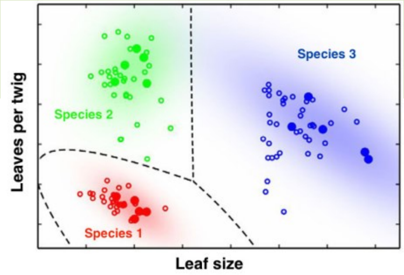
UCCD2063

Artificial Intelligence Techniques

Unit 04:

The Classification Pipeline



Outline

1. Look at the big picture
2. Get, explore and prepare data
3. Select and train model
    1) Binary Classification
        i. Accuracy
        ii. Confusion Matrix
        iii. Precision, Recall and F1 score
        iv. Precision vs Recall (PR) Curve
        v. Receiver Operating Curve (ROC)
        vi. Area Under Curve (AUC)
    2) Multi-class classification
4. Fine-tune model

Python Libraries & Tools

- To learn the classification pipeline, we will be using the following Python libraries and tools:
  - Numpy, Pandas, Matplotlib, Scikit-Learn
  - Jupyter Notebook

- Dataset needed for the hand-on exercise:
  - MNIST dataset

Will practice the classification pipeline in Lab 6

Dataset used in this lecture

MNIST Digit

- A set of 70,000 small images of digits written by high school students and employees of US Census Bureau
- First **60,000** samples for training, remaining **10,000** for testing
- All images are 28x28 pixels in size, gray scale with values of 0 ~ 255
- Comes with Scikit-Learn
- **Task**: given an image, classify what digit is in the image

Look at the Big Picture

What do we want to do?

- **Binary Classification**: Detect images with **digit 5** in the dataset. Classify images with digit 5 as **true** and the others as **false**.

- **Multi-class Classification**: Detect images with digit 0 to 9 in the dataset.

Get data

Fetch dataset and look at data structure

- MNIST dataset is provided by Scikit-Learn

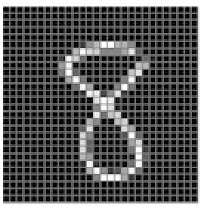
```
from sklearn.datasets import fetch_openml
X, y = fetch_openml('mnist_784', version=1, return_X_y=True)
```

```
print('Shape of X:', X.shape)
print('Shape of y:', y.shape)
```

```
Shape of X: (70000, 784)
Shape of y: (70000,)
```

- Total 70,000 samples
- Each sample has **784 attributes** (pixels) which are acquired by flattening the image (784 = 28 x 28).
- Each pixel has a value between 0 (black) and 255 (white)
- X stores the images in rows as float
- y stores the labels of corresponding images as pandas ‘CategoricalDtype’

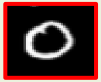
Image Representation



28 x 28
784 pixels

0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	4	62	146	182	254	254	181	176	139	15	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	34	186	253	217	208	136	136	136	166	232	99	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	61	242	208	111	3	0	0	0	0	0	18	32	107	43	0	0	0	0	0	0
0	0	0	0	0	0	0	0	156	242	23	0	0	0	0	0	0	0	13	191	181	6	0	0	0	0	0	0
0	0	0	0	0	0	0	0	121	255	98	3	0	0	0	0	0	8	194	225	12	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	169	253	120	3	0	0	0	0	128	247	51	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	3	111	244	169	19	0	14	131	249	117	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	59	241	235	72	142	229	66	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	25	218	254	231	36	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	133	253	221	33	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	19	237	111	196	217	19	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	174	138	0	23	193	204	18	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	96	224	0	0	0	25	218	169	3	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	215	138	0	0	0	0	86	253	99	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	215	97	0	0	0	0	3	162	214	11	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	215	97	0	0	0	0	0	118	253	68	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	185	157	0	0	0	0	0	40	254	98	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	50	244	61	0	0	0	0	112	244	58	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	174	251	142	59	83	167	244	111	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	6	133	253	253	253	169	61	3	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Explore data

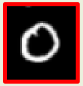


- Looking at the distribution of digits, it is quite even

```
print('label  frequency')
print(pd.Series(y).value_counts())
```

```
label  frequency
1     7877
7     7293
3     7141
2     6990
9     6958
0     6903
6     6876
8     6825
4     6824
5     6313
dtype: int64
```

Prepare data



- First, define the input matrix **X** and output variable *y*
- Then, split the dataset into training and testing set

```
X_train, X_test, y_train, y_test = X[:60000], X[60000:], y[:60000], y[60000:]

print('Shape of X_train:', X_train.shape, ' y_train:', y_train.shape)
print('Shape of X_test:', X_test.shape, ' y_test:', y_test.shape)
```

```
Shape of X_train: (60000, 784)  y_train: (60000,)
Shape of X_test: (10000, 784)  y_test: (10000,)
```

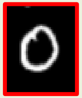
```
print('Labels of first 10 training samples:', y_train[:10])  # show the first 10 samples
```

```
Labels of first 10 training samples: ['5' '0' '4' '1' '9' '2' '1' '3' '1' '4']
```

Outline

1. Look at the big picture
2. Get, explore and prepare data
3. Select and train model
    1) Binary Classification
        i. Accuracy
        ii. Confusion Matrix
        iii. Precision, Recall and F1 score
        iv. Precision vs Recall (PR) Curve
        v. Receiver Operating Curve (ROC)
        vi. Area Under Curve (AUC)
    2) Multi-class classification
4. Fine-tune model

Binary Classification



- **Binary classifier**: distinguish between two classes, **5** (true) and **not-5** (false) images

- First, let's create the targeted variable y_train_5 and y_test_5 which is *True* only for samples with digit 5 and *False* otherwise

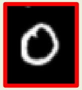
```
y_train_5 = (y_train == '5')
y_test_5  = (y_test == '5')
```

y_train
1
6
8
5
2

y_train_5
False
False
False
True
False

Select and Train a Model

Training a binary classifier: `SGDClassifier`



- Train the `SGDClassifier` classifier:

```
from sklearn.linear_model import SGDClassifier

sgd_clf = SGDClassifier (random_state = 42, max_iter = 5, tol = None)
sgd_clf.fit(X_train, y_train_5)
```

- After we have trained it, we can now use it to classify 5 / not-5 images

```
y_pred = sgd_clf.predict(X_train)
y_pred
```

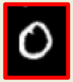
```
array([False, False, False, ..., False, False, False], dtype=bool)
```

`SGDClassifier` is a **linear classifier** that uses stochastic gradient descent (SGD) which which has the advantage of being capable of handling very large datasets efficiently.

Classification Accuracy

- One way to measure the performance of a classifier is **accuracy** measure:

$$\text{Accuracy} = \frac{\#\text{Correctly predicted samples}}{\#\text{samples}}$$



- Evaluating accuracy of predicted labels

```python
from sklearn.metrics import accuracy_score

train_acc = accuracy_score(y_train_5, y_pred)

print("Training accuracy: {:.4f}".format(train_acc))
```

```
Training accuracy: 0.9648
```

accuracy_score `helper function` computes the accuracy score given the actual and predicted labels

Evaluation using Cross-Validation

- A good way to evaluate a model is to use the K-fold cross-validation:

3-fold cross-validation

```
from sklearn.model_selection import cross_val_score

k_scores = cross_val_score(sgd_clf, X_train, y_train_5, cv=3, scoring='accuracy')
k_scores
```

```
array([0.9502 , 0.96565, 0.96495])
```

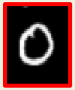
Performance on all folds are above 0.95. This looks very amazing.
But is it really so?

`cross_validate` **allows specifying multiple metrics for evaluation**

Issues with Accuracy

- Accuracy measure may not give a true idea about the performance of the system especially for skewed dataset

Problem with accuracy for skewed dataset



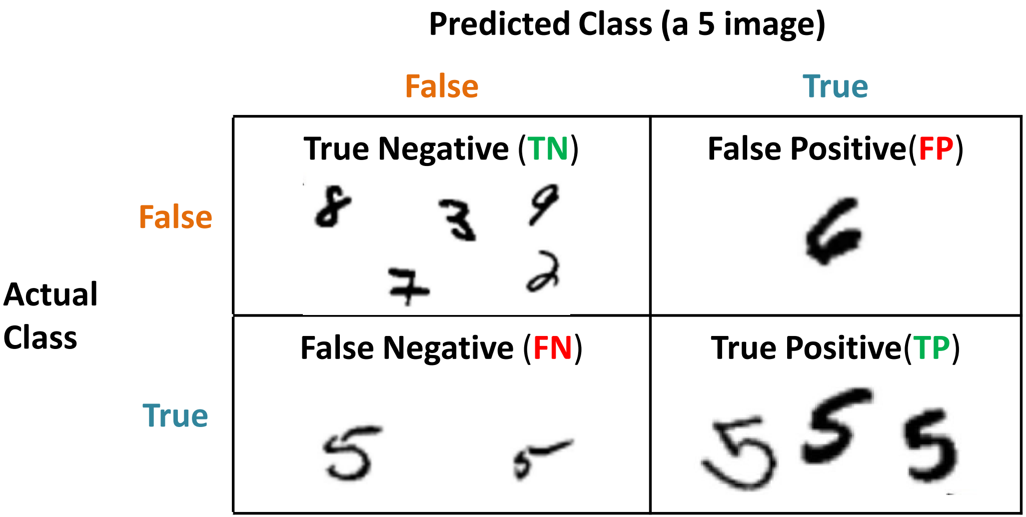
```
from sklearn.metrics import accuracy_score

y_train_pred = sgd_clf.predict(X_train)
print('Accuracy using prediction values:', accuracy_score(y_train_5, y_train_pred))

y_train_allfalse = np.zeros(len(y_train_5), dtype=bool)
print('Accuracy if set to all false:', accuracy_score(y_train_5, y_train_allfalse))
```

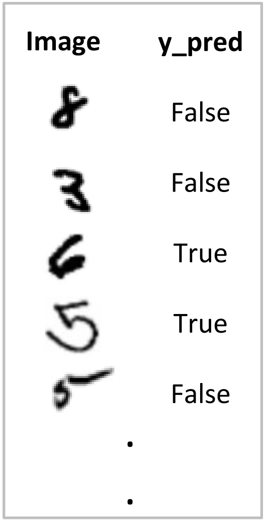
```
Accuracy using prediction values: 0.9648
Accuracy if set to all false: 0.90965
```

- Although we predict all images with digit 5 wrongly, we still get a high accuracy value of over 90%
- This is because only 10% of the images are 5

Confusion matrix

- A better way to present our results is the **confusion matrix**



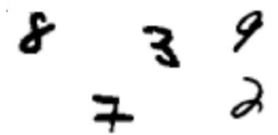
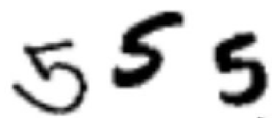


TP (true positives) : number of positive samples that are correctly predicted as positive
FN (false negatives) : number of positive samples that are falsely predicted as negative
TN (true negatives) : number of negative samples that are correctly predicted as negative
FP (false positives) : number of negative samples that are falsely predicted as positive

Confusion matrix

- A better way to present our results is the **confusion matrix**

Predicted Class (a 5 image)

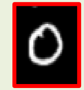
		False	**True**
Actual Class	**False**	**True Negative (TN)**  (5)	**False Positive(FP)**  (1)
	True	**False Negative (FN)**  (2)	**True Positive(TP)**  (3)

- We can derive accuracy from the confusion matrix

$$\text{Accuracy} = (TN + TP) / (TN + TP + FP + FN)$$

Accuracy = 8 / 11

Confusion Matrix



- Compute the cross-validated prediction of all samples in training set

```
from sklearn.model_selection import cross_val_predict

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

```
array([False, False, False, ..., False, False, False], dtype=bool)
```

- Compute the confusion matrix

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

Prediction

Actual	Prediction: False	Prediction: True
False	53272	1307
True	1077	4344

```
[[53272  1307]
 [ 1077  4344]]
```

Question: What is the confusion matrix for the best possible performance?

[[54579, 0]
[0, 5421]

cross_val_predict generates the cross-validated estimates for each sample
confusion_matrix computes the confusion matrix given the actual and predicted values

Precision and Recall

- **Precision** or **positive predictive value** (**PPV**) measures the accuracy of the positive prediction:

  If the system predicts something to be true, how reliable is its prediction?

  Precision = TP / (FP + TP)

		Predicted Class	
		No	Yes
Actual Class	No	True Negative (TN)	False Positive(FP)
	Yes	False Negative (FN)	True Positive(TP)

Pre

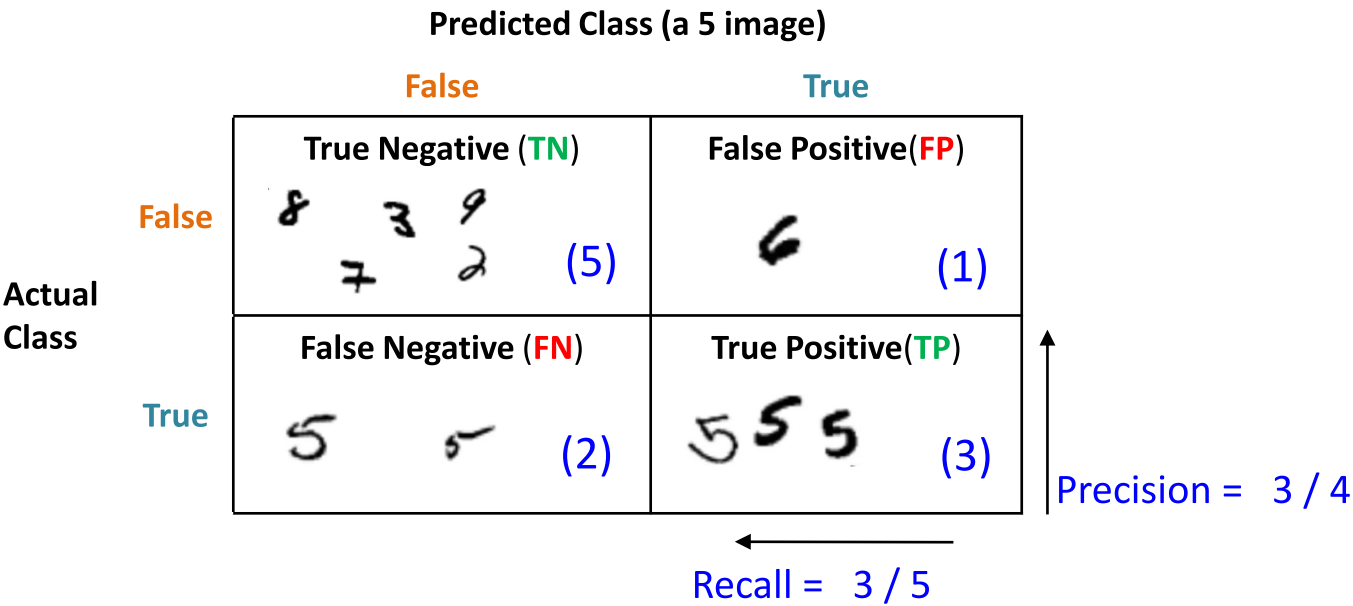
Recall

- **Recall** or **sensitivity** or **true positive rate** (**TPR**) measures the ratio of positive samples that are correctly detected

  How many positive samples in the dataset are correctly detected?

  Recall = TP / (FN + TP)

Confusion matrix



The Precision/Recall trade-off: increase precision usually will lower the recall rate and vice versa

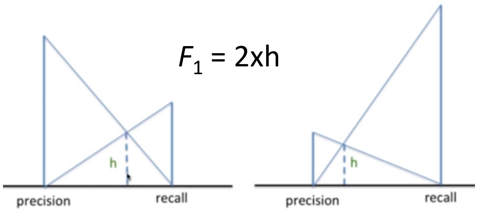
F_1 Score

- Combines the precision and recall into a single score
- The *harmonic mean* of precision and recall:

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

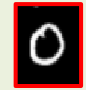
Examples:

Precision	Recall	Mean	F1-score
0.9	0.3	0.6	0.45
0.7	0.5	0.6	0.58



- F_1 is high only when both recall and precision are high (harmonic mean gives much more weight to the low values)
- F_1 favors classifiers that have similar precision and recall
- Used when both precision and recall are equally important

Precision, Recall and F1 Score



```
from sklearn.metrics import precision_score, recall_score, f1_score

print('precision = ', precision_score(y_train_5, y_train_pred))
print('recall    = ', recall_score (y_train_5, y_train_pred))
print('f1 score  = ', f1_score(y_train_5, y_train_pred))
```

```
precision =  0.768713502035
recall    =  0.801328168235
f1 score  =  0.784682080925
```

- The 5-detector does not look as good now (correct only 77% of the time and only detect 80% of the 5s).

precision_score computes the precision given the actual and predicted values
recall_score computes the recall given the actual and predicted values
f1_score computes the f1 score given the actual and predicted values

Review question

- For a classifier that detect safe video contents for the kids, which performance measure is more important: precision or recall?

  Want high precision (all selected videos are actually safe), able to tolerate low recall (some good videos are filtered out)

Predicted Class

		False	True
Actual Class	False	**True Negative (TN)** ~~Safe~~ ~~Safe~~ ~~Safe~~	**False Positive(FP)** 0
	True	**False Negative (FN)** Safe Safe	**True Positive(TP)** Safe Safe Safe Safe

Precision = 4 / 4

Review question

- For a classifier that detect shoplifters, which performance measure is more important: precision or recall?

Want high recall (detect potential shoplifters). Low precision (detect non-shoplifters) can be screened manually. Want to catch all shoplifters.

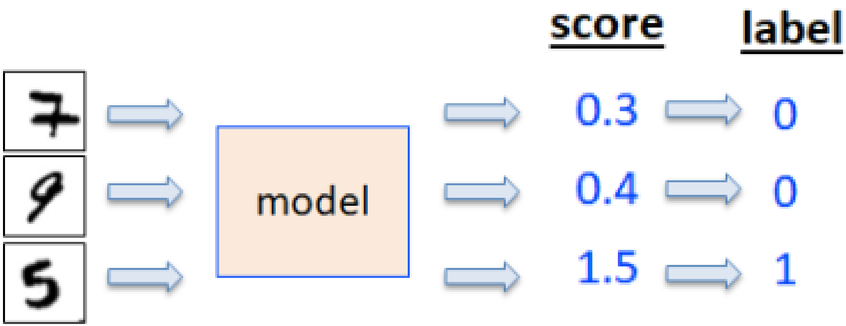
Predicted Class

		False	**True**
Actual Class	**False**	**True Negative (TN)**  	**False Positive(FP)** 
	True	**False Negative (FN)** 0	**True Positive(TP)**   

Recall = 3 / 3

Precision/Recall Tradeoff

- Desirable to have high precision and high recall
- In reality, increasing precision reduces recall, and vice versa.
- Most classifiers compute a **score** (how likely a sample belongs to the positive class) based on a **decision function**, and then threshold on the score to output the label.



Example:
Threshold = 1
score ≥ 1 = 1
score < 1 = 0

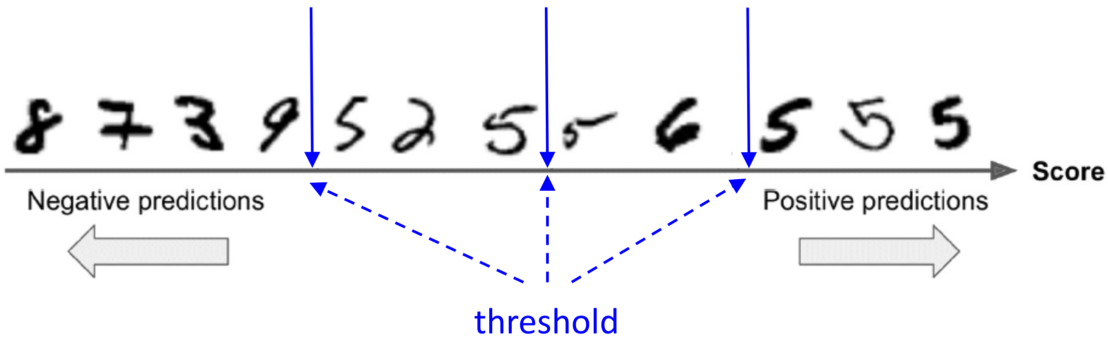
- Thus, we can set some **threshold** on the score to control the tradeoff between precision and recall.

Note that in SGDClassifier, the threshold value is fixed at 0

Precision/Recall Tradeoff

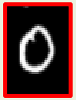
- The score **threshold** value controls precision and recall
- As we lower the threshold, recall increases but precision decreases (less precise but get more positive samples), and vice versa

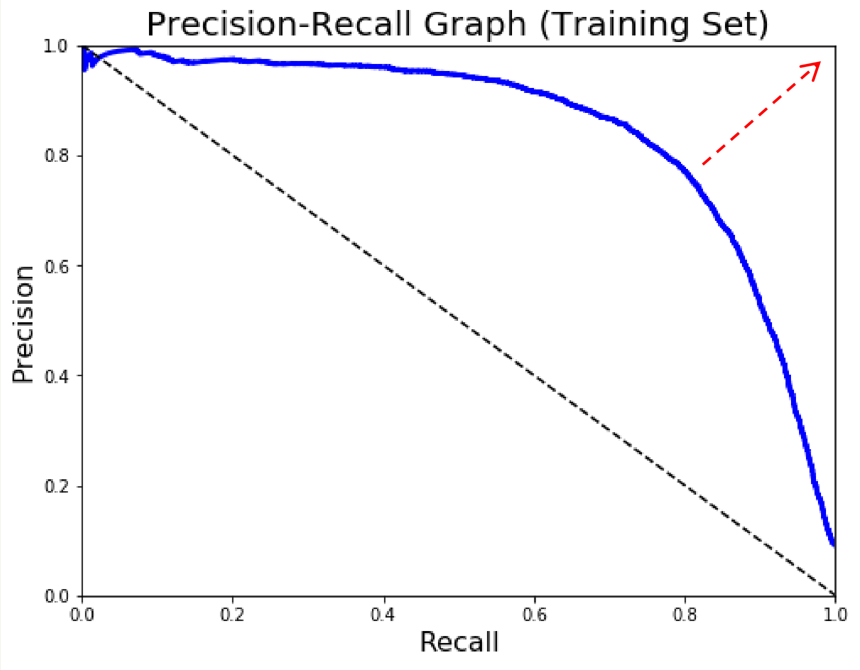
Precision:	6/8 = 75%	4/5 = 80%	3/3 = 100%
Recall:	6/6 = 100%	4/6 = 67%	3/6 = 50%



- How to decide which threshold to use?
  - Plot the **precision-recall graph** and choose manually

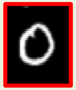
Precision-Recall Curve





- P-R Curve shows the relationship of precision and recall when changing threshold values
- A good classifier should have a curve that is close to the top-right corner.

Precision and Recall scores for different threshold values



- We can retrieve the average cross-validated scores for all samples through the function cross_val_predict() with the parameter method="decision_fuction"

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

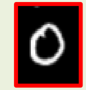
```
array([ -434076.49813641, -1825667.15281624,  -767086.76186905, ...,
        -867191.25267994,  -565357.11420164,  -366599.16018198])
```

- Next, compute the precision and recall scores at threshold values using precision_recall_curve()

```
from sklearn.metrics import precision_recall_curve

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

precision_recall_curve computes the precision and recall for different probability thresholds. Note that this implementation is restricted to binary classification task

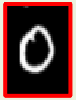
Plot Precision-Recall graph

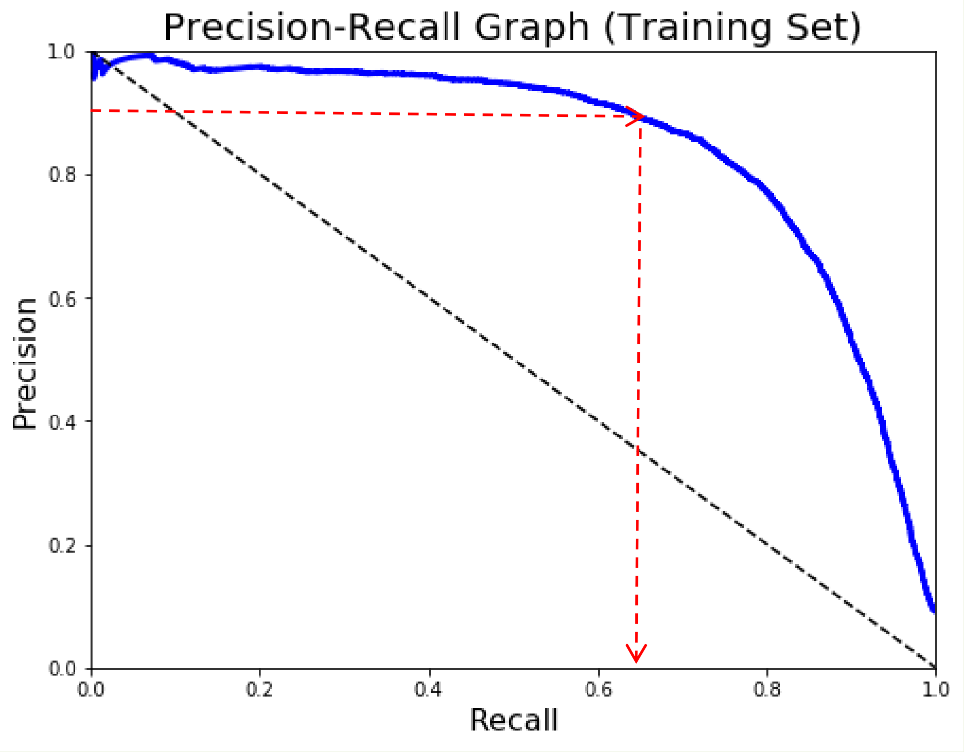


- Using the precision and recall values at different thresholds, we can now plot our precision-recall graph.

```
def plot_precision_vs_recall(precisions, recalls):
    plt.plot(recalls, precisions, "b-", linewidth=3)
    plt.plot(np.linspace(0, 1, 20), np.linspace(1, 0, 20), 'k--')
    plt.xlabel("Recall", fontsize=16)
    plt.ylabel("Precision", fontsize=16)
    plt.axis([0, 1, 0, 1])

plt.figure(figsize=(8, 6))
plot_precision_vs_recall(precisions, recalls)
plt.title ('Precision-Recall Graph (Training Set)', fontsize = 20)
plt.show()
```

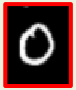
Precision-Recall Curve





When,
Precision = 0.9,
Recall = 0.64

Getting the desired threshold value



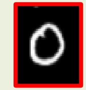
- Suppose we desire a higher precision (0.9) and can tolerate a lower recall (between 0.6 and 0.7). The following code shows how to retrieve the desired threshold value when the precision is at least 0.9.

```
idx = np.argmax(precisions >= 0.9)   # the position of the first precision value that is >= 0.9
selected_threshold = thresholds[idx]

print('selected threshold = {:.2f}'.format(selected_threshold))
print('precision at selected threshold = {:.2f}'.format(precisions[idx]))
print('recall at selected threshold = {:.2f}'.format(recalls[idx]))
```

```
selected threshold = 103561.39
precision at selected threshold = 0.90
recall at selected threshold = 0.64
```

When threshold =103561,
we can achieve 0.9 precision and
0.64 recall rate
(compare to default threshold=0,
precision=0.768, recall=0.801)

Prediction with manually selected threshold value



- Recall that in SGDClassifier, the threshold value is fixed at 0
- We can manually use the selected threshold value to perform prediction.

```
# evaluate on the first 200 training samples
samples = X_train[:200]
samples_label = y_train_5[:200]

# use our model to compute the scores on selected samples
scores = sgd_clf.decision_function(samples)

# perform prediction
predictions = (scores > selected_threshold)
```

```
# compute precision and recall
print ('precision =', precision_score (samples_label, predictions))
print ('recall =', recall_score (samples_label, predictions))
```

```
precision = 0.9090909090909091
recall = 0.6666666666666666
```

SGDClassifier.decision_function() returns the predicted scores of samples computed by SGDClassifier

Receiver Operating Characteristics (ROC)

- Another common tool to evaluate binary classifier is the Receiver Operating Characteristic (ROC) curve
- ROC curve plots the True Positive Rate (TPR or recall) vs False Positive Rate (FPR)

TPR = ratio of the positive instances that are correctly predicted positive

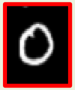
= TP / (TP + FN)

FPR = ratio of negative instances that are incorrectly classified as positive

= FP / (TN + FP)

		Predicted Class: No	Predicted Class: Yes
Actual Class	No	True Negative (TN)	False Positive(FP)
	Yes	False Negative (FN)	True Positive(TP)

Plot ROC curve

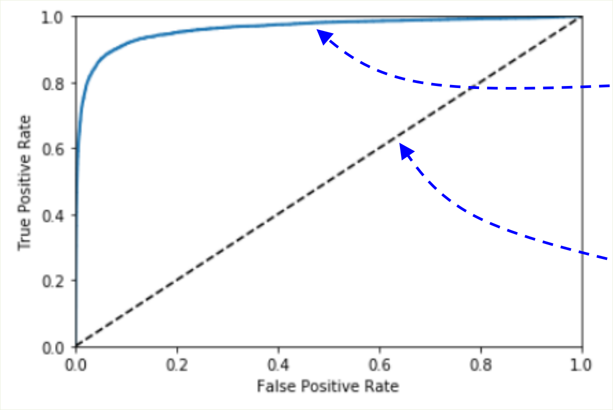


```
from sklearn.metrics import roc_curve
fpr, tpr, thresholds = roc_curve(y_train_5, y_scores_cv)
```

Computes the FPR and TPR for different threshold values

```
def plot_roc_curve (fpr, tpr, label = None):
    plt.plot(fpr, tpr, linewidth = 2, label = label)
    plt.plot([0,1], [0, 1], 'k--')
    plt.axis([0, 1, 0, 1])
    plt.xlabel ('False Positive Rate')
    plt.ylabel ('True Positive Rate')

plot_roc_curve(fpr, tpr)
```

Plots the ROC curve



A good classifier should have a curve that is close to the top-left corner

The ROC curve of a purely random classifier

roc_curve() computes the FPR and FPR for different threshold values

Area Under Curve (AUC)

- AUC converts the ROC curve into a quantitative value
- A perfect AUC has a value of 1
- A purely random classifier has a AUC value of 0.5

Compute the AUC

```
from sklearn.metrics import roc_auc_score

auc = roc_auc_score(y_train_5, y_scores_cv)

print('AUC = {:.4f}'.format(auc))
```

```
AUC = 0.9624
```

roc_auc_score() computes the AUC score given the predicted and actual value

Evaluate Model on Test Set

- The final model should be evaluated on unseen data (test set) before deployment.

```
from sklearn.linear_model import SGDClassifier
from sklearn.metrics import precision_score, recall_score, f1_score

sgd_clf = SGDClassifier(random_state = 42, max_iter = 5, tol = None)
sgd_clf.fit(X_train, y_train_5)
y_pred_test = sgd_clf.predict(X_test) # fit on test set

print('Test precision = {:.4f}'.format(precision_score(y_test_5, y_pred_test)))
print('Test recall    = {:.4f}'.format(recall_score (y_test_5, y_pred_test)))
print('Test f1 score  = {:.4f}'.format(f1_score(y_test_5, y_pred_test)))
```

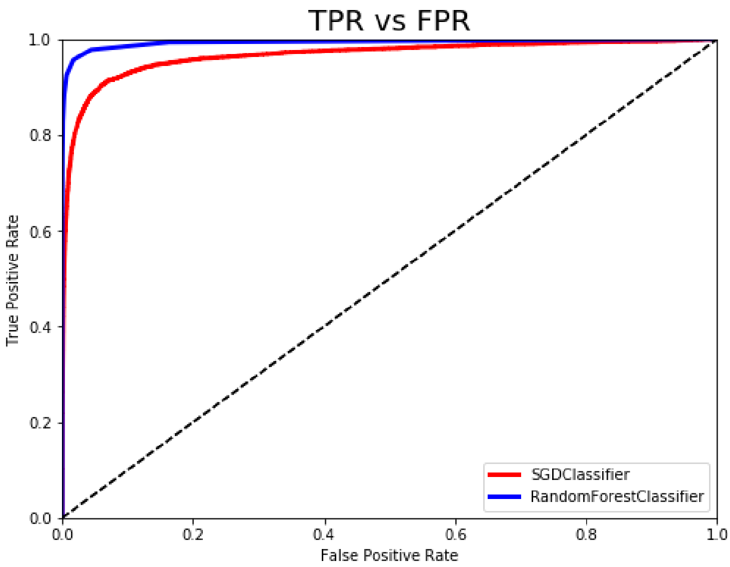
```
Test precision = 0.8188
Test recall    = 0.8105
Test f1 score  = 0.8146
```

The test performance results are close to the training results, indicating that the model is not overfitted.

Comparing classifiers using ROC curve

- We can use ROC curve and AUC measure to compare the performance of different classification models.

- Example:



For this task, RandomForestClassifier should perform better than SGDClassifier

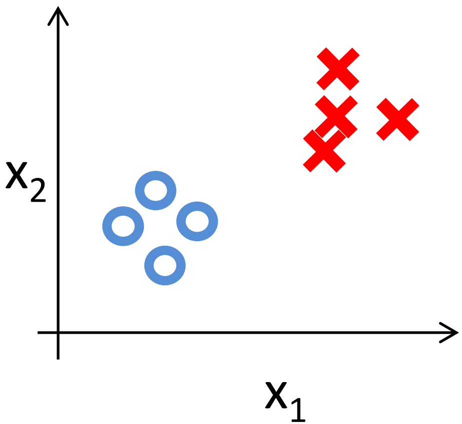
Outline

1. Look at the big picture
2. Get, explore and prepare data
3. Select and train model
    1) Binary Classification
        i. Accuracy
        ii. Confusion Matrix
        iii. Precision, Recall and F1 score
        iv. Precision vs Recall (PR) Curve
        v. Receiver Operating Curve (ROC)
        vi. Area Under Curve (AUC)
    2) Multi-class classification
4. Fine-tune model

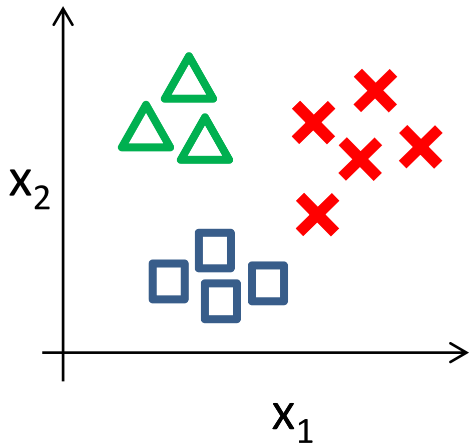
Multi-class Classification

What is a multi-class classification?

Binary classification:



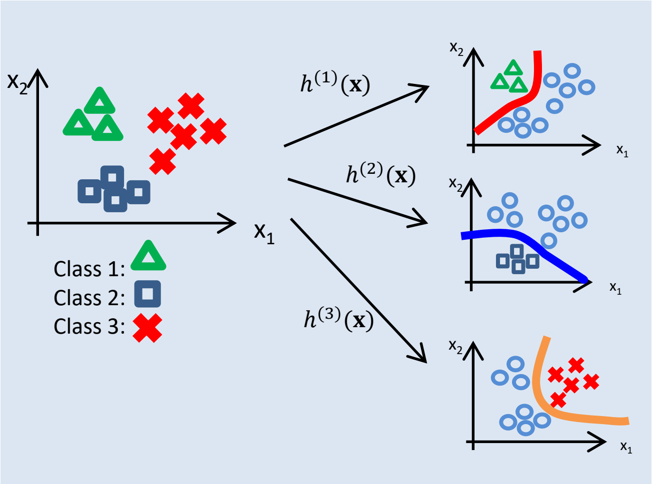
Multi-class classification:



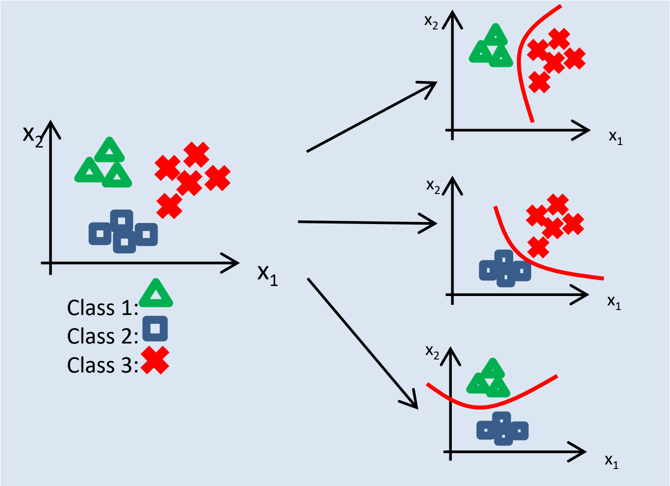
- Digit classification: *0, 1, 2, ..., 9*
- Vehicle classification: *car, bus, truck, ...*

Multi-class Classification

- Some classifiers (e.g. Random Forest) can naturally handle multi-class classification.
- For binary classifiers (e.g. SGDClassifier , SVM), multi-class classification is supported through One-Versus-One (OVO) or One-Versus-All (OVA) schemes.

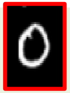


OVA (N classifiers)



OVO (N(N-1)/2 classifiers)

Multi-class classification for SGDClassifier



- In Scikit-learn, by default all binary classifier uses OVA except for SVM which uses OVO
- `SGDClassifier` is a binary classifier. When it detects that targeted output has three or more classes, it will automatically runs OVA

```
from sklearn.linear_model import SGDClassifier

sgd_clf = SGDClassifier (random_state = 42, max_iter = 5, tol = None)
sgd_clf.fit(X_train, y_train)
print(sgd_clf.classes_) # show class labels
```

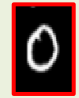
```
[0. 1. 2. 3. 4. 5. 6. 7. 8. 9.]
```

The syntax remains unchanged!

- Measuring training accuracy

```
from sklearn.metrics import accuracy_score

y_pred = sgd_clf.predict(X_train)
train_acc = accuracy_score(y_train, y_pred)
print("Training accuracy: {:.4f}".format(train_acc))
```

```
Training accuracy: 0.8639
```



- Measuring training accuracy using Cross-Validation

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

```
array([0.84063187, 0.84899245, 0.86652998])
```

- Achieve around 85% accuracy on all test folds
- If we scale the dataset, we can improve the performance to around 90%.
- If we use *RandomForestClassifier*, we can get around 94%

Performance Measure on Multiclass Classification

Multiclass Confusion Matrix

Model Predictions

Actual Classes

	A	**B**	**C**	**D**
A	9	1	0	0
B	1	15	3	1
C	5	0	24	1
D	0	4	1	15

Total number of samples = 80

Performance Metric for Class A

Model Predictions

Actual Classes

	A	B	C	D
A	9	1	0	0
B	1	15	3	1
C	5	0	24	1
D	0	4	1	15

True Positive (TP) = 9

False Positive (FP) = 6

True Negative (TN) = 64

False Negative (FN) = 1

Performance Metric for Class B

Model Predictions

Actual Classes

	A	B	C	D
A	9	1	0	0
B	1	15	3	1
C	5	0	24	1
D	0	4	1	15

True Positive (TP) = 15

False Positive (FP) = 5

True Negative (TN) = 55

False Negative (FN) = 5

Performance Metric for Class C

Model Predictions (columns) / **Actual Classes** (rows)

	A	B	C	D
A	9	1	0	0
B	1	15	3	1
C	5	0	24	1
D	0	4	1	15

True Positive (TP) = 24

False Positive (FP) = 4

True Negative (TN) = 46

False Negative (FN) = 6

Performance Metric for Class D

Model Predictions

Actual Classes

	A	B	C	D
A	9	1	0	0
B	1	15	3	1
C	5	0	24	1
D	0	4	1	15

True Positive (TP) = 15

False Positive (FP) = 2

True Negative (TN) = 58

False Negative (FN) = 5

Accuracy

Model Predictions

Actual Classes

	A	B	C	D
A	9	1	0	0
B	1	15	3	1
C	5	0	24	1
D	0	4	1	15

Accuracy = (9 + 15 + 24 + 15) / 80 = 0.7875

Precision (P) = TP / (TP+FP)

Model Predictions

Actual Classes

	A	B	C	D
A	9	1	0	0
B	1	15	3	1
C	5	0	24	1
D	0	4	1	15

TP: 9 FP: 6	TP: 15 FP: 5	TP: 24 FP: 4	TP: 15 FP: 2

P(A) = 9/15 P(B) = 15/20 P(C) = 24/28 P(D) = 15/17

Average (macro) Precision = (P(A)+P(B)+P(C)+P(D)) / 4 = 0.7724

Recall (R) = TP / (TP+FN)

Model Predictions

Actual Classes

	A	B	C	D	
A	9	1	0	0	TP: 9 FN: 1
B	1	15	3	1	TP: 15 FN: 5
C	5	0	24	1	TP: 24 FN: 6
D	0	4	1	15	TP: 15 FN: 5

R(A) = 9/10 R(B) = 15/20 R(C) = 24/30 R(D) = 15/20

Average (macro) Recall = (R(A)+R(B)+R(C)RP(D)) / 4 = 0.8

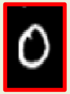
Outline

1. Look at the big picture
2. Get, explore and prepare data
3. Select and train model
    1) Binary Classification
        i. Accuracy
        ii. Confusion Matrix
        iii. Precision, Recall and F1 score
        iv. Precision vs Recall (PR) Curve
        v. Receiver Operating Curve (ROC)
        vi. Area Under Curve (AUC)
    2) Multi-class classification
4. Fine-tune model

Fine-tuning model

- Fine-tune model hyperparamters using grid search or random search
- Improve the model by analyzing the types of errors it makes. Strategy:
  - Plot the confusion matrix
  - Check the difficult cases.

Plot the confusion matrix



```
from sklearn.model_selection import cross_val_predict
from sklearn.metrics import confusion_matrix

y_pred_cv = cross_val_predict (sgd_clf, X_train, y_train, cv=3)
cm = confusion_matrix (y_train, y_pred_cv)
print(cm)
```

	0	1	2	3	4	5	6	7	8	9
0	5604	2	81	25	8	44	32	9	115	3
1	1	6292	72	54	20	22	8	8	262	3
2	59	73	5016	218	68	25	82	58	344	15
3	40	16	148	5315	19	199	23	41	297	33
4	24	26	145	39	4746	22	63	49	502	226
5	73	25	74	375	62	3868	91	22	771	60
6	50	19	128	13	84	105	5369	8	140	2
7	65	19	69	117	100	18	2	5641	112	122
8	47	96	127	196	77	135	30	23	5088	32
9	58	45	59	199	233	59	2	385	725	4184

```
plt.matshow(cm, cmap=plt.cm.gray) # plot the confusion matrix
```

- True positive for digit 5 and 9 are a bit low: opportunity for improvement
- Many digits are misclassified as 8, especially 5 & 9

Possible Improvements:

- Get more training data for digits 5/8/9
- Preprocess the image, e.g. make sure digits are centered & not rotated.

Next:

Linear Regression and Gradient Descent