index]
print("len(X_train):",len(X_train))
print("len(y_train):",len(y_train))

#true and false for 5 or not
y_train_5 = (y_train == 5)
y_test_5 = (y_test == 5)

print("len(y_train_5):",len(y_train_5))
print("len(y_test_5):",len(y_test_5))

sgd_clf = SGDClassifier(random_state=42)
sgd_clf.fit(X_train, y_train_5)

print("Stochastic Gradient Descent:",sgd_clf)

some_digit_index = 36000
some_digit_image = X_train[some_digit_index]

print("predict:",sgd_clf.predict([some_digit_in
print(y_train[some_digit_index])
plot(some_digit_image)
```

- MNIST data is split for training and testing (60000, 10000)
- Binary Classifier for 5 or not
- SGDClassifier uses a linear SVM Kernel by default.
- And gets it right. But let evaluate the models performance

```
(ml_home) mohit@nomind:~/Work/ArtificialIntelligence$ ./ML/classification/
mnist data already there
[44994 26899 51987 ..., 37093 33655 46428]
len(X_train): 60000
len(y_train): 60000
len(y_train_5): 60000
len(y_test_5): 10000
Stochastic Gradient Descent: SGDClassifier(alpha=0.0001, average=False, cl
       eta0=0.0, fit_intercept=True, l1_ratio=0.15,
       learning_rate='optimal', loss='hinge', n_iter=5, n_jobs=1,
       penalty='l2', power_t=0.5, random_state=42, shuffle=True, verbose=0
       warm_start=False)
predict: [False]
4.0
```



Error based learning:Measure of Error

```
sgd_clf = SGDClassifier(random_state=42)
sgd_scores=cross_val_score(sgd_clf, X_train, y_train_5, cv=3, scoring="accuracy")

def display_scores(scores):
    print("Scores:", scores)
    print("Mean:", scores.mean())
    print("Standard deviation:", scores.std())

display_scores(sgd_scores)

never_5_clf = Never5Classifier()
not5_score=cross_val_score(never_5_clf, X_train, y_train_5, cv=3, scoring="accuracy")
display_scores(not5_score)
```

```
from sklearn.base import BaseEstimator
import numpy as np
class Never5Classifier(BaseEstimator):
    def fit(self, X, y=None):
        pass
    def predict(self, X):
        return np.zeros((len(X), 1), dtype=bool)
```

- This is not a good performance measure as 10% of samples are 5.
- This is the reason why a Never5 Classifier gets it right 90% of the time.

```
./ML/classification/binaryclassifierperformance.py
mnist data already there
[19547 45296 10515 ..., 30496 38753 21637]
len(X_train): 60000
len(y_train): 60000
Scores: [ 0.9385  0.96    0.97  ]
Mean: 0.956166666667
Standard deviation: 0.0131423826691
Scores: [ 0.91065  0.90915  0.90915]
Mean: 0.90965
Standard deviation: 0.000707106781187
```

Error based learning:Measure of Error:Confusion Matrix

```
X, y = mnist["data"], mnist["target"]
X_train, X_test, y_train, y_test = X[:60000], X[60000:], y[:60000], y[60000:]

shuffle_index = rnd.permutation(60000)
print(shuffle_index)
X_train, y_train = X_train[shuffle_index], y_train[shuffle_index]
print("len(X_train):",len(X_train))
print("len(y_train):",len(y_train))

#true and false for 5 or not
y_train_5 = (y_train == 5)
y_test_5 = (y_test == 5)

sgd_clf = SGDClassifier(random_state=42)

y_train_pred = cross_val_predict(sgd_clf, X_train, y_train_5, cv=3)
cm=confusion_matrix(y_train_5, y_train_pred)
print("confusion matrix:",cm)
ps=precision_score(y_train_5, y_train_pred)
print("precision_score:",ps)
rs=recall_score(y_train_5, y_train_pred)
print("recall_score:",rs)
f1s=f1_score(y_train_5, y_train_pred)
print("f1_score:",f1s)
```

- $\begin{bmatrix} TN & FP \\ FN & TP \end{bmatrix}$
- In this case a perfect classifier is only going to have TN and TP or $\begin{bmatrix} 54579 & 0 \\ 0 & 5421 \end{bmatrix}$

rows contain classes, in this case

$\begin{bmatrix} \text{Non 5s} & \text{———} \\ \text{5s} & \text{———} \end{bmatrix}$

Columns contain predicted classes

$\begin{bmatrix} \text{Yays} & \text{Nays} \\ | & | \end{bmatrix}$

```
./ML/classification/confusionmatrix.py
mnist data already there
[32160 45677 47492 ..., 46342 22114 23427]
len(X_train): 60000
len(y_train): 60000

confusion matrix:        [[52392  2187]
                          [ 1072  4349]]

precision_score: 0.665391676867
recall_score: 0.802250507286
f1_score: 0.727439993309
```

Error based learning:Measure of Error:Confusion Matrix

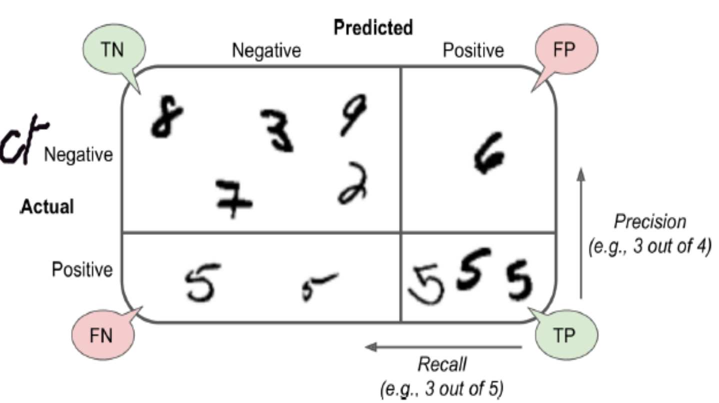
```python
X, y = mnist["data"], mnist["target"]
X_train, X_test, y_train, y_test = X[:60000], X[60000:], y[:60000], y[60000:]

shuffle_index = rnd.permutation(60000)
print(shuffle_index)
X_train, y_train = X_train[shuffle_index], y_train[shuffle_index]
print("len(X_train):",len(X_train))
print("len(y_train):",len(y_train))

#true and false for 5 or not
y_train_5 = (y_train == 5)
y_test_5 = (y_test == 5)

sgd_clf = SGDClassifier(random_state=42)

y_train_pred = cross_val_predict(sgd_clf, X_train, y_train_5, cv=3)
cm=confusion_matrix(y_train_5, y_train_pred)
print("confusion matrix:",cm)
ps=precision_score(y_train_5, y_train_pred)
print("precision_score:",ps)
rs=recall_score(y_train_5, y_train_pred)
print("recall_score:",rs)
f1s=f1_score(y_train_5, y_train_pred)
print("f1_score:",f1s)
```

- $$\text{Precision} = \frac{TP}{TP+FP}$$
- $$\text{Recall} = \frac{TP}{TP+FN}$$

```
./ML/classification/confusionmatrix.py
mnist data already there
[32160 45677 47492 ..., 46342 22114 23427]
len(X_train): 60000
len(y_train): 60000

confusion matrix:        [[52392  2187]
                          [ 1072  4349]]

precision_score: 0.665391676867
recall_score: 0.802250507286
f1_score: 0.727439993309
```

when it claims an image represents a 5, it is 66% times correct

It detect only 80% of 5s.



Error based learning:Measure of Error:Confusion Matrix

```
X, y = mnist["data"], mnist["target"]
X_train, X_test, y_train, y_test = X[:60000], X[60000:], y[:60000], y[60000:]

shuffle_index = rnd.permutation(60000)
print(shuffle_index)
X_train, y_train = X_train[shuffle_index], y_train[shuffle_index]
print("len(X_train):",len(X_train))
print("len(y_train):",len(y_train))

#true and false for 5 or not
y_train_5 = (y_train == 5)
y_test_5 = (y_test == 5)

sgd_clf = SGDClassifier(random_state=42)

y_train_pred = cross_val_predict(sgd_clf, X_train, y_train_5, cv=3)
cm=confusion_matrix(y_train_5, y_train_pred)
print("confusion matrix:",cm)
ps=precision_score(y_train_5, y_train_pred)
print("precision_score:",ps)
rs=recall_score(y_train_5, y_train_pred)
print("recall_score:",rs)
f1s=f1_score(y_train_5, y_train_pred)
print("f1_score:",f1s)
```

- It is often convenient to combine precision and recall into a single metric.
- $F_1 = \dfrac{2}{1/p + 1/R} = 2 \times \dfrac{P \times R}{P + R} = \dfrac{TP}{TP + \left(\dfrac{FN + FP}{2}\right)}$
- F1 score is metric that builds both precision and recall into it. But the percentage of their contribution is not clear.

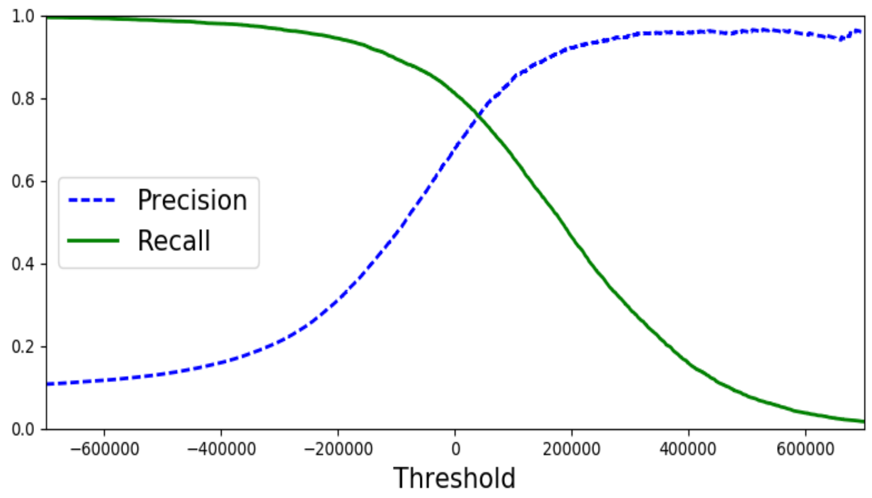
```
./ML/classification/confusionmatrix.py
mnist data already there
[32160 45677 47492 ..., 46342 22114 23427]
len(X_train): 60000
len(y_train): 60000


confusion matrix:        [[52392  2187]
                          [ 1072  4349]]

precision_score: 0.665391676867
recall_score: 0.802250507286
f1_score: 0.727439993309
```

- But often classifiers need to have high precision at the cost of low recall. e.g. If detecting good videos for kids.

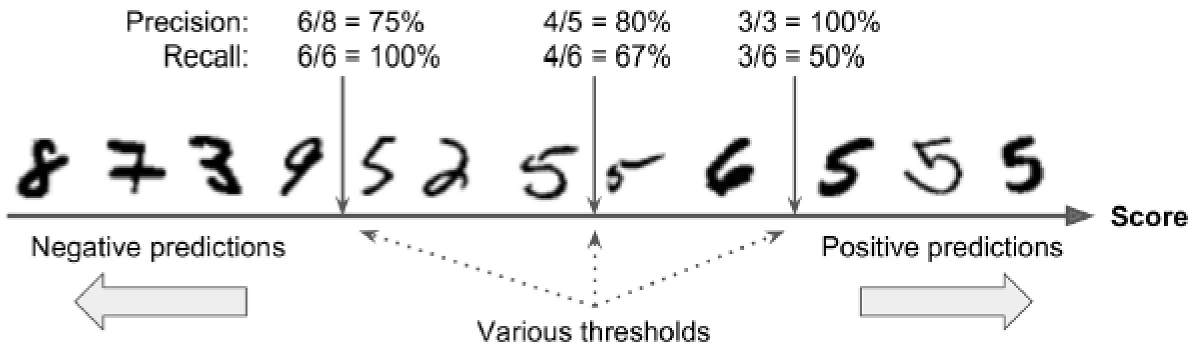
Error based learning:Measure of Error:Precision/Recall Trade-off

```
X, y = mnist["data"], mnist["target"]
X_train, X_test, y_train, y_test = X[:60000], X[60000:], y[:60000], y[60000:]

shuffle_index = rnd.permutation(60000)
print(shuffle_index)
X_train, y_train = X_train[shuffle_index], y_train[shuffle_index]
print("len(X_train):",len(X_train))
print("len(y_train):",len(y_train))

#true and false for 5 or not
y_train_5 = (y_train == 5)
y_test_5 = (y_test == 5)

print("len(y_train_5):",len(y_train_5))
print("len(y_test_5):",len(y_test_5))

sgd_clf = SGDClassifier(random_state=42)
y_scores = cross_val_predict(sgd_clf, X_train, y_train_5, cv=3, method="decision_function")
precisions, recalls, thresholds = precision_recall_curve(y_train_5, y_scores)

plt.figure(figsize=(8, 4))
plot_precision_recall_vs_threshold(precisions, recalls, thresholds)
plt.xlim([-700000, 700000])
fl1.save_fig("precision_recall_vs_threshold_plot")
plt.show()
```

- SGDClassifier computes a score based on decision function, and if the score is greater than threshold then it assigns it to positive class, and negative class otherwise.



Error based learning:Measure of Error:Precision/Recall Trade-off

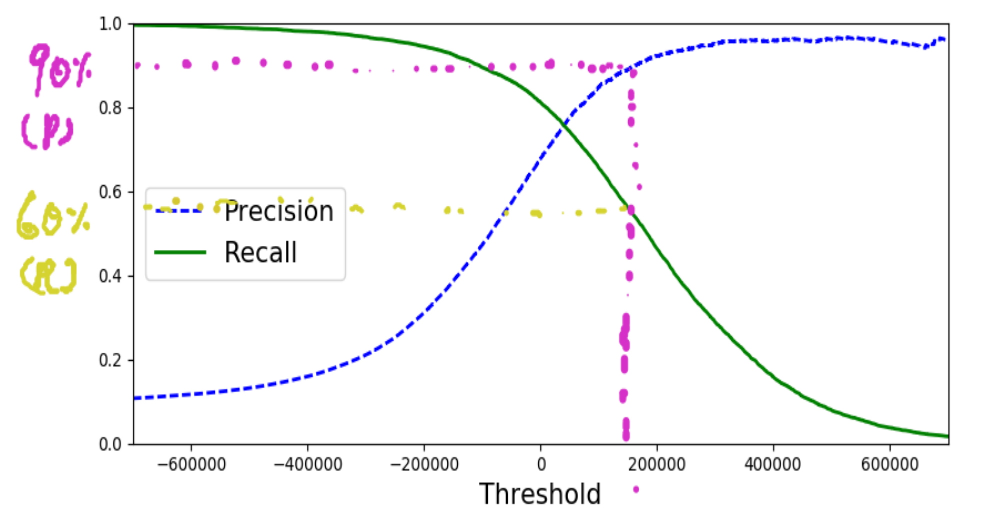


Error based learning:Measure of Error:Precision/Recall Trade-off

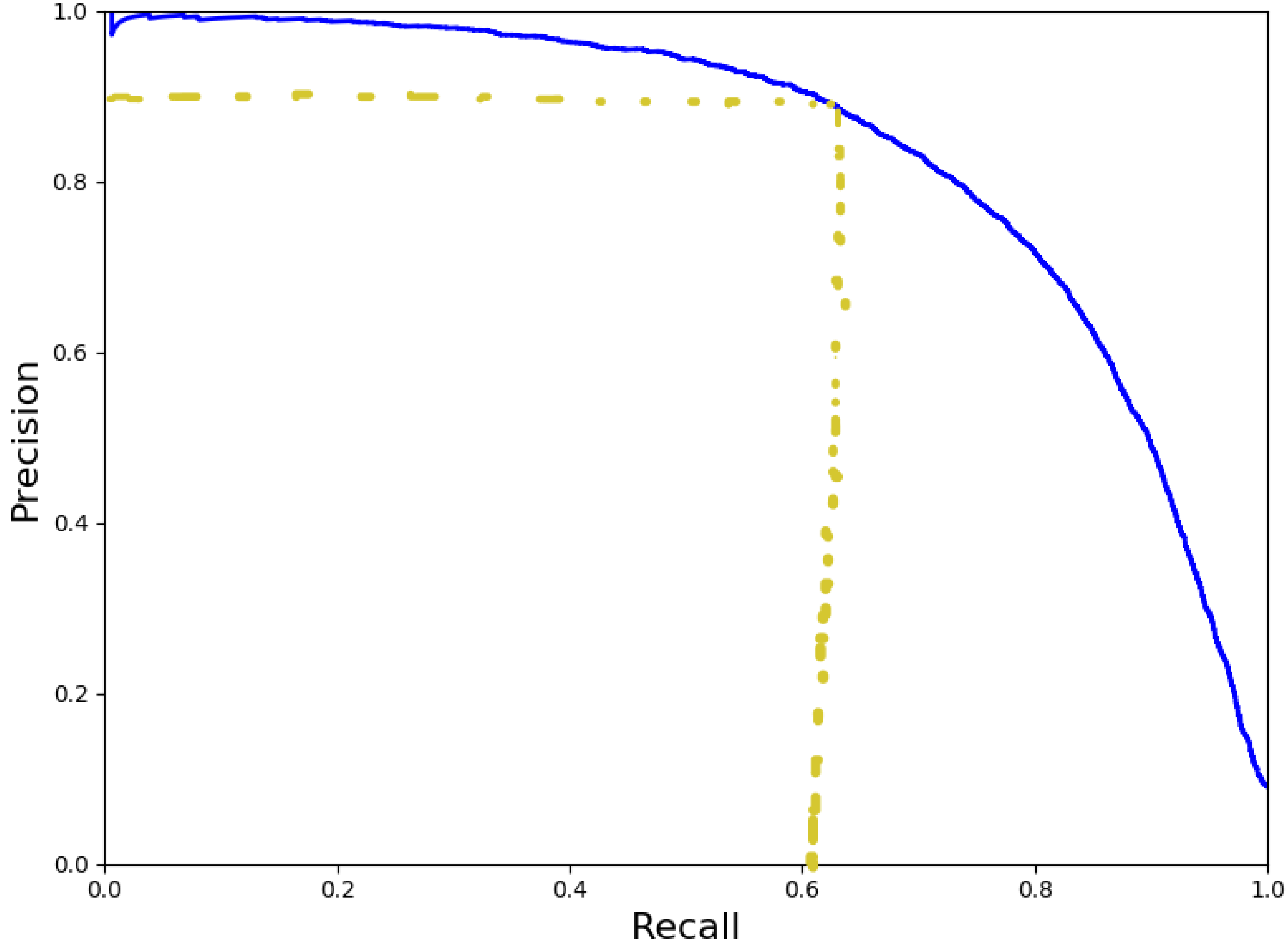
```
sgd_clf = SGDClassifier(random_state=42)
y_train_pred = cross_val_predict(sgd_clf, X_train, y_train_5, cv=3)
cm=confusion_matrix(y_train_5, y_train_pred)
print("confusion matrix:",cm)
ps=precision_score(y_train_5, y_train_pred)
print("precision_score:",ps)
rs=recall_score(y_train_5, y_train_pred)
print("recall_score:",rs)
f1s=f1_score(y_train_5, y_train_pred)
print("f1_score:",f1s)

sgd_clf = SGDClassifier(random_state=42)
y_scores = cross_val_predict(sgd_clf, X_train, y_train_5, cv=3, method="decision_function")
precisions, recalls, thresholds = precision_recall_curve(y_train_5, y_scores)

#at threashold 0 perdictions are equal to descision scores
print("y_scores:",y_scores)
print("At Threshold 0:",(y_train_pred == (y_scores > 0)).all())
threshold=120000;
y_train_pred_90 = (y_scores > threshold)

ps=precision_score(y_train_5, y_train_pred_90)
print("precision_score:at threshold",threshold,":",ps)

rs=recall_score(y_train_5, y_train_pred_90)
print("recall_score::at threshold",threshold,":",rs)

def plot_precision_vs_recall(precisions, recalls):
    plt.plot(recalls, precisions, "b-", linewidth=2)
    plt.xlabel("Recall", fontsize=16)
    plt.ylabel("Precision", fontsize=16)
    plt.axis([0, 1, 0, 1])
```

```
(ml_home) mohit@nomind:~/Work/ArtificialIntelligence$ ./ML/classification/controllingprecesion.py
mnist data already there
[11898 29983 18894 ..., 26027 46993 49459]
len(X_train): 60000
len(y_train): 60000
confusion matrix: [[53381  1198]
 [ 1332  4089]]
precision_score: 0.773406468697
recall_score: 0.754288876591
f1_score: 0.763728053792
y_scores: [-441369.74590094 -283942.69402507  -47862.86278212 ...,  -94675.93944119
 -305110.12070577 -790079.64791734]
At Threshold 0: True
precision_score:at threshold 120000 : 0.92068243041
recall_score::at threshold 120000 : 0.567422984689
```

- Scikit does not allow changing threshold. Which defaults to 0.
- But decision function can be accessed and compared agianst manual threshold.



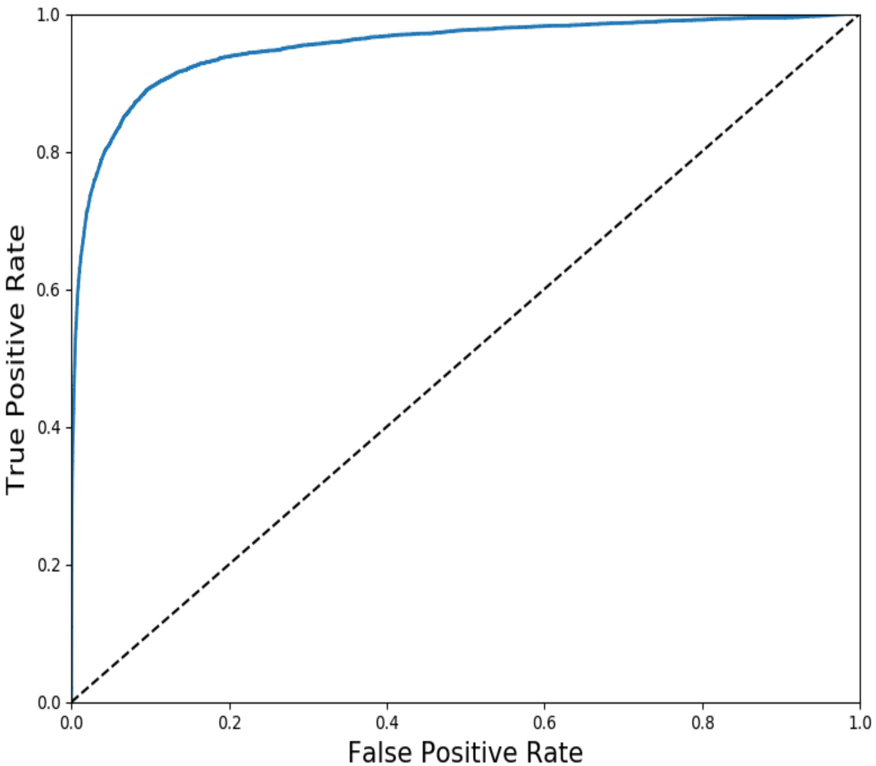
Error based learning:Measure of Error:Precision/Recall Trade-off



- Precision against recall directly

Error based learning:Measure of Error:ROC Curve

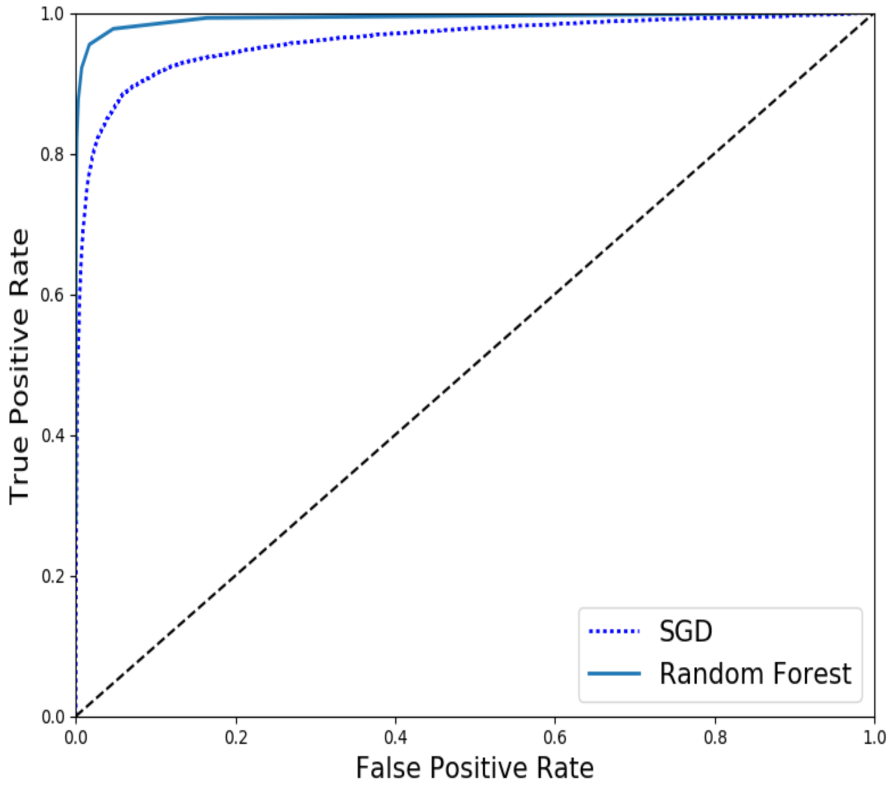
```
sgd_clf = SGDClassifier(random_state=42)
y_scores = cross_val_predict(sgd_clf, X_train, y_train_5, cv=3, method="decision_function"
fpr, tpr, thresholds = roc_curve(y_train_5, y_scores)

def plot_roc_curve(fpr, tpr, **options):
    plt.plot(fpr, tpr, linewidth=2, **options)
    plt.plot([0, 1], [0, 1], 'k--')
    plt.axis([0, 1, 0, 1])
    plt.xlabel('False Positive Rate', fontsize=16)
    plt.ylabel('True Positive Rate', fontsize=16)

plt.figure(figsize=(8, 6))
plot_roc_curve(fpr, tpr)
fl1.save_fig("roc_curve_plot")
plt.show()

print(roc_auc_score(y_train_5, y_scores))
```



- TPR or true positive rate, or recall.
- FPR is the ratio of negative intances that are incorrectly classified as positive.
- TNR = TN/(TN+FN) also called specificity
- FPR = 1-TNR

Error based learning:Measure of Error:ROC Curve

```
sgd_clf = SGDClassifier(random_state=42)
y_scores = cross_val_predict(sgd_clf, X_train, y_train_5, cv=3, method="decision_function")
fpr, tpr, thresholds = roc_curve(y_train_5, y_scores)

forest_clf = RandomForestClassifier(random_state=42)
y_probas_forest = cross_val_predict(forest_clf, X_train, y_train_5, cv=3, method="predict_proba")
y_scores_forest = y_probas_forest[:, 1] # score = proba of positive class
fpr_forest, tpr_forest, thresholds_forest = roc_curve(y_train_5, y_scores_forest)

print("AUC SGD:",roc_auc_score(y_train_5, y_scores))
print("AUC RFR:",roc_auc_score(y_train_5, y_scores_forest))
```




```
AUC SGD: 0.960389854613
AUC RFR: 0.992464264005
```

- Due to the way RF works it has a "predict_proba" function.
- One way to measure the performance to calculate the AUC or area under the curve.
- Bigger is better.

Error based learning:Multiclass Classification

- Some algorithms (such as Random Forest classifiers or naive Bayes classifiers) are capable of handling multiple classes directly.
- Others (such as Support Vector Machine classifiers or Linear classifiers) are strictly binary classifiers.
  - There are 2 strategies:
    - OVA
    - OVO

Error based learning:Multiclass Classification

- OVA
  - One way to create a system that can classify the digit images into 10 classes (from 0 to 9) is to train 10 binary classifiers, one for each digit (a 0-detector, a 1-detector, a 2-detector, and so on).
  - Then when you want to classify an image, you get the decision score from each classifier for that image and you select the class whose classifier outputs the highest score.
  - This is called the one-versus-all (OvA) strategy (also called one-versus-the-rest).

Error based learning:Multiclass Classification

- OVO
  - Another strategy is to train a binary classifier for every pair of digits: one to distinguish 0s and 1s, another to distinguish 0s and 2s, another for 1s and 2s, and so on.
  - This is called the one-versus-one (OvO) strategy. If there are N classes, you need to train N × (N – 1) / 2 classifiers. For the MNIST problem, this means training 45 binary classifiers!
  - Some algorithms (such as **Support Vector Machine** classifiers) scale poorly with the size of the training set, so for these algorithms OvO is preferred since it is faster to train many classifiers on small training sets than training few classifiers on large training sets.

Error based learning:Multiclass Classification

- Scikit-Learn detects when you try to use a binary classification algorithm for a multiclass classification task, and it automatically runs OvA (except for SVM classifiers for which it uses OvO).

Error based learning:Multiclass Classification

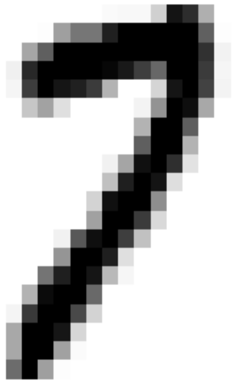
```
mnist_raw=fl1.load_ml_data()
mnist = {
    "data": mnist_raw["data"].T,
    "target": mnist_raw["label"][0],
    "COL_NAMES": ["label", "data"],
    "DESCR": "mldata.org dataset: mnist-original",
}

X, y = mnist["data"], mnist["target"]
X_train, X_test, y_train, y_test = X[:60000], X[60000:], y[:60000], y[60000:]

shuffle_index = rnd.permutation(60000)
print(shuffle_index)
X_train, y_train = X_train[shuffle_index], y_train[shuffle_index]

sgd_clf = SGDClassifier(random_state=42)
sgd_clf.fit(X_train, y_train)

print("Stochastic Gradient Descent:",sgd_clf)

some_digit_index = 36000
some_digit_image = X_train[some_digit_index]

print("predict:",sgd_clf.predict([some_digit_image]))
print("label:",y_train[some_digit_index])
plot(some_digit_image)
```

```
(ml_home) mohit@nomind:~/Work/ArtificialIntelligence$ ./ML/classification/multiclassclassification.py
mnist data already there
[20297 38156 48820 ..., 17344 30378 19114]
Stochastic Gradient Descent: SGDClassifier(alpha=0.0001, average=False, class_weight=None, epsilon=0.1,
       eta0=0.0, fit_intercept=True, l1_ratio=0.15,
       learning_rate='optimal', loss='hinge', n_iter=5, n_jobs=1,
       penalty='l2', power_t=0.5, random_state=42, shuffle=True, verbose=0,
       warm_start=False)
predict: [ 3.]
label: 3.0
```

Error based learning:Multiclass Classification

```python
X, y = mnist["data"], mnist["target"]
X_train, X_test, y_train, y_test = X[:60000], X[60000:], y[:60000], y[60000:]

shuffle_index = rnd.permutation(60000)
X_train, y_train = X_train[shuffle_index], y_train[shuffle_index]

sgd_clf = SGDClassifier(random_state=42)
sgd_clf.fit(X_train, y_train)

print("Stochastic Gradient Descent:",sgd_clf)

some_digit_index = 36000
some_digit_image = X_train[some_digit_index]

some_digit_score=sgd_clf.decision_function([some_digit_image])
print("decision_function:",some_digit_score)
print("decision_function max:",np.argmax(some_digit_score))
print("label:",y_train[some_digit_index])
print("sgd_clf.classes_:",sgd_clf.classes_)
plot(some_digit_image)
```

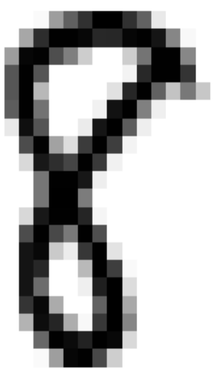
```
mnist data already there
Stochastic Gradient Descent: SGDClassifier(alpha=0.0001, average=False, class_weight=None, epsilon=0.1,
       eta0=0.0, fit_intercept=True, l1_ratio=0.15,
       learning_rate='optimal', loss='hinge', n_iter=5, n_jobs=1,
       penalty='l2', power_t=0.5, random_state=42, shuffle=True, verbose=0,
       warm_start=False)
decision_function: [[-341701.84419277 -523144.58856338 -268950.11826992 -110489.7361237
  -413416.34267046 -457645.01662802 -960132.29296566  176449.00202004
  -280997.79266231 -263782.7401648 ]]
decision_function max: 7
label: 7.0
sgd_clf.classes_: [ 0.  1.  2.  3.  4.  5.  6.  7.  8.  9.]
```



Error based learning:Multiclass Classification

```python
X, y = mnist["data"], mnist["target"]
X_train, X_test, y_train, y_test = X[:60000], X[60000:], y[:60000], y[60000:]

shuffle_index = rnd.permutation(60000)
X_train, y_train = X_train[shuffle_index], y_train[shuffle_index]

from sklearn.multiclass import OneVsOneClassifier
ovo_clf = OneVsOneClassifier(SGDClassifier(random_state=42))
ovo_clf.fit(X_train, y_train)

some_digit_index = 36000
some_digit_image = X_train[some_digit_index]
print("predict:",ovo_clf.predict([some_digit_image]))
print("label:",y_train[some_digit_index])
#(N X N-1)/2
print("num estimators:",len(ovo_clf.estimators_))
plot(some_digit_image)
```

- Scikit can be forced to use "OneVsOneClassifier(OVO)" or "OneVsRestClassifier(OVA).

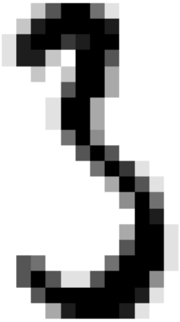


```
(ml_home) mohit@nomind:~/Work/ArtificialIntelligence$ ./ML/classification/multiclassovo.py
mnist data already there
predict: [ 8.]
label: 8.0
num estimators: 45
```

Error based learning:Multiclass Classification

```
X, y = mnist["data"], mnist["target"]
X_train, X_test, y_train, y_test = X[:60000], X[60000:], y[:60000], y[60000:]

shuffle_index = rnd.permutation(60000)
X_train, y_train = X_train[shuffle_index], y_train[shuffle_index]

#random forest can directly classify multiple classes
forest_clf = RandomForestClassifier(random_state=42)
forest_clf.fit(X_train, y_train)

some_digit_index = 36000
some_digit_image = X_train[some_digit_index]
print("predict:",forest_clf.predict([some_digit_image]))
print("label:",y_train[some_digit_index])
print("probablities assigned:",forest_clf.predict_proba([some_digit_image]))
plot(some_digit_image)
```

• RFC is naturally a multi-class classifier.



```
(ml_home) mohit@nomind:~/Work/ArtificialIntelligence$ ./ML/classification/multiclassrfr.py
mnist data already there
predict: [ 3.]
label: 3.0
probablities assigned: [[ 0.  0.  0.  1.  0.  0.  0.  0.  0.  0.]]
```

Error based learning:Multiclass Classification

```
X, y = mnist["data"], mnist["target"]
X_train, X_test, y_train, y_test = X[:60000], X[60000:], y[:60000], y[60000:]

shuffle_index = rnd.permutation(60000)
X_train, y_train = X_train[shuffle_index], y_train[shuffle_index]

#random forest can directly classify multiple classes
scaler = StandardScaler()
X_train_scaledr = scaler.fit_transform(X_train.astype(np.float64))
forest_clf = RandomForestClassifier(random_state=42)
print("RFR cross validation score(standard scaler):",cross_val_score(forest_clf, X_train_scaledr, y_train, cv=3, scoring="a

forest_clf = RandomForestClassifier(random_state=42)
print("RFR cross validation score:",cross_val_score(forest_clf, X_train, y_train, cv=3, scoring="accuracy"))

sgd_clf = SGDClassifier(random_state=42)
print("SGD cross validation score:",cross_val_score(sgd_clf, X_train, y_train, cv=3, scoring="accuracy"))

scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train.astype(np.float64))
print("SGD cross validation score(standard scaler):",cross_val_score(sgd_clf, X_train_scaled, y_train, cv=3, scoring="accur
```

```
(ml_home) mohit@nomind:~/Work/ArtificialIntelligence$ ./ML/classification/multiclasscv.py
mnist data already there
RFR cross validation score(standard scaler): [ 0.93956209  0.940097    0.93969095]
RFR cross validation score: [ 0.93966207  0.94014701  0.93959094]
SGD cross validation score: [ 0.85542891  0.86169308  0.85792869]
SGD cross validation score(standard scaler): [ 0.90726855  0.91139557  0.90888633]
```

- As usual cross validate and fine tune the model.

Error based learning:Multiclass Classification:Confusion Matrix

```
X, y = mnist["data"], mnist["target"]
X_train, X_test, y_train, y_test = X[:60000], X[60000:], y[:60000], y[60000:]

shuffle_index = rnd.permutation(60000)
X_train, y_train = X_train[shuffle_index], y_train[shuffle_index]

#random forest can directly classify multiple classes
sgd_clf = SGDClassifier(random_state=42)
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train.astype(np.float64))
y_train_pred=cross_val_predict(sgd_clf, X_train_scaled, y_train, cv=3)
conf_mx = confusion_matrix(y_train, y_train_pred)
#along the diagonals it predicted right and rest are wrongs
print("conf_mx:",conf_mx)

plt.matshow(conf_mx, cmap=plt.cm.gray)
fl1.save_fig("confusion_matrix_plot", tight_layout=False)

row_sums = conf_mx.sum(axis=1, keepdims=True)
#sum of all classes by classes
print("row_sums:",row_sums)
norm_conf_mx = conf_mx / row_sums
#after div we get the error rates.
print("norm_conf_mx:",norm_conf_mx)
np.fill_diagonal(norm_conf_mx, 0)
plt.matshow(norm_conf_mx, cmap=plt.cm.gray)
fl1.save_fig("confusion_matrix_errors_plot", tight_layout=False)
plt.show()
```

- Confusion matrix has lots of numbers this time.
- rows represent classes as before, but positive identification is along the numbers index, which is the diagnal.
- 0 classified as 0.
- 0 misclassified as 1 and so on.

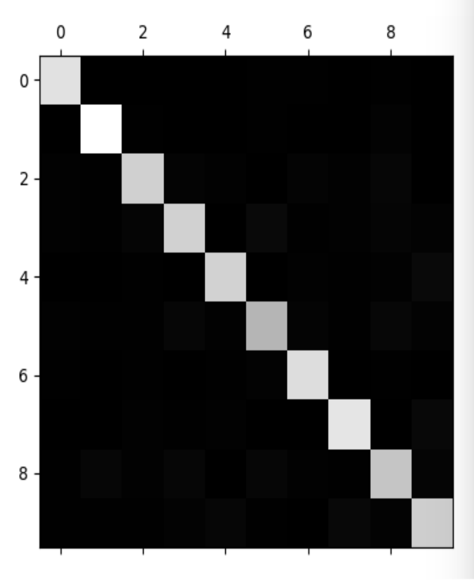
```
conf_mx: [[5737    2   21   11   11   41   49    8   37    6]
 [   1 6474   50   28    6   37    8   11  116   11]
 [  63   40 5326   99   86   19   85   67  159   14]
 [  48   40  127 5366    1  224   36   55  126  108]
 [  20   24   42    8 5356    8   50   24   86  224]
 [  72   46   37  198   71 4586  110   27  179   95]
 [  35   29   55    2   49   88 5600    9   51    0]
 [  24   19   71   32   60    8    5 5784   15  247]
 [  53  152   73  156   13  154   54   30 5022  144]
 [  44   28   26   79  153   33    2  211   84 5289]]
```

Error based learning:Multiclass Classification:Confusion Matrix

```python
X, y = mnist["data"], mnist["target"]
X_train, X_test, y_train, y_test = X[:60000], X[60000:], y[:60000], y[60000:]

shuffle_index = rnd.permutation(60000)
X_train, y_train = X_train[shuffle_index], y_train[shuffle_index]

#random forest can directly classify multiple classes
sgd_clf = SGDClassifier(random_state=42)
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train.astype(np.float64))
y_train_pred=cross_val_predict(sgd_clf, X_train_scaled, y_train, cv=3)
conf_mx = confusion_matrix(y_train, y_train_pred)
#along the diagonals it predicted right and rest are wrongs
print("conf_mx:",conf_mx)

plt.matshow(conf_mx, cmap=plt.cm.gray)
fl1.save_fig("confusion_matrix_plot", tight_layout=False)

row_sums = conf_mx.sum(axis=1, keepdims=True)
#sum of all classes by classes
print("row_sums:",row_sums)
norm_conf_mx = conf_mx / row_sums
#after div we get the error rates.
print("norm_conf_mx:",norm_conf_mx)
np.fill_diagonal(norm_conf_mx, 0)
plt.matshow(norm_conf_mx, cmap=plt.cm.gray)
fl1.save_fig("confusion_matrix_errors_plot", tight_layout=False)
plt.show()
```

- looks good, but represents correct classifications along the main diagonal.
- 5 looks a shade darker, which could mean
  1. 5s are less in number
  2. Classifier does not perform well on 5s.

```
conf_mx:[[5737    2   21   11   11   41   49    8   37    6]
 [   1 6474   50   28    6   37    8   11  116   11]
 [  63   40 5326   99   86   19   85   67  159   14]
 [  48   40  127 5366    1  224   36   55  126  108]
 [  20   24   42    8 5356    8   50   24   86  224]
 [  72   46   37  198   71 4586  110   27  179   95]
 [  35   29   55    2   49   88 5600    9   51    0]
 [  24   19   71   32   60    8    5 5784   15  247]
 [  53  152   73  156   13  154   54   30 5022  144]
 [  44   28   26   79  153   33    2  211   84 5289]]
```



- Error rates rather than total errors would be more indicative.

Error based learning:Multiclass Classification:Confusion Matrix

```
conf_mx:[[5737    2   21   11   11   41   49    8   37    6]
 [   1 6474   50   28    6   37    8   11  116   11]
 [  63   40 5326   99   86   19   85   67  159   14]
 [  48   40  127 5366    1  224   36   55  126  108]
 [  20   24   42    8 5356    8   50   24   86  224]
 [  72   46   37  198   71 4586  110   27  179   95]
 [  35   29   55    2   49   88 5600    9   51    0]
 [  24   19   71   32   60    8    5 5784   15  247]
 [  53  152   73  156   13  154   54   30 5022  144]
 [  44   28   26   79  153   33    2  211   84 5289]]
```

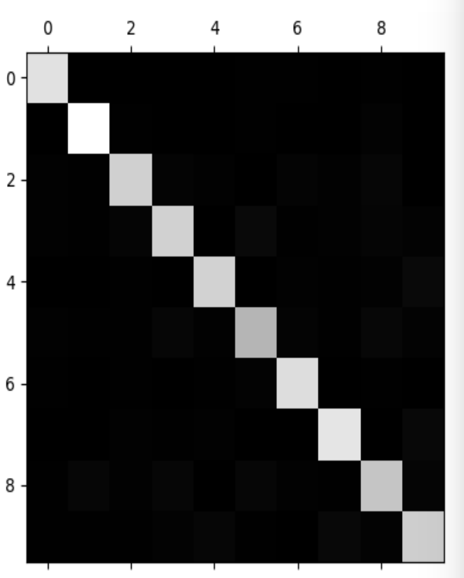
```
rows_sum:[[5923]
 [6742]
 [5958]
 [6131]
 [5842]
 [5421]
 [5918]
 [6265]
 [5851]
 [5949]]
```

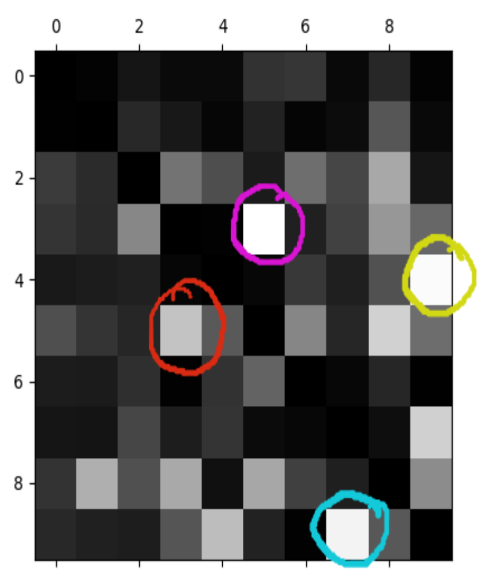
sum along the axis=1

```
norm_conf_mx:
[[ 9.68596995e-01 3.37666723e-04 3.54550059e-03 1.85716698e-03 1.85716698e-03 6.92216782e-03 8.27283471e-03 1.35066689e-03  6.24683437e-03 1.01300017e-03]
 [ 1.48323939e-04 9.60249184e-01 7.41619697e-03 4.15307031e-03 8.89943637e-04 5.48798576e-03 1.18659152e-03 1.63156333e-03  1.72055770e-02 1.63156333e-03]
 [ 1.05740181e-02 6.71366230e-03 8.93924136e-01 1.66163142e-02 1.44343740e-02 3.18898959e-03 1.42665324e-02 1.12453844e-02  2.66868077e-02 2.34978181e-03]
 [ 7.82906541e-03 6.52422117e-03 2.07144022e-02 8.75224270e-01 1.63105529e-04 3.65356386e-02 5.87179905e-03 8.97080411e-03  2.05512967e-02 1.76153972e-02]
 [ 3.42348511e-03 4.10818213e-03 7.18931873e-03 1.36939404e-03 9.16809312e-01 1.36939404e-03 8.55871277e-03 4.10818213e-03  1.47209860e-02 3.83430332e-02]
 [ 1.32816823e-02 8.48551928e-03 6.82530898e-03 3.65246265e-02 1.30972145e-02 8.45969378e-01 2.02914591e-02 4.98063088e-03  3.30197381e-02 1.75244420e-02]
 [ 5.91416019e-03 4.90030416e-03 9.29368030e-03 3.37952011e-04 8.27982426e-03 1.48698885e-02 9.46265630e-01 1.52078405e-03  8.61777628e-03 0.00000000e+00]
 [ 3.83080607e-03 3.03272147e-03 1.13328013e-02 5.10774142e-03 9.57701516e-03 1.27693536e-03 7.98084597e-04 9.23224262e-01  2.39425379e-03 3.94253791e-02]
 [ 9.05828064e-03 2.59784652e-02 1.24764997e-02 2.66621090e-02 2.22184242e-03 2.63202871e-02 9.22919159e-03 5.12732866e-03  8.58314818e-01 2.46111776e-02]
 [ 7.39620104e-03 4.70667339e-03 4.37048243e-03 1.32795428e-02 2.57186082e-02 5.54715078e-03 3.36190956e-04 3.54681459e-02  1.41200202e-02 8.89056984e-01]]
```

rate

eg. (5737/5923)

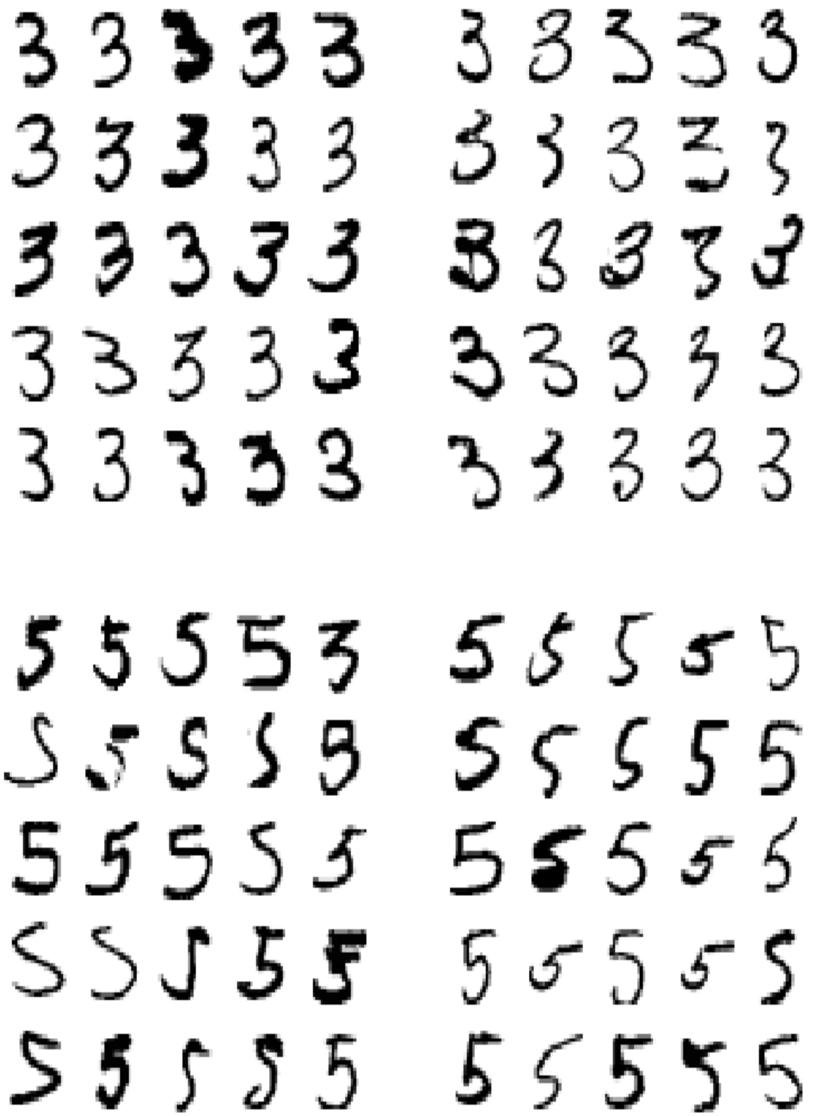




- 3 confused with 5
- 5 " " 3
- 9 " " 7
- 4 " " 9

Error based learning:Multiclass Classification:Confusion Matrix

```
X, y = mnist["data"], mnist["target"]
X_train, X_test, y_train, y_test = X[:60000], X[60000:], y[:60000], y[60000:]

shuffle_index = rnd.permutation(60000)
X_train, y_train = X_train[shuffle_index], y_train[shuffle_index]

#random forest can directly classify multiple classes
sgd_clf = SGDClassifier(random_state=42)
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train.astype(np.float64))
y_train_pred=cross_val_predict(sgd_clf, X_train_scaled, y_train, cv=3)

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)
```



Error based learning:Multilabel classification

```
X, y = mnist["data"], mnist["target"]
X_train, X_test, y_train, y_test = X[:60000], X[60000:], y[:60000], y[60000:]

shuffle_index = rnd.permutation(60000)
X_train, y_train = X_train[shuffle_index], y_train[shuffle_index]


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

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

print("KNN:",knn_clf)

some_digit_index = 36000
some_digit_image = X_train[some_digit_index]

print("predict:",knn_clf.predict([some_digit_image]))
print("label:",y_train[some_digit_index])
plot(some_digit_image)
```

- Trained on multiple conditions and a multilable output.

```
(ml_home) mohit@nomind:~/Work/ArtificialIntelligence$ ./ML/classification/multilabelclassification.py
mnist data already there
KNN: KNeighborsClassifier(algorithm='auto', leaf_size=30, metric='minkowski',
           metric_params=None, n_jobs=1, n_neighbors=5, p=2,
           weights='uniform')
predict: [[False False]]
label: 6.0
```

Error based learning:Multilabel classification:Cross Validation

```
X, y = mnist["data"], mnist["target"]
X_train, X_test, y_train, y_test = X[:60000], X[60000:], y[:60000], y[60000:]

shuffle_index = rnd.permutation(60000)
X_train, y_train = X_train[shuffle_index], y_train[shuffle_index]

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

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

y_train_knn_pred = cross_val_predict(knn_clf, X_train, y_train, cv=3)
print("y_train_knn_pred:",y_train_knn_pred)
print("f1_score:",f1_score(y_train, y_train_knn_pred, average="macro"))
```

Error based learning:Multioutputclassification

```
X, y = mnist["data"], mnist["target"]
X_train, X_test, y_train, y_test = X[:60000], X[60000:], y[:60000], y[60000:]

shuffle_index = rnd.permutation(60000)
X_train, y_train = X_train[shuffle_index], y_train[shuffle_index]

noise = rnd.randint(0, 100, (len(X_train), 784))
X_train_mod = X_train + noise
noise = rnd.randint(0, 100, (len(X_test), 784))
X_test_mod = X_test + noise
y_train_mod = X_train
y_test_mod = X_test

some_index = 5500
plt.subplot(121); plot(X_test_mod[some_index])
plt.subplot(122); plot(y_test_mod[some_index])
#fl1.save_fig("noisy_digit_example_plot")
#plt.show()

knn_clf = KNeighborsClassifier()
knn_clf.fit(X_train_mod, y_train_mod)

print("knn classifier:",knn_clf)
clean_digit = knn_clf.predict([X_test_mod[some_index]])
plot(clean_digit)
```