SVM: Support Vector Machines

For this case the data is imported from sklearn datasets.
It is easier to use so it is good to test the models directly.
However since it is so easy to use and it is already prepared, it is unreal for a real case where data comes in irregular shape and with missing or wrong values.

SVM does classification: it predicts if the data belongs to a certain class (maybe like KNN?)

In [1]:
```
import sklearn
from sklearn import datasets
from sklearn import svm
```

In [2]:
```
# And import metrics and KNN to compare them
from sklearn import metrics
from sklearn.neighbors import KNeighborsClassifier
```

Get the data

We're going to use a breast cancer dataset, which is ok for this method.

In [3]:
```
cancer = datasets.load_breast_cancer()
print(type(cancer))
print(cancer.feature_names)
print(cancer.target_names)
```

```
<class 'sklearn.utils.Bunch'>
['mean radius' 'mean texture' 'mean perimeter' 'mean area'
 'mean smoothness' 'mean compactness' 'mean concavity'
 'mean concave points' 'mean symmetry' 'mean fractal dimension'
 'radius error' 'texture error' 'perimeter error' 'area error'
 'smoothness error' 'compactness error' 'concavity error'
 'concave points error' 'symmetry error' 'fractal dimension error'
 'worst radius' 'worst texture' 'worst perimeter' 'worst area'
 'worst smoothness' 'worst compactness' 'worst concavity'
 'worst concave points' 'worst symmetry' 'worst fractal dimension']
['malignant' 'benign']
```

Split into train and test batches

In [4]:
```
x = cancer.data
y = cancer.target
x_train, x_test, y_train, y_test = sklearn.model_selection.train_test_split(x
```

We took 20% of the data as the training data.
We can play with than number but Tim's recommendation is to not go over 30%.

In [5]:
```
print(x_train)
```

```
[[1.327e+01 1.702e+01 8.455e+01 ... 9.678e-02 2.506e-01 7.623e-02]
 [1.246e+01 1.989e+01 8.043e+01 ... 7.625e-02 2.685e-01 7.764e-02]
 [1.134e+01 1.861e+01 7.276e+01 ... 8.542e-02 3.060e-01 6.783e-02]
 ...
 [1.390e+01 1.924e+01 8.873e+01 ... 8.150e-02 2.356e-01 7.603e-02]
 [2.031e+01 2.706e+01 1.329e+02 ... 1.697e-01 3.151e-01 7.999e-02]
 [1.181e+01 1.739e+01 7.527e+01 ... 4.306e-02 3.200e-01 6.576e-02]]
```

In [6]:

```
print(y_train)
```

```
[1 1 1 1 0 1 0 0 0 1 1 0 0 1 0 1 0 1 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 0 0 1
 1 1 0 1 1 0 1 0 1 1 1 1 1 0 1 1 1 0 1 1 0 1 0 1 0 0 1 1 0 1 0 0 1 1 1 1 0
 0 0 0 0 1 0 1 1 1 0 1 0 0 1 0 0 0 1 1 1 1 0 1 0 1 0 0 0 1 1 0 0 1 0 0 1 0
 1 1 1 1 1 1 1 1 1 1 0 0 1 1 0 0 1 1 0 1 1 1 1 1 1 1 1 1 0 0 1 1 1 1 1 1 0
 1 0 1 1 0 0 0 1 0 1 0 1 0 1 1 0 1 0 0 1 1 1 1 1 1 0 0 0 1 1 0 1 1 1 1 1 0
 0 1 0 1 0 1 1 1 0 1 0 0 1 1 0 0 0 0 1 1 0 1 1 0 1 0 1 1 1 1 0 1 0 1 1 1 1
 0 1 0 0 1 1 1 0 1 0 0 1 1 1 0 1 0 1 1 0 1 0 0 1 1 0 1 1 1 1 1 0 0 1 1 1 1
 0 0 1 1 1 1 0 1 1 0 0 0 1 1 1 1 1 1 1 0 1 1 1 0 0 0 0 0 1 1 1 1 1 0 1 0 1
 1 1 1 1 0 1 0 1 1 0 1 0 1 1 1 1 1 1 1 1 0 1 1 1 0 1 0 1 0 1 1 1 0 1 0 1 0
 1 0 1 1 1 0 0 0 0 0 0 1 0 0 1 0 1 0 1 1 0 1 1 1 1 1 1 1 0 0 1 1 1 0 0 1
 1 0 1 1 0 1 1 1 1 1 1 1 1 1 1 1 1 0 1 0 0 1 1 0 0 1 1 1 1 1 0 1 1 1 0 0 0
 1 1 0 0 1 1 1 1 0 0 1 1 0 0 0 1 0 0 1 1 1 1 0 1 0 0 1 1 1 1 1 1 0 1 1 0 1
 1 0 0 1 1 0 1 0 1 0 1]
```

1 Represents 'benign', 0 represents 'malign'.

Create and fit the model

In [7]:

```
clf = svm.SVC()
clf.fit(x_train, y_train)
```

Out[7]: `SVC()`

Predict

In [8]:

```
y_pred = clf.predict(x_test)
```

In [9]:

```
acc = metrics.accuracy_score(y_test, y_pred)
print(acc)
```

```
0.9210526315789473
```

In Tim's video the `acc` value was crap (approx 50%) because the lack of parameters passed to SVC().

I don't know why in my case the result is so good (93%)!

Let's try to repeat the process and see what happens:

In [10]:

```
x_train, x_test, y_train, y_test = sklearn.model_selection.train_test_split(x
clf = svm.SVC()
clf.fit(x_train, y_train)
y_pred = clf.predict(x_test)
acc = metrics.accuracy_score(y_test, y_pred)
print(acc)
```

```
0.9385964912280702
```

Even better!!!

It may very well be the case that sklearn improved its performance with default parameters.

Reading some comments in the video, it seems that what I had guessed is correct:

> LACIEE mai:
> 1 week ago
> I have the same thing. that's probably because gamma default value was changed from "auto" to "scale" (I found that in documentation). if you try svm.SVC(gamma="auto") you'll have a low accuracy as Tim has.

Let's check it out:

In [11]:
```
x_train, x_test, y_train, y_test = sklearn.model_selection.train_test_split(x
clf = svm.SVC(gamma="auto")
clf.fit(x_train, y_train)
y_pred = clf.predict(x_test)
acc = metrics.accuracy_score(y_test, y_pred)
print(acc)
```

```
0.631578947368421
```

Ok great, now that we are on the same page as Tim, let's keep following the tutorial:

In [12]:
```
x_train, x_test, y_train, y_test = sklearn.model_selection.train_test_split(x
clf = svm.SVC(gamma="auto", kernel="linear")
clf.fit(x_train, y_train)
y_pred = clf.predict(x_test)
acc = metrics.accuracy_score(y_test, y_pred)
print(acc)
```

```
0.9385964912280702
```

That is a great result comparing with the 61% from the previous attempt.

In [13]:
```
# Polynomial is a good kernel (maybe) but Tim says it takes a lot of time to
#   while the previous attempts were a matter of a few seconds
if False:
    import time
    x_train, x_test, y_train, y_test = sklearn.model_selection.train_test_spl
    clf = svm.SVC(gamma="auto", kernel="poly")
    start = time.time()
    clf.fit(x_train, y_train)
    print("fit time: ", time.time() - start)
    y_pred = clf.predict(x_test)
    acc = metrics.accuracy_score(y_test, y_pred)
    print(acc)
```

That just never ends... I killed it.

If I remove `gamma="auto"` it runs very fast, since Tim's tutorial was in 2019, it seems that sklearn has improved a lot since...

Now let's see the *soft margin* option (parameter `C`):

In [14]:
```
x_train, x_test, y_train, y_test = sklearn.model_selection.train_test_split(x
clf = svm.SVC(gamma="auto", kernel="linear", C=2)
clf.fit(x_train, y_train)
y_pred = clf.predict(x_test)
acc = metrics.accuracy_score(y_test, y_pred)
print(acc)
```

```
0.9736842105263158
```

The accuracy improves, but it is also slower to train (I didn't measure but felt it).

Compare with KNN

In [15]:
```
knn = KNeighborsClassifier(n_neighbors=9)
knn.fit(x_train, y_train)
knn_y_pred = knn.predict(x_test)
knn_acc = metrics.accuracy_score(y_test, knn_y_pred)
print(knn_acc)
```

```
0.956140350877193
```

95% is ok!

Still SVM is usually better, I should tweak around the parameters to see how much I can improve SVM.

In [16]:
```
clf = svm.SVC(kernel="linear", C=1)
clf.fit(x_train, y_train)
y_pred = clf.predict(x_test)
acc = metrics.accuracy_score(y_test, y_pred)

print(acc)
```

```
0.9736842105263158
```

And it worked much better!