17 - Splitting, Metrics

Data Science Practicum 2021/22, Lesson 17

Marko Tkalčič

Univerza na Primorskem

Table of Contents

Data splitting

- Our machine learning model needs to predict **future data**
- But we can train it only on **past data**, i.e. data that we have at hand

Document	f1	f2	f3	Label
It was the best of times	1	2	1	C
it was the worst of times	3	3	3	S
it was the age of wisdom	4	1	2	C
it was the age of foolishness	2	2	1	?

- Question: how well will our model perform on new data?

Data splitting

- Our machine learning model needs to predict **future data**
- But we can train it only on **past data**, i.e. data that we have at hand

Document	f1	f2	f3	Label
It was the best of times	1	2	1	C
it was the worst of times	3	3	3	S
it was the age of wisdom	4	1	2	C
it was the age of foolishness	2	2	1	?

- Question: how well will our model perform on new data?

- Data splitting

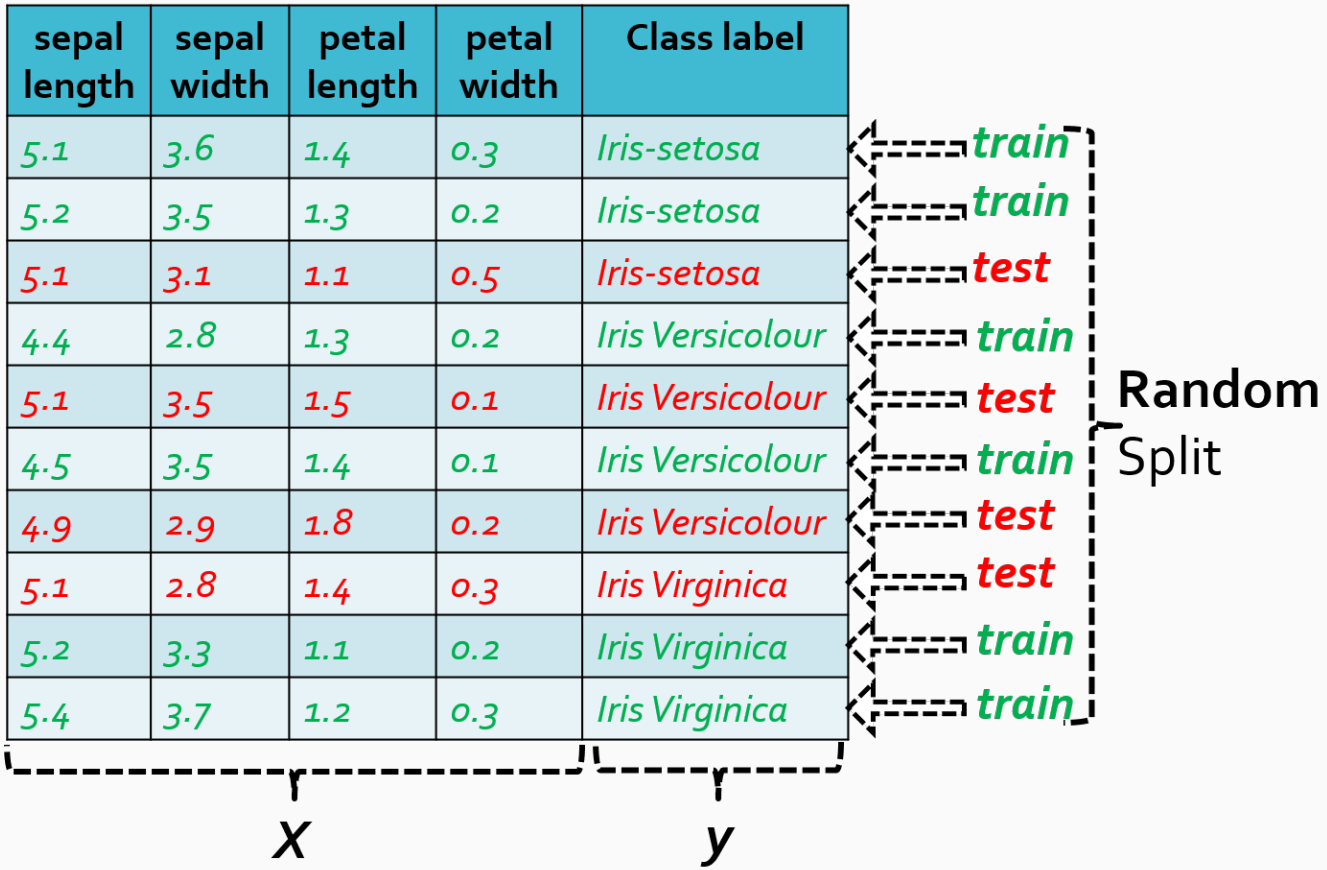
Data splitting

- We simulate the past/future data by splitting the data that we have into:
  - train set (past data)
  - test set (future data)

Document	f1	f2	f3	Label
It was the best of times	1	2	1	C
it was the worst of times	3	3	3	S
it was the age of wisdom	4	1	2	C
it was the age of foolishness	2	2	1	?

Document	f1	f2	f3	Label
It was the best of times	1	2	1	C
it was the worst of times	3	3	3	S
it was the age of wisdom	4	1	2	C
it was the age of foolishness	2	2	1	?

Data splitting



Data Splitting

sepal length	sepal width	petal length	petal width	Class label
5.1	3.6	1.4	0.3	Iris-setosa
5.2	3.5	1.3	0.2	Iris-setosa
4.4	2.8	1.3	0.2	Iris Versicolour
...	...	...	...	...
5.2	3.3	1.1	0.2	Iris Virginica
5.4	3.7	1.2	0.3	Iris Virginica

Train set

70% of dataset

`X_train` `y_train`

sepal length	sepal width	petal length	petal width	Class label
5.1	3.1	1.1	0.5	Iris-setosa
5.1	3.5	1.5	0.1	Iris Versicolour
...	...	...	...	...
5.1	2.8	1.4	0.3	Iris Virginica

Test set

30% of dataset

`X_test` `y_test`

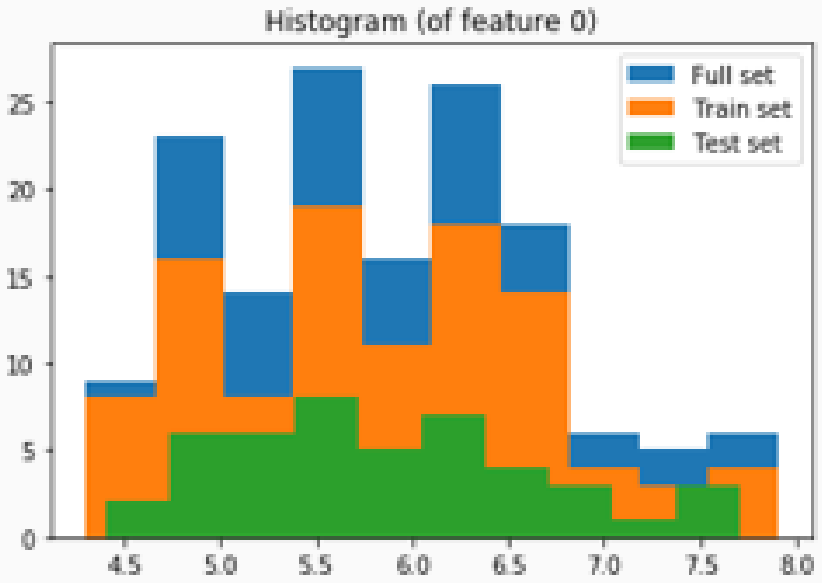
train_test_split

```
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn import datasets

iris = datasets.load_iris()
X = iris.data
y = iris.target

ts = 0.3

X_train, X_test, y_train, y_test = train_test_split(X,y,test_size=ts)

print("Train set relative size: ",X_train.size / X.size)
print("Test set relative size: ",X_test.size / X.size)
```

```
Train set relative size:  0.7
Test set relative size:  0.3
```

Exercise

- Write the function `split_show(X,y,my_test_size)` that takes the dataset and test size
- It performs the splitting
- It plots the histograms of one variable (e.g. the first) in the full, train and test sets



Exercise

```
from matplotlib import pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn import datasets

def split_show(X,y,my_test_size):
    X_train, X_test, y_train, y_test = train_test_split(X,y,test_size=my_test_size)
    plt.hist(X[:,0])
    plt.hist(X_train[:,0])
    plt.hist(X_test[:,0])
    plt.legend(["Full set","Train set","Test Set"])
    plt.show()

iris = datasets.load_iris()
X = iris.data
y = iris.target

ts = 0.3

split_show(X,y,ts)
```

Table of Contents

Evaluation

- After the train/test splitting we:
  - train our model on the train set
  - evaluate the quality of the model on the test set

Evaluation

- After the train/test splitting we:
  - train our model on the train set
  - evaluate the quality of the model on the test set

- In sklearn, three ways of doing evaluation:
  - each model has a default **score** method (depends on the model, see documentation)
  - model evaluation tools (e.g. `cross_val_score`) have internal scoring parameter
  - functions in the **metrics** module

Default score method

```
model = RandomForestClassifier()
model.fit(X_train,y_train)
a_tr = model.score(X_train,y_train) # Mean accuracy of model.predict(X_train) wrt. y_train
```

Default score method

```
model = RandomForestClassifier()
model.fit(X_train,y_train)
a_tr = model.score(X_train,y_train) # Mean accuracy of model.predict(X_train) wrt. y_train
```

```
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn import datasets
from sklearn.ensemble import RandomForestClassifier

iris = datasets.load_iris()
X = iris.data
y = iris.target

ts = 0.3

X_train, X_test, y_train, y_test = train_test_split(X,y,test_size=ts)
model = RandomForestClassifier(max_depth=2, random_state=0)

model.fit(X_train,y_train)
a_tr = model.score(X_train,y_train)
a_ts = model.score(X_test,y_test)
print(a_tr)
print(a_ts)
```

```
0.9714285714285714
0.9333333333333333
```

Exercise

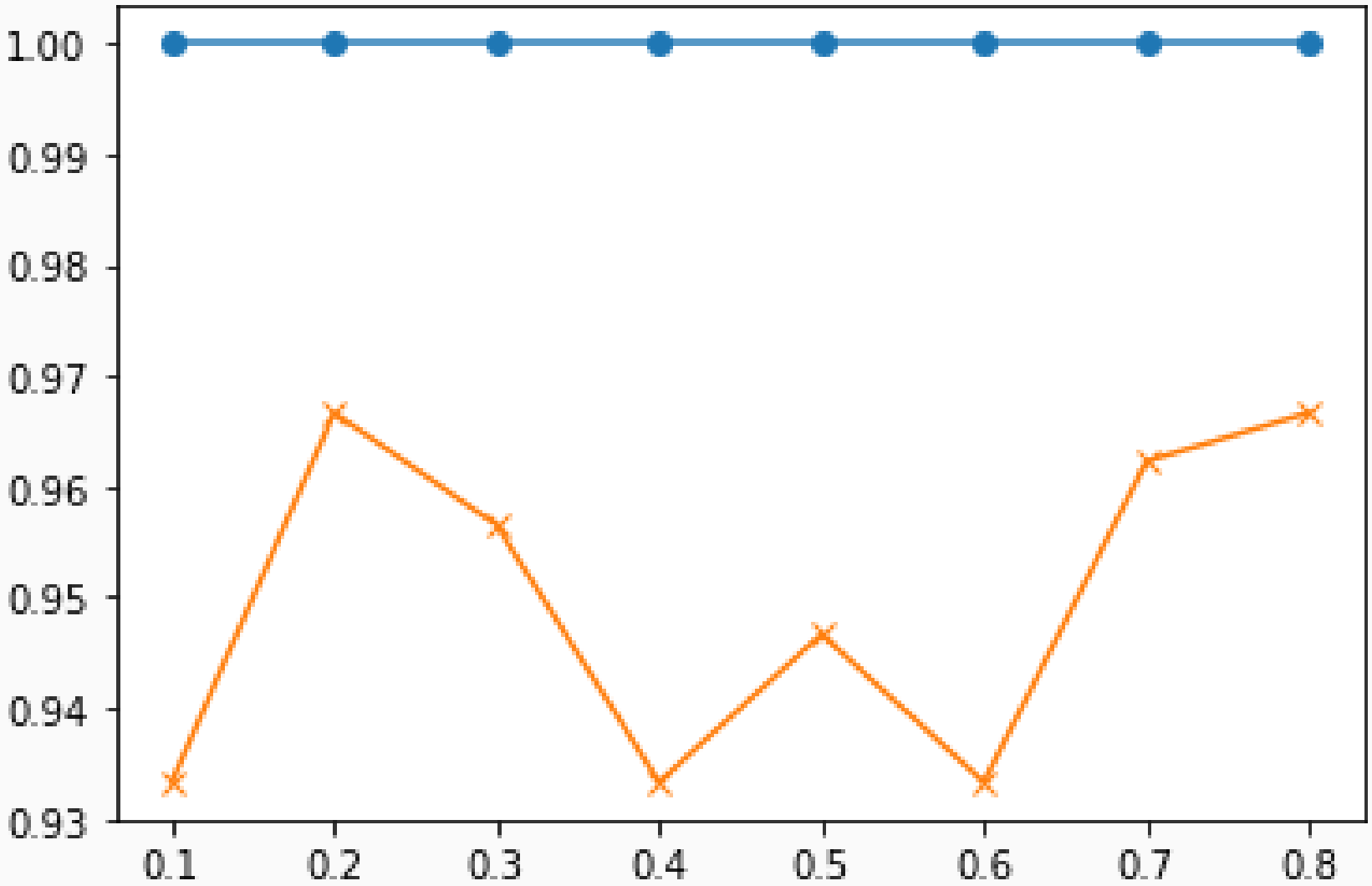
- Load the iris dataset
- Write a piece of code that iterates the test size from 0.1 to 0.9 in steps of 0.1
- Calculate the score for the test and train set on the RandomForestClassifier
- Plot both test and train vs test size

Exercise

- Load the iris dataset
- Write a piece of code that iterates the test size from 0.1 to 0.9 in steps of 0.1
- Calculate the score for the test and train set on the RandomForestClassifier
- Plot both test and train vs test size

```
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn import datasets
from sklearn.ensemble import RandomForestClassifier
from matplotlib import pyplot as plt

iris = datasets.load_iris()
X = iris.data
y = iris.target

a_tr = []
a_ts = []

tss = np.arange(0.1,0.9,0.1)
for ts in tss:
    X_train, X_test, y_train, y_test = train_test_split(X,y,test_size=ts)
    model = RandomForestClassifier(random_state=2)
    model.fit(X_train,y_train)
    a_tr.append(model.score(X_train,y_train))
    a_ts.append(model.score(X_test,y_test))

plt.plot(tss,a_tr,"-o")
plt.plot(tss,a_ts,"-x")
plt.show()
```

Exercise



Scoring parameter

- The methods
  - `model_selection.GridSearchCV`
  - `model_selection.cross_val_score`
- take the *scoring* parameter
  - predefined
  - custom

Scoring	Function
Classification	
'accuracy'	metrics.accuracy_score
'average_precision'	metrics.average_precision_score
'f1'	metrics.f1_score
'neg_log_loss'	metrics.log_loss
'precision' etc.	metrics.precision_score
'recall' etc.	metrics.recall_score
'jaccard' etc.	metrics.jaccard_score
'roc_auc'	metrics.roc_auc_score
Clustering	
'completeness_score'	metrics.completeness_score
'homogeneity_score'	metrics.homogeneity_score
'mutual_info_score'	metrics.mutual_info_score

Scoring	Function
Regression	
'explained_variance'	metrics.explained_variance_score
'max_error'	metrics.max_error
'neg_mean_absolute_error'	metrics.mean_absolute_error
'neg_mean_squared_error'	metrics.mean_squared_error
'neg_root_mean_squared_error'	metrics.mean_squared_error
'neg_mean_squared_log_error'	metrics.mean_squared_log_error
'neg_median_absolute_error'	metrics.median_absolute_error
'r2'	metrics.r2_score
'neg_mean_poisson_deviance'	metrics.mean_poisson_deviance
'neg_mean_gamma_deviance'	metrics.mean_gamma_deviance

Scoring Parameter

```
from sklearn import svm, datasets
from sklearn.model_selection import cross_val_score

iris = datasets.load_iris()
X = iris.data
y = iris.target
model = svm.SVC()
print(cross_val_score(model, X, y, cv=5, scoring='f1_macro'))
print(cross_val_score(model, X, y, cv=5, scoring='accuracy'))
```

```
[0.96658312 0.96658312 0.96658312 0.93333333 1.        ]
[0.96666667 0.96666667 0.96666667 0.93333333 1.        ]
```

Scoring Parameter

```
from sklearn import svm, datasets
from sklearn.model_selection import cross_val_score

iris = datasets.load_iris()
X = iris.data
y = iris.target
model = svm.SVC()
print(cross_val_score(model, X, y, cv=5, scoring='f1_macro'))
print(cross_val_score(model, X, y, cv=5, scoring='accuracy'))
```

```
[0.96658312 0.96658312 0.96658312 0.93333333 1.        ]
[0.96666667 0.96666667 0.96666667 0.93333333 1.        ]
```

```
from sklearn.metrics import make_scorer

def custom_func(y_1, y_2):
    ...

model = svm.SVR()
score = make_scorer(custom_func, greater_is_better=False)
print(cross_val_score(model, X, y, cv=5, scoring=score))
```

Exercise

- Write a code that
  - uses the IRIS dataset
  - splits the data in 70:30
  - uses the svm.SVR() regressor
  - implement a custom scoring function `my_custom_loss_func(y_true, y_pred)`
    - returns the max absolute value of the difference between y_true, y_pred
    - evaluate the prediction on the train set using your function

Exercise

- Write a code that
  - uses the IRIS dataset
  - splits the data in 70:30
  - uses the svm.SVR() regressor
  - implement a custom scoring function `my_custom_loss_func(y_true, y_pred)`
    - returns the max absolute value of the difference between y_true, y_pred
    - evaluate the prediction on the train set using your function

```
from sklearn import svm, datasets
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import train_test_split
import numpy as np

def my_custom_loss_func(y_true, y_pred):
    diff = np.abs(y_true - y_pred).max()
    return diff

iris = datasets.load_iris()
X = iris.data
y = iris.target

X_train, X_test, y_train, y_test = train_test_split(X,y,test_size=0.3)

model = svm.SVR()
ft = model.fit(X_train,y_train)
my_custom_loss_func(y_test, ft.predict(X_test))
```

```
0.7316016952184128
```

Exercise

- Repeat the previous exercise with the following differences:
  - don't split the data
  - don't call your function
  - use `make_Scorer` and `cross_val_score`

Exercise

- Repeat the previous exercise with the following differences:
  - don't split the data
  - don't call your function
  - use `make_Scorer` and `cross_val_score`

```
from sklearn import svm, datasets
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import train_test_split
import numpy as np
from sklearn.metrics import make_scorer

def my_custom_loss_func(y_true, y_pred):
    diff = np.abs(y_true - y_pred).max()
    return diff

iris = datasets.load_iris()
X = iris.data
y = iris.target

model = svm.SVR()
score = make_scorer(my_custom_loss_func, greater_is_better=False)
print(cross_val_score(model, X, y, cv=5, scoring=score))
```

```
[-0.12028995 -0.38782271 -0.76467227 -0.47468343 -0.61528216]
```

Classification Metrics

- `sklearn.metrics` implements several loss, score, and utility functions to measure classification performance.
- Classification
  - binary class (f1_score, roc_auc_score)
  - multiclass - a collection of binary problems
    - `average = {None, macro, weighted, micro, samples}`

Binary:

precision_recall_curve(y_true, probas_pred, *)	Compute precision-recall pairs for different probability thresholds
roc_curve(y_true, y_score, *[, pos_label, . . .])	Compute Receiver operating characteristic (ROC)

Multiclass:

balanced_accuracy_score(y_true, y_pred, *[, . . .])	Compute the balanced accuracy
cohen_kappa_score(y1, y2, *[, labels, . . .])	Cohen's kappa: a statistic that measures inter-annotator agreement.
confusion_matrix(y_true, y_pred, *[, . . .])	Compute confusion matrix to evaluate the accuracy of a classification.
hinge_loss(y_true, pred_decision, *[, . . .])	Average hinge loss (non-regularized)
matthews_corrcoef(y_true, y_pred, *[, . . .])	Compute the Matthews correlation coefficient (MCC)
roc_auc_score(y_true, y_score, *[, average, . . .])	Compute Area Under the Receiver Operating Characteristic Curve (ROC AUC) from prediction scores.

Confusion Matrix

		Actual class	
		Cat	Dog
Predicted class	Cat	5	2
	Dog	3	3

- Cat = 1
- Dog = 0

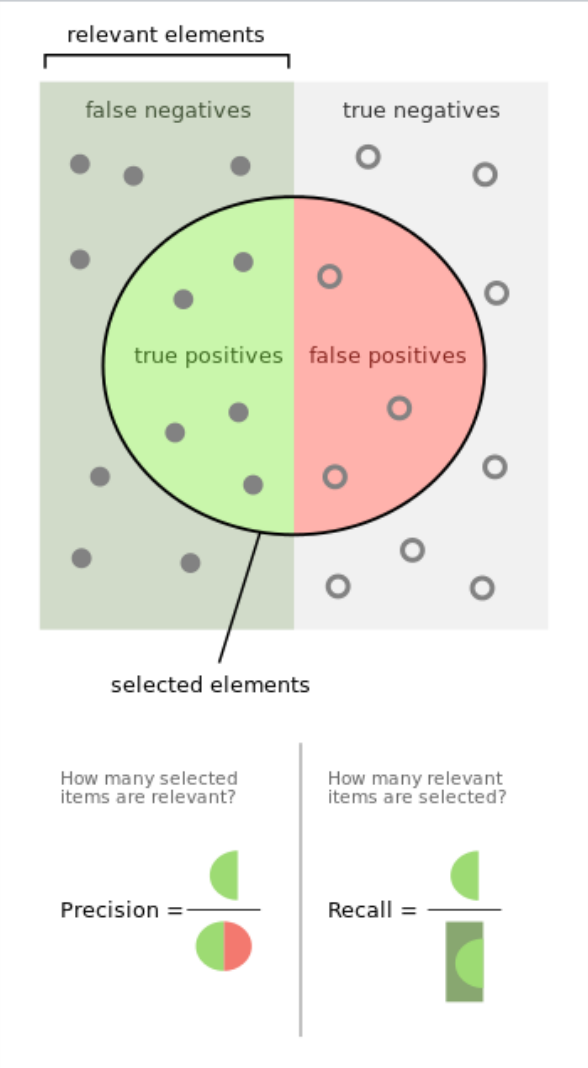
Confusion Matrix

		Actual class	
		Cat	Dog
Predicted class	Cat	5	2
	Dog	3	3

- Cat = 1
- Dog = 0

		Actual class	
		Cat	Dog
Predicted class	Cat	TP	FP
	Dog	FN	TN

Precision, Recall



$$P = \frac{TP}{TP + FP}$$

$$R = \frac{TP}{TP + FN}$$

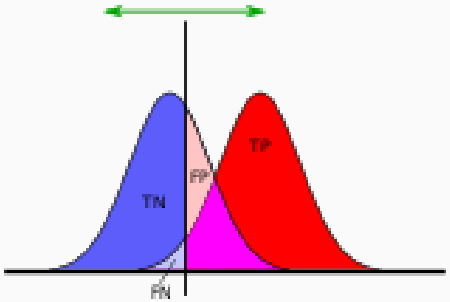
$$A = \frac{TP + TN}{TP + FP + FN + TN}$$

$$F = 2\frac{P \cdot R}{P + R}$$

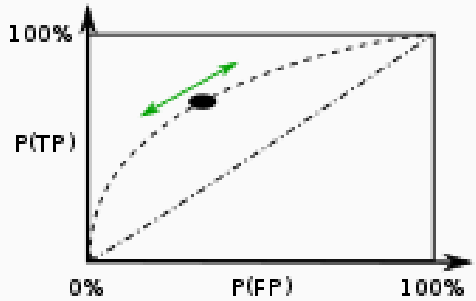
$$TPR = \frac{TP}{P}$$

$$FPR = \frac{FP}{N}$$

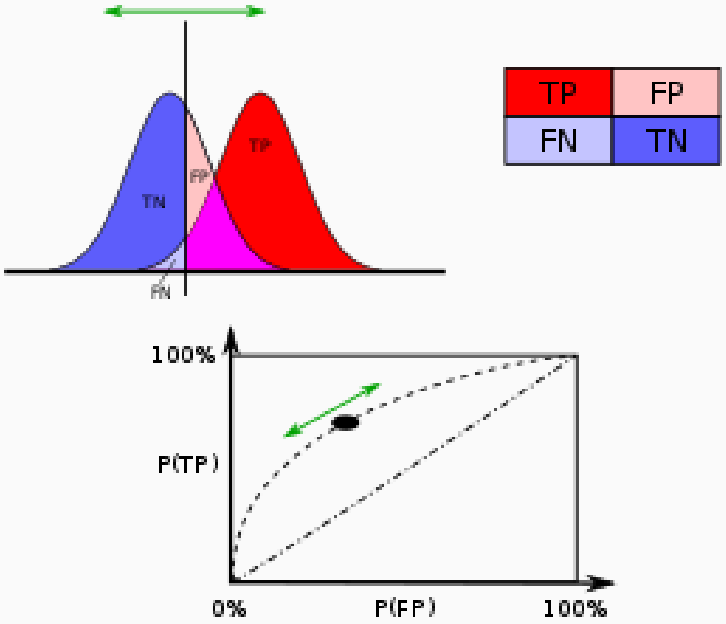
Receiver Operating Characteristicv (ROC)



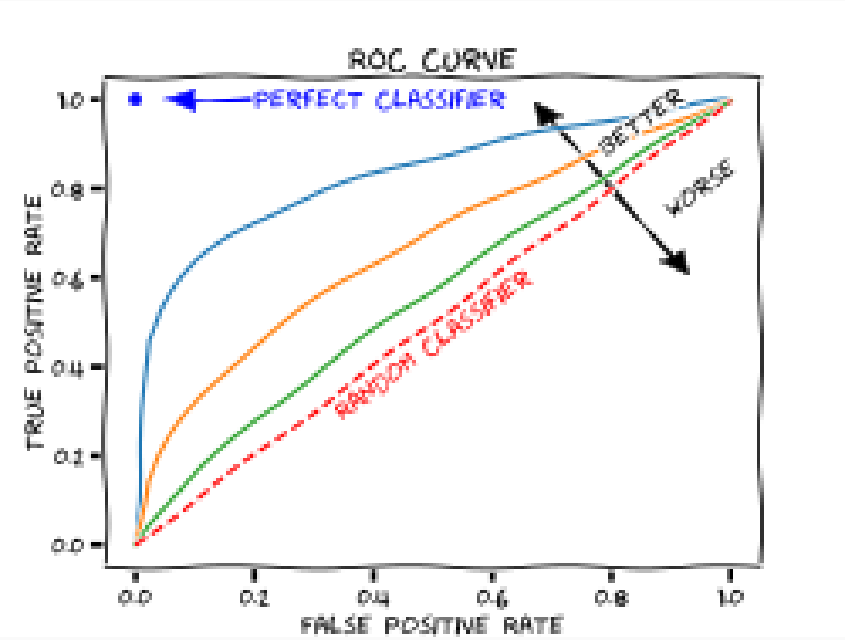
TP	FP
FN	TN



Receiver Operating Characteristicv (ROC)



TP	FP
FN	TN



$$TPR = \frac{TP}{P}$$

$$FPR = \frac{FP}{N}$$

Classification metrics

```
import numpy as np
from sklearn.metrics import accuracy_score
y_pred = [0, 2, 1, 3]
y_true = [0, 1, 2, 3]
accuracy_score(y_true, y_pred)
```

```
0,5
```

Exercise

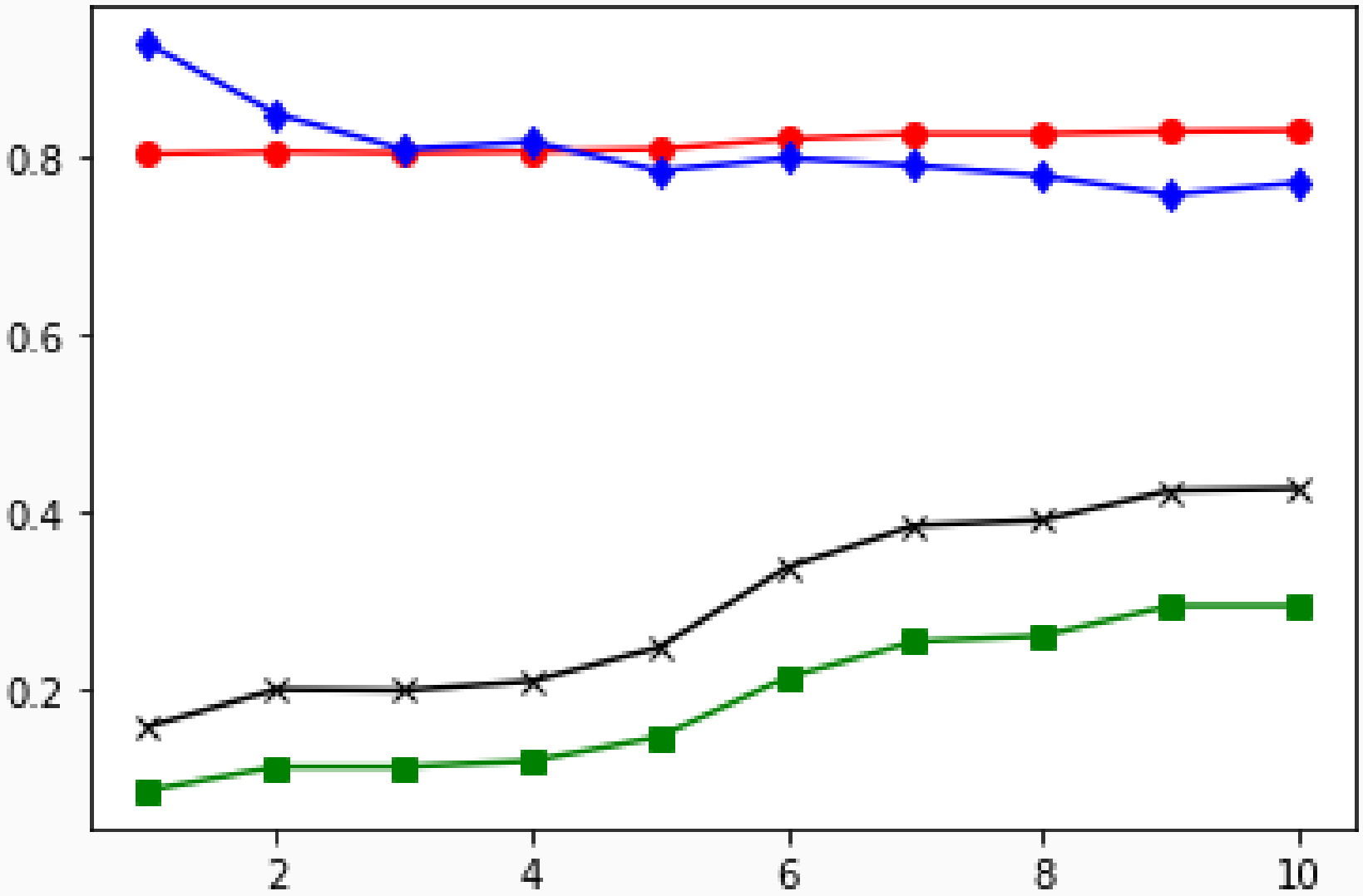
- Create a 2-class imbalanced (80/20) dataset with 10 features (7 informative, 3 redundant)
- Use the RandomForestClassifier(max_depth=n) and iterate n from 1 to 9
- Plot accuracy, precision, recall, and F measure vs max depth

Exercise

- Create a 2-class imbalanced (80/20) dataset with 10 features (7 informative, 3 redundant)
- Use the RandomForestClassifier(max_depth=n) and iterate n from 1 to 9
- Plot accuracy, precision, recall, and F measure vs max depth

```
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
from matplotlib import pyplot as plt
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score
from sklearn.datasets import make_classification

X, y = make_classification(weights=[0.8, 0.2], n_classes=2,  n_samples=1000, n_features=10,
                           n_informative=7, n_redundant=3, random_state=7)
a = []
p = []
r = []
f = []
X_train, X_test, y_train, y_test = train_test_split(X,y,test_size=0.7)
tss = range(1,11)
for n in tss:
    model = RandomForestClassifier(max_depth=n, random_state=2)
    model.fit(X_train,y_train)
    a.append(accuracy_score(model.predict(X_test),y_test))
    p.append(precision_score(model.predict(X_test),y_test))
    r.append(recall_score(model.predict(X_test),y_test))
    f.append(f1_score(model.predict(X_test),y_test))
plt.plot(tss,a,"-or")
plt.plot(tss,p,"-sg")
plt.plot(tss,r,"-db")
plt.plot(tss,f,"-xk")
plt.show()
```

Exercise



Regression Metrics

- Regression: prediction of continuous variables

metrics.explained_variance_score(y_true, ...)	Explained variance regression score function
metrics.max_error(y_true, y_pred)	max_error metric calculates the maximum residual error.
metrics.mean_absolute_error(y_true, y_pred, *)	Mean absolute error regression loss
metrics.mean_squared_error(y_true, y_pred, *)	Mean squared error regression loss
metrics.mean_squared_log_error(y_true, y_pred, *)	Mean squared logarithmic error regression loss
metrics.median_absolute_error(y_true, y_pred, *)	Median absolute error regression loss
metrics.r2_score(y_true, y_pred, *[, ...])	R^2 (coefficient of determination) regression score function.
metrics.mean_poisson_deviance(y_true, y_pred, *)	Mean Poisson deviance regression loss.
metrics.mean_gamma_deviance(y_true, y_pred, *)	Mean Gamma deviance regression loss.
metrics.mean_tweedie_deviance(y_true, y_pred, *)	Mean Tweedie deviance regression loss.

Regression Metrics

```
from sklearn.metrics import explained_variance_score
y_true = [3, -0.5, 2, 7]
y_pred = [2.5, 0.0, 2, 8]
print(explained_variance_score(y_true, y_pred))
```

Exercise

- create a dataset using `make_classification` with 1000 samples and 10 features
- use RandomForestRegressor and iterate max_depth from 1 to 10
- plot the following regression metrics: `mean_absolute_error` ,`mean_squared_error`, `mean_squared_log_error`, `median_absolute_error`, `r2_score`

Exercise

- create a dataset using `make_classification` with 1000 samples and 10 features
- use RandomForestRegressor and iterate max_depth from 1 to 10
- plot the following regression metrics: `mean_absolute_error` ,`mean_squared_error`, `mean_squared_log_error`, `median_absolute_error`, `r2_score`

```
from sklearn.metrics import mean_absolute_error ,mean_squared_error, mean_squared_log_error, median_absolute_error, r2_score
from sklearn.datasets import make_classification
from sklearn.ensemble import RandomForestRegressor
from sklearn.model_selection import train_test_split
from matplotlib import pyplot as plt
X, y = make_classification(n_samples=1000, n_features=10, random_state=7)
mae = []
mse = []
msle = []
medae = []
r2 = []
X_train, X_test, y_train, y_test = train_test_split(X,y,test_size=0.7)
tss = range(1,11)
for n in tss:
    model =  RandomForestRegressor(max_depth=n, random_state=2)
    model.fit(X_train,y_train)
    mae.append(mean_absolute_error(model.predict(X_test),y_test))
    mse.append(mean_squared_error(model.predict(X_test),y_test))
    msle.append(mean_squared_log_error(model.predict(X_test),y_test))
    medae.append(median_absolute_error(model.predict(X_test),y_test))
    r2.append(r2_score(model.predict(X_test),y_test))
plt.plot(tss,mae,"-or")
plt.plot(tss,mse,"-sg")
plt.plot(tss,msle,"-db")
plt.plot(tss,medae,"-xk")
plt.plot(tss,r2,"-xy")
plt.show()
```

Exercise

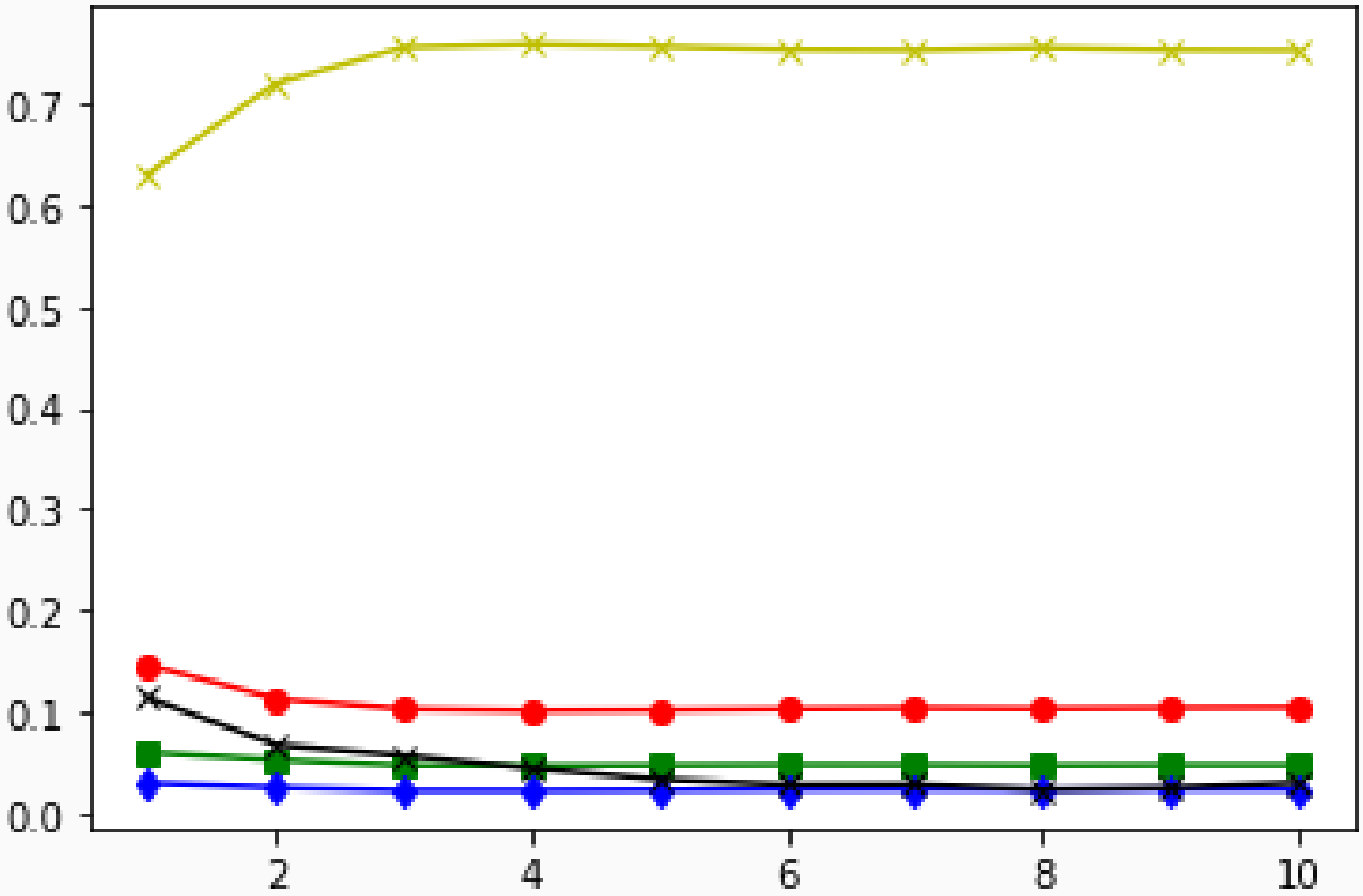


Table of Contents

References

Part of the material has been taken from the following sources. The usage of the referenced copyrighted work is in line with fair use since it is for nonprofit educational purposes.

- https://scikit-learn.org/stable/modules/model_evaluation.html
- https://en.wikipedia.org/wiki/Precision_and_recall