Artificial Intelligence with Python

Classification

Classification

In machine learning, classification means the prediction of the value of a categorical variable (having finitely many different values) based on training data. In other words, the input data is to be classified or labeled using categories such as 0 or 1.

Examples of classification problems:

- predict if email is spam or not
- predict if image depicts cat or not
- predict if a patient has cancer or not
- predict the party a person is going to vote for
- predict if a bank should give a person loan or not

In simplest form classification deals with two possible classes. This is called binary classification. In binary classification problems one of the two classes usually represents a "normal" state and the other corresponds to "abnormal" state (see examples above).

Algorithms for binary classification include

- logistic regression
- k-nearest neighbors ("knn")
- decision trees
- support vector machine (SVM)
- naive Bayes

Some of these algorithms work also with multiclass prediction problems.

Multiclass classification means, obviously, the task of assigning input sample to one of several classes.

- For example:
- optical character recognition (OCR); assign e.g. handwritten digits to 10 classes
- image recognition; detect/predict persons in photographs
- plant species classification

Here we don't have the notion of normal and abnormal states as the input samples are to be assigned to multitude of different classes.

Algorithms:

- k-nearest neighbors
- decision trees
- naive Bayes
- random forest
- gradient boosting

There is a trick which allows one to apply binary classification algorithms to work also in multiclass classification problems. It is called one-vs-rest strategy.

Multi-label classification means that input samples may be assigned to several labels at the same time. For example, a photo may represent cats, dogs and giraffes. In this case this photo receives three labels. Here the above algorithms do not apply directly but they can be modified to work here too.

Model evaluation, confusion matrix

Suppose we are solving a binary classification problem with classes 0 and 1 and we record our predicted results against true labels in a table like below.

sample #	prediction	true
1	1	0
2	1	1
3	0	0
4	1	1
5	0	0
6	0	0
7	1	0
8	1	1
9	0	0
10	1	1

confusion matrix

sample #	prediction	true
1	1	0
2	1	1
3	0	0
4	1	1
5	0	0
6	0	0
7	1	0
8	1	1
9	0	0
10	1	1

Using this table we can easily calculate the number of correctly predicted classes for each class.

True positives (TP) is the number of correctly predicted 1's. Here 4.

True negatives (TN) is the number of correctly predicted 0'. Here 4.

False positives (FP) is the number of incorrectly predicted 1's. Here 2.

False negatives (FN) is the number of incorrectly predicted 0's. Here 0.

These four numbers sum up to the total number of predictions. Here 10.

Moreover, these numbers are often presented in 2-by-2 array form as follows:

n=10	predicted 0	predicted 1	
actual 0	TN=4	FP=2	6
actual 1	FN=0	TP=4	4
	4	6	

Looking at the column sums we can also say that we predicted 0 4 times and we predicted 1 6 times. Similarly, looking at the row sums we say that, actually, we had 6 samples of class 0 and 4 samples of class 1.

Confusion matrix allows us to compute some metrics which indicate the performance of our predictions. Accuracy is defined as the ratio

$$\text{accuracy} = \frac{TP + TN}{n} = 0.8$$

Precision means how often the model is correct when predicting a class 1. It is defined as

$$\text{precision} = \frac{TP}{TP + FP} = \frac{2}{3}$$

Recall (or sensitivity) is the proportion of cases correctly identified as belonging to class 1 among all cases that truly belong to class 1 i.e.

$$\text{recall} = \frac{TP}{TP + FN} = 1$$

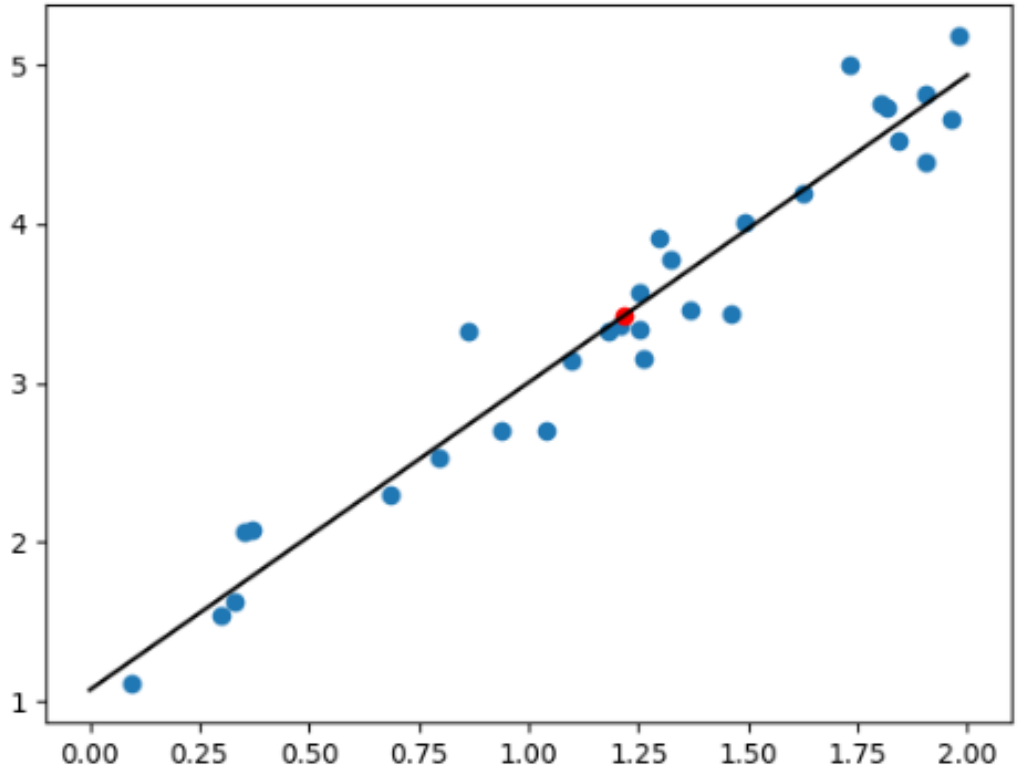
In multiclass classification precision and recall are calculated for each class by dividing the diagonal value with row and column sums, respectively.

Logistic regression, basic ideas

Logistic regression borrows ideas from linear regression that we have met earlier.

Recall how the aim in linear regression was to predict the value of a continuous variable y by fitting a straight line to data. The variable y was allowed to take on any numerical value, positive or negative, small or large.

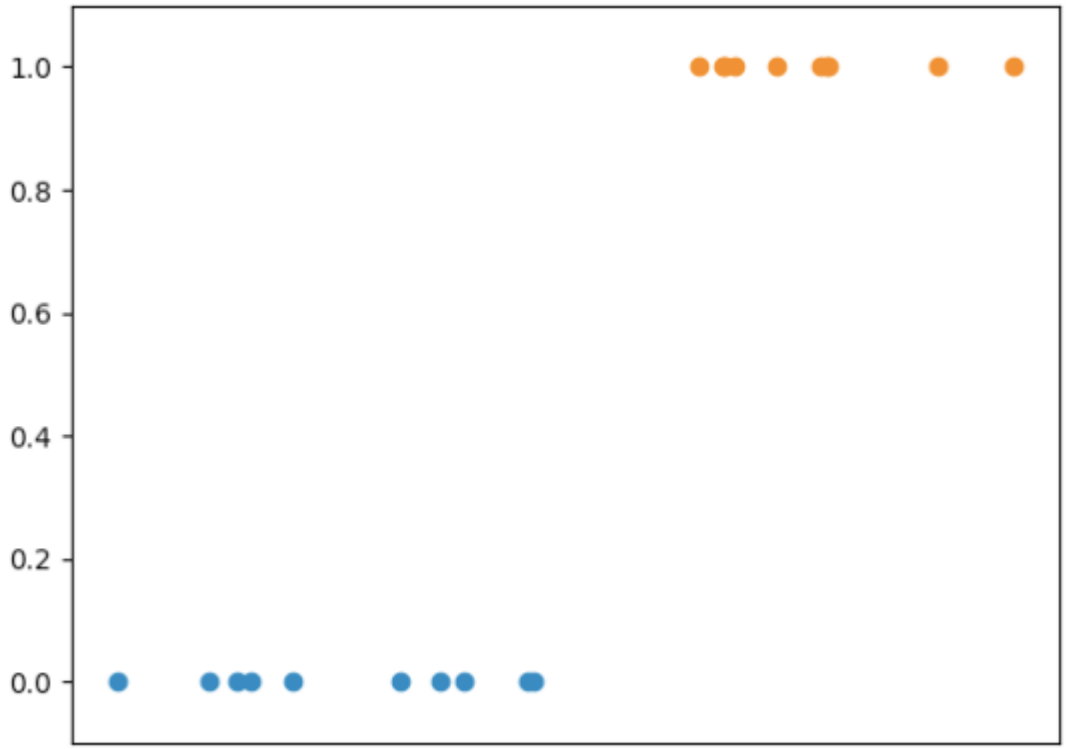
Logistic regression, basic ideas



In binary classification the situation is much more restricted. We are predicting the value of a categorical variable having only two possible values. Let us call them 0 and 1.

For example, we might have the following situation in terms of training data, where part of the data is labeled 1 and the rest 0.

Instead of just considering either 0 or 1 we extend our idea to allow the vertical axis to represent a probability of new data point belonging to class 1. In particular, this probability can not fall below 0 or above 1. The observed data points are exactly 0 or 1 but predictions might fall somewhere in between.



Fitting a straight line to data does not make sense anymore and we need to proceed differently. In a sense we try to fit a curved line (S-shaped).

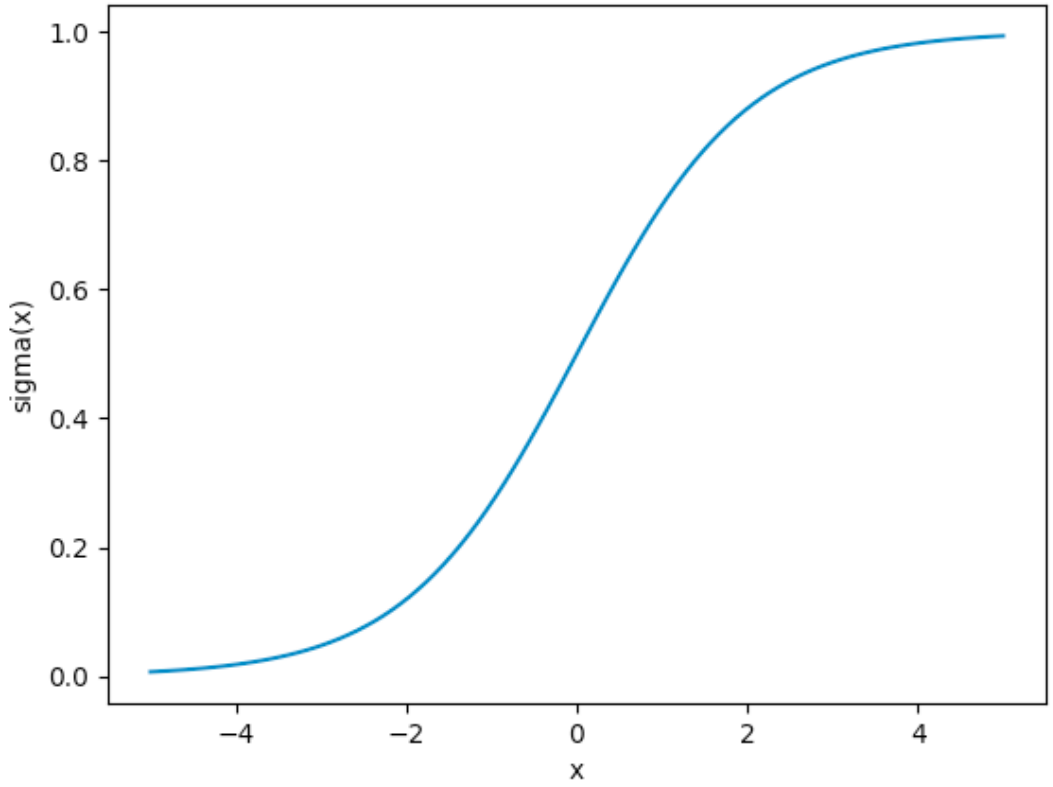
Hence, in the very core of logistic regression is the so called sigmoid function (or logistic function) defined by the formula

$$\sigma(x) = \frac{1}{1 + e^{-x}}$$

where e is the base in natural logarithm i.e. e = 2.7182818. This is also known as the exponential function and it is available in numpy as exp():

```
import numpy as np

print(np.exp(1))
```

The plot of the sigmoid function is shown below. It shows that the sigmoid function will convert (map) any value to be between 0 and 1.



We see that as x increases then σ(x) approaches 1 but never attains 1. Similarly, as x decreases then σ(x) approaches zero but never attains it either. In the middle when x = 0 we have

$$\sigma(0) = \frac{1}{1+e^0} = \frac{1}{1+1} = \frac{1}{2}$$

This middle value serves at the role of deciding if the model should predict 0 or 1 for the class label.

The sigmoid function has also central role in deep learning when one uses neural networks.

We will omit the mathematical details how logistic regression is developed and how it works under the hood. Instead, we focus on using it via sklearn module next.

Logistic regression, sklearn

sklearn module in Python provides again a high-level interface for logistic regression problems. The interface is used very similar to linear regression (simple or multiple).

Let us illustrate this using a dataset containing students' exam scores and information if student was admitted to university or not.

First of all, we import all the modules

```
import matplotlib.pyplot as plt
import pandas as pd
from sklearn.linear_model import LogisticRegression
from sklearn import metrics
from sklearn.model_selection import train_test_split

import seaborn as sns
```

The data is read from CSV in usual fashion

```
df = pd.read_csv("exams.csv", skiprows=0, delimiter=",")
print(df)
```

As the data contains three columns we pick the feature variables and target variables by column indexing

```
X = df.iloc[:, 0:2]
y = df.iloc[:, -1]
```

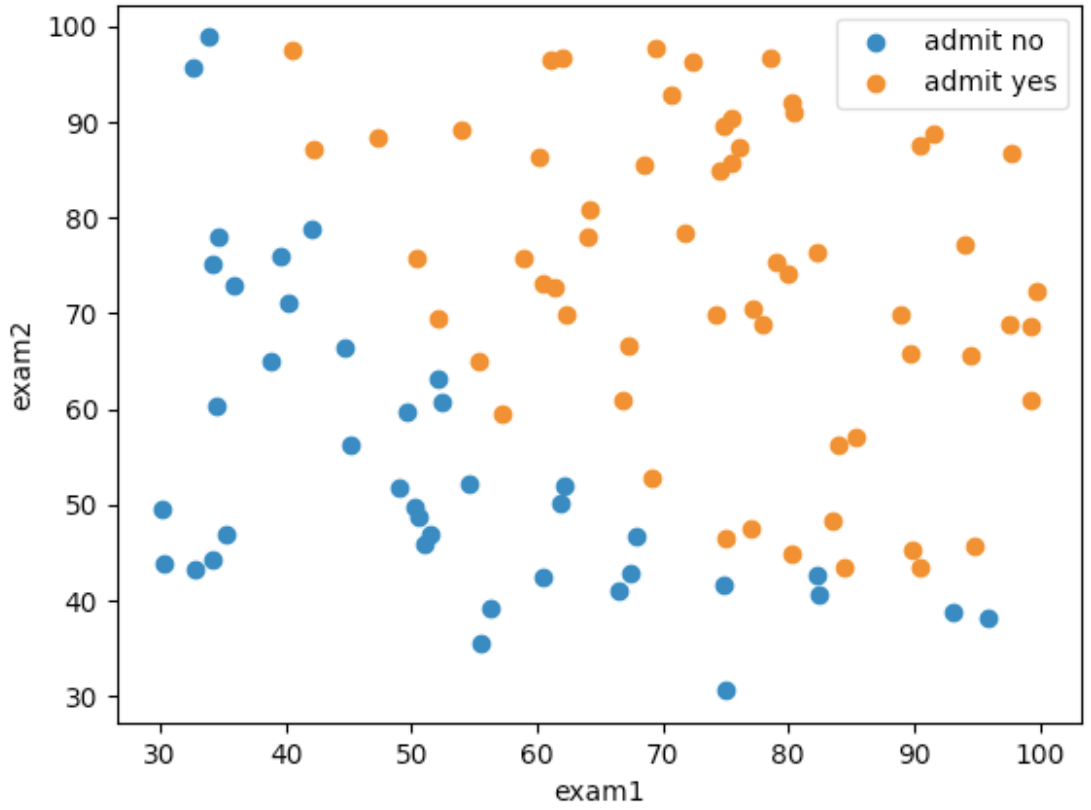
Next, we filter the rows based on the value in 3rd column (admit yes/no)

```
admit_yes = df.loc[y == 1]

admit_no = df.loc[y == 0]
```

Now it's time to prepare a scatter plot of both classes (exam1 vs. exam2)

```
plt.scatter(admit_no.iloc[:,0],admit_no.iloc[:,1],label="admit no")
plt.scatter(admit_yes.iloc[:,0],admit_yes.iloc[:,1],label="admit yes")
plt.xlabel("exam1")
plt.ylabel("exam2")
plt.legend()

plt.show()
```

This scatter plot shows clearly how student needs to do reasonably well in both exams in order to get admitted to university.



For training of our machine learning model

we do the usual data splitting into training and testing sets

```
X_train, X_test, y_train, y_test=train_test_split(X,y,test_size=0.25,random_state=0)
print(X_train.shape)
```

Model is constructed and trained simply by

```
model = LogisticRegression()
model.fit(X_train, y_train)
```

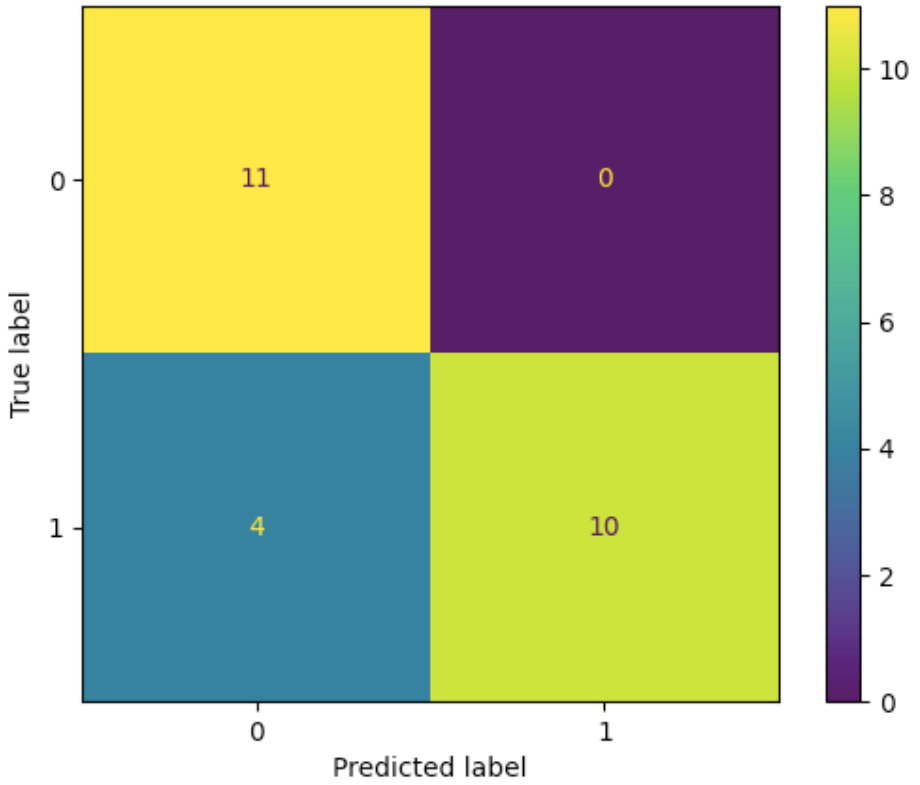
In order to evaluate the model we predict on the testing set

```
y_pred = model.predict(X_test)
```

Our first evaluation involves the confusion matrix,

both in numerical and visual form

```
cnf_matrix = metrics.confusion_matrix(y_test, y_pred)

print(cnf_matrix)metrics.ConfusionMatrixDisplay.from_estimator(model, X_test, y_test)


plt.show()
```


11
0
4
10
0
1
True label
Predicted label
10
8
6
4
2
0

Finally, we compute some other metrics

which are readily available in sklearn metrics interface

```
print("Accuracy:", metrics.accuracy_score(y_test, y_pred))
print("Precision:", metrics.precision_score(y_test, y_pred))
print("Recall:", metrics.recall_score(y_test, y_pred))
```

Accuracy: 0.84

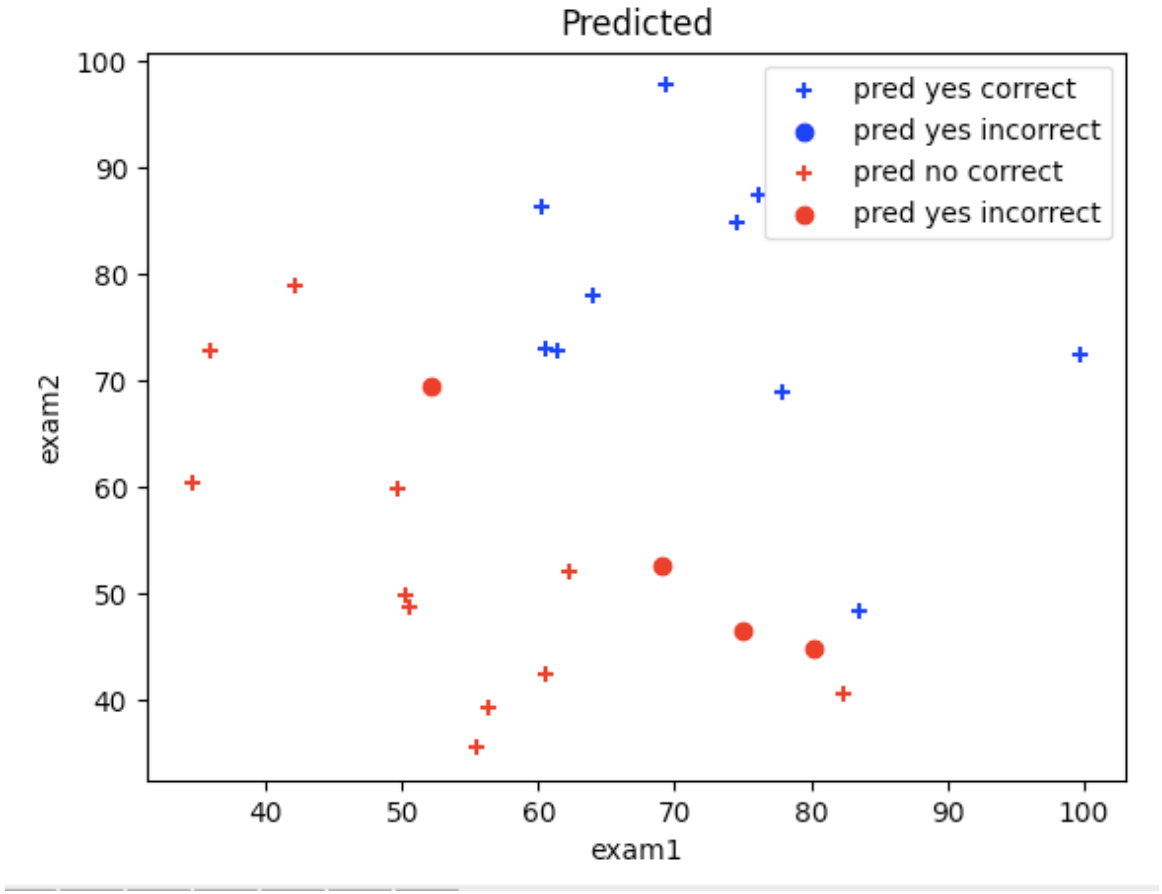
Precision: 1.0

Recall: 0.7142857142857143

For reference, we plot also the predicted labels as

```
y_test2 = y_test.to_numpy()
idx1 = np.logical_and(y_pred == 1, y_test2 == 1)
idx2 = np.logical_and(y_pred == 1, y_test2 == 0)
idx3 = np.logical_and(y_pred == 0, y_test2 == 0)
idx4 = np.logical_and(y_pred == 0, y_test2 == 1)
X1 = X_test.loc[idx1]
X2 = X_test.loc[idx2]
X3 = X_test.loc[idx3]
X4 = X_test.loc[idx4]

plt.scatter(X1.iloc[:,0],X1.iloc[:,1],label="pred yes correct",marker="+",color="blue")
plt.scatter(X2.iloc[:,0],X2.iloc[:,1],label="pred yes incorrect",marker="o",color="blue")
plt.scatter(X3.iloc[:,0],X3.iloc[:,1],label="pred no correct",marker="+",color="red")
plt.scatter(X4.iloc[:,0],X4.iloc[:,1],label="pred yes incorrect",marker="o",color="red")

plt.xlabel("exam1")
plt.ylabel("exam2")
plt.legend()
plt.title("Predicted")
plt.show()
```

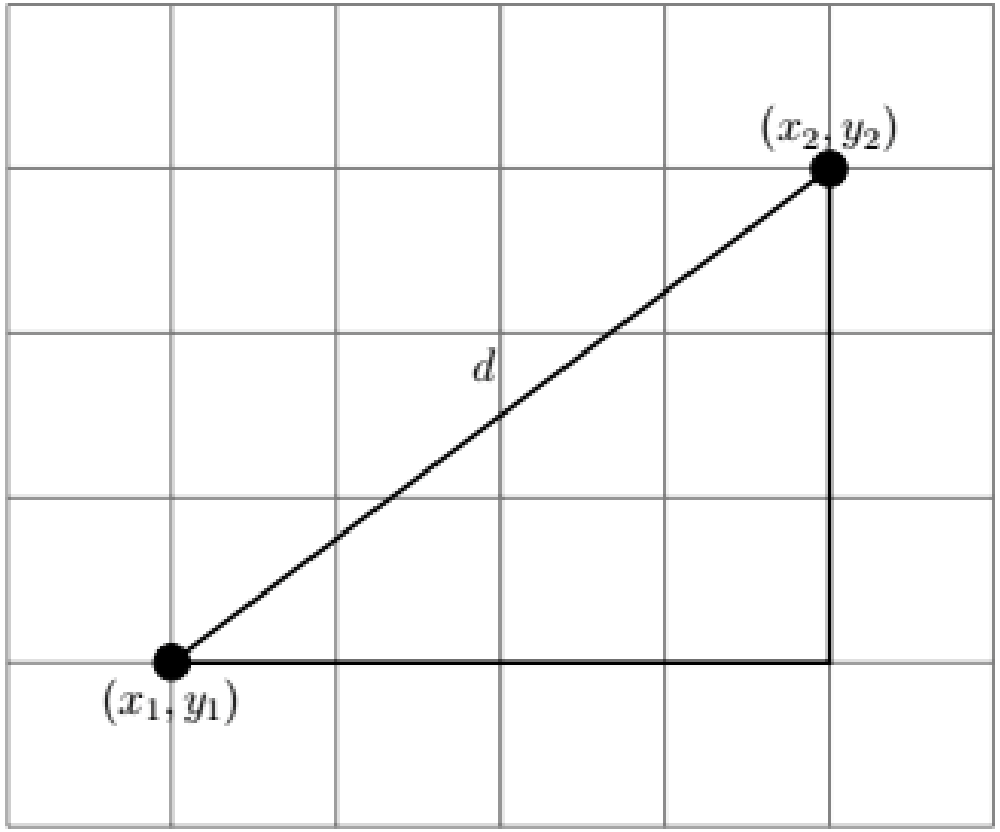
By visual inspection the model seems to have done quite a good job at predicting admittance based on the two exam scores.



k-nearest neighbors

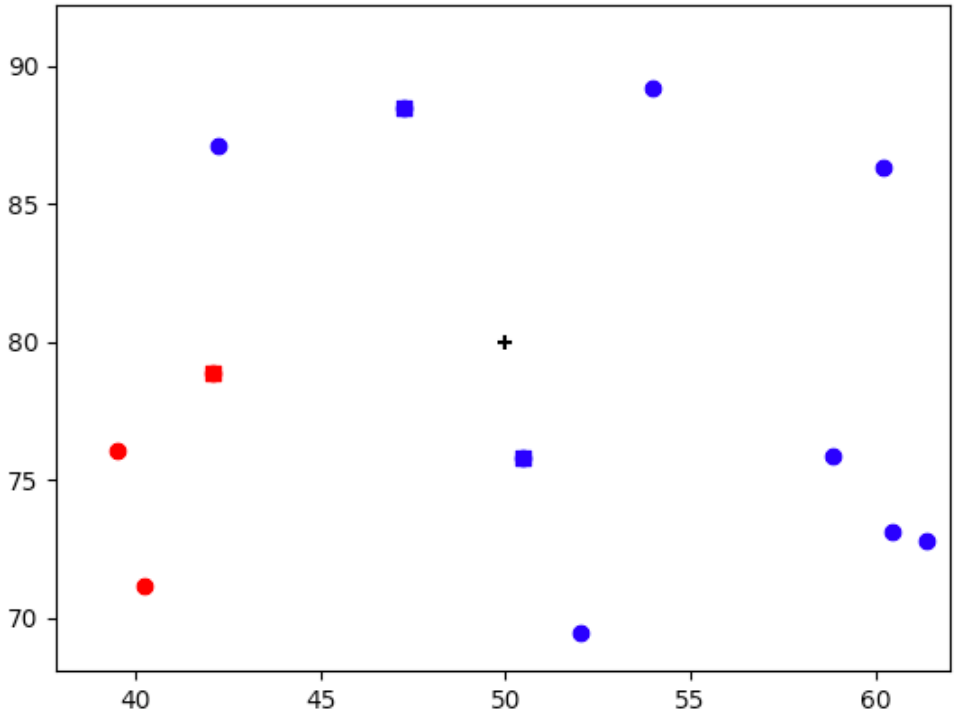
The basic idea in k-nearest algorithm is to compute distances to all data points and look for the nearest neighbors and their class. For the concept of distance several possibilities can be used e.g. Euclidean distance d or Manhattan distance d_M which are illustrated below.

$$d = \sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2}, \quad d_M = |x_1 - x_2| + |y_1 - y_2|$$



The algorithm finds k nearest neighbors, checks their class and assigns the class which majority of neighbors belong to. This is the so called majority vote principle. Knowing how sklearn handles ties (may happen if k is even) is difficult and hence it is better to use odd k in this course.

In the figure below, 3 nearest points to the black cross are indicated with square symbols. Colors indicate class labels.



The algorithm is non-parametric i.e. it does not make any assumptions on the underlying data. The choice of k is of course crucial and there are no automatic methods available for it but user needs to try different values to see what works best.

As the algorithm described above is very simple let us implement it for educational purposes.

K-nearest neighbors algorithm is available in sklearn as KNeighborsClassifier() and we illustrate it's usage here via an example.

```
import pandas as pd
import matplotlib.pyplot as plt
import numpy as np
from sklearn import metrics
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier

from sklearn.metrics import classification_report
```

The data consists of petal and sepal measurements in iris flowers.

```
df = pd.read_csv("iris.csv")
print(df)
```

We use all four numerical variables as features and species column as target

```
X = df.iloc[:, 0:4].values
y = df.iloc[:, 4].values
print(X)
print(y)
```

Split to training and testing sets again

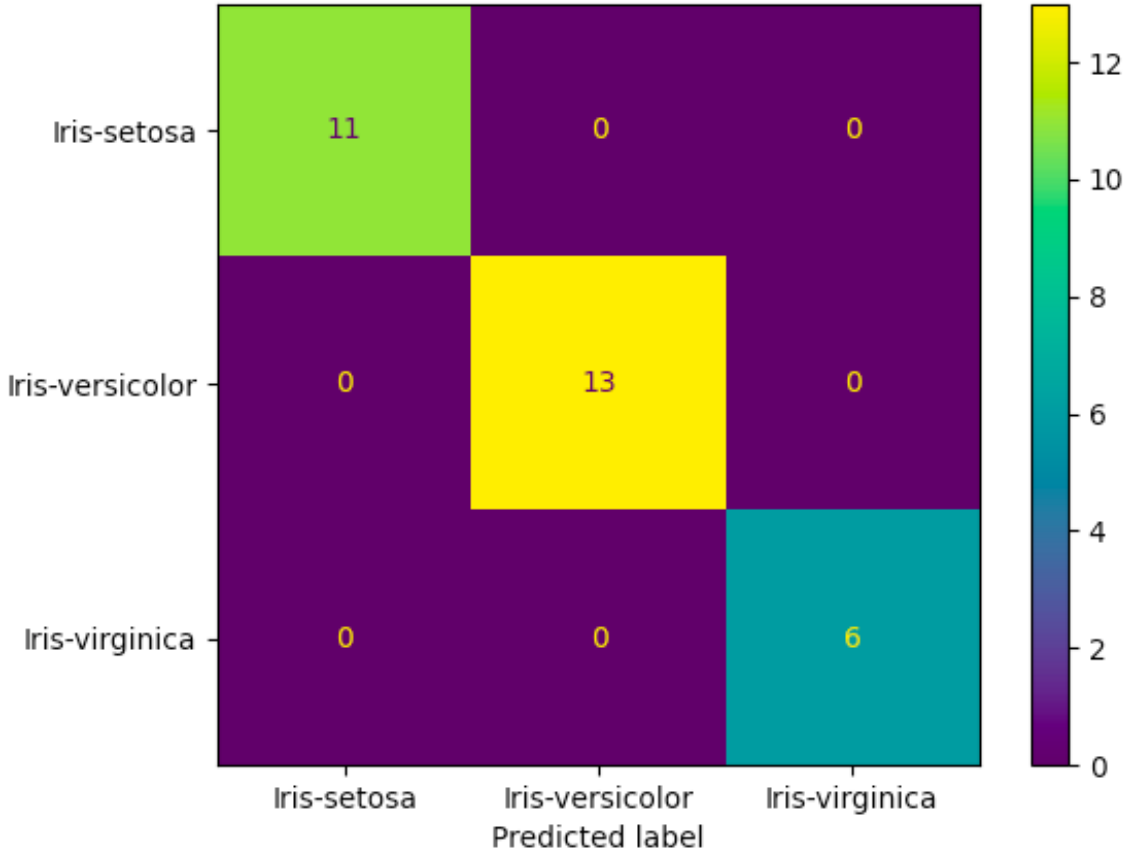
```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.20, random_state=1)
```

Setup KNN classifier, train it, and make predictions. We use k=5 here.

```
classifier = KNeighborsClassifier(n_neighbors=5)

classifier.fit(X_train, y_train)

y_pred = classifier.predict(X_test)
```

Evaluate the model using predictions of the test set. We form the confusion matrix

```
metrics.ConfusionMatrixDisplay.from_estimator(classifier, X_test, y_test)

plt.show()
```


Iris-setosa
Iris-versicolor
Iris-virginica
11
0
0
0
13
0
0
0
6
Iris-setosa
Iris-versicolor
Iris-virginica
Predicted label
12
10
8
6
4
2
0

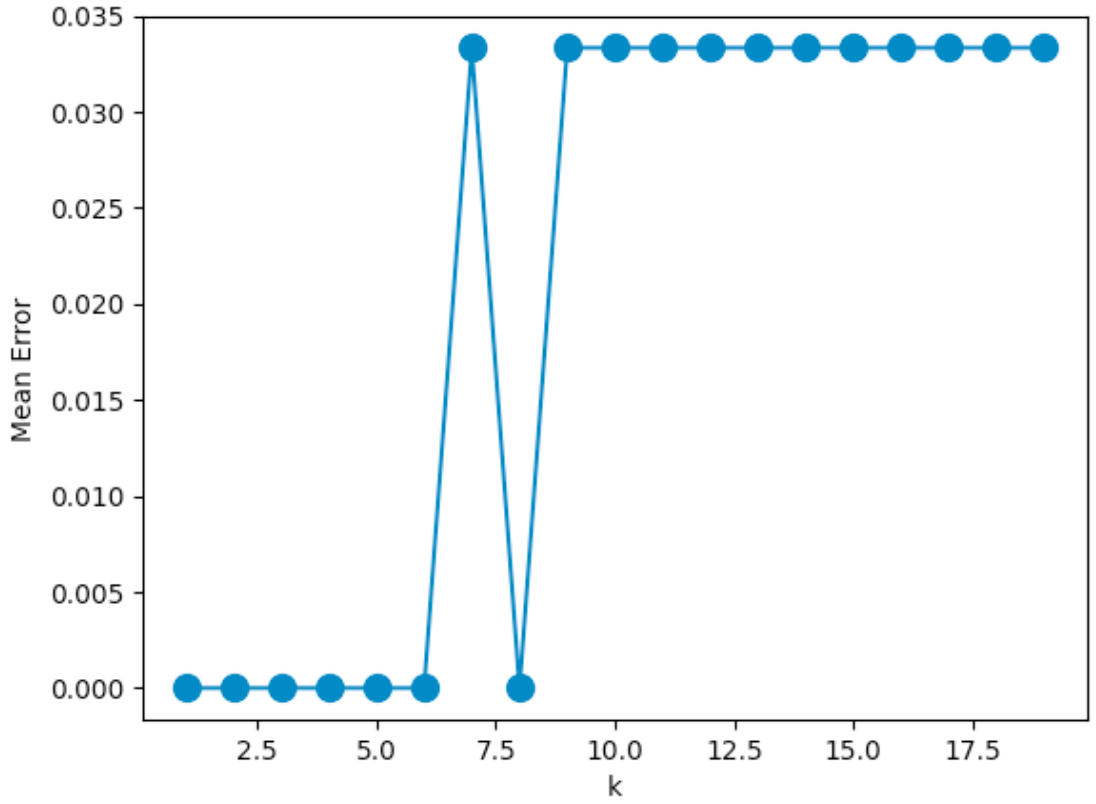
Also we ask sklearn to give us a classification report

```
print(classification_report(y_test, y_pred))
```

We see that we reached 100% accuracy in this example!

Above we pick the value k=5 somewhat randomly. To see how other values of k perform we check it using a loop and for each value of k we record the mean error rate.

```
error = []
for k in range(1, 20):
knn = KNeighborsClassifier(n_neighbors=k)
knn.fit(X_train, y_train)
y_pred = knn.predict(X_test)
error.append(np.mean(y_pred != y_test))plt.plot(range(1, 20), error, marker='o',
markersize=10)
plt.xlabel('k')
plt.ylabel('Mean Error')

plt.show()
```


0.035
0.030
0.025
0.020
0.015
0.010
0.005
0.000
Mean Error
2.5
5.0
7.5
10.0
12.5
15.0
17.5
k

THANKS!