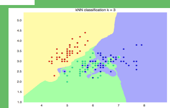
CISC 5352 FINANCIAL PROGRAMMING AND DATA ANALYTICS LECTURE NOTE (6)

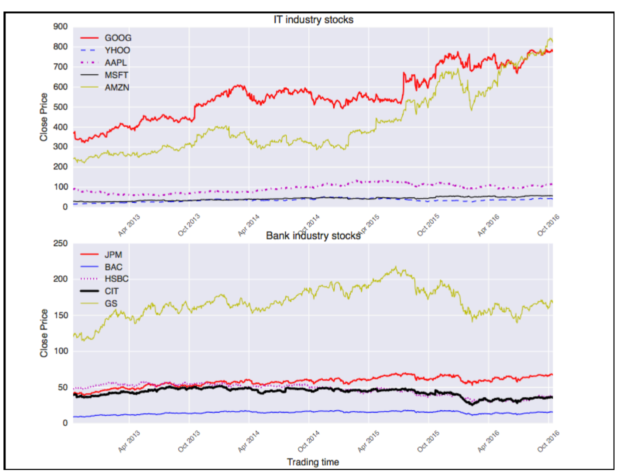


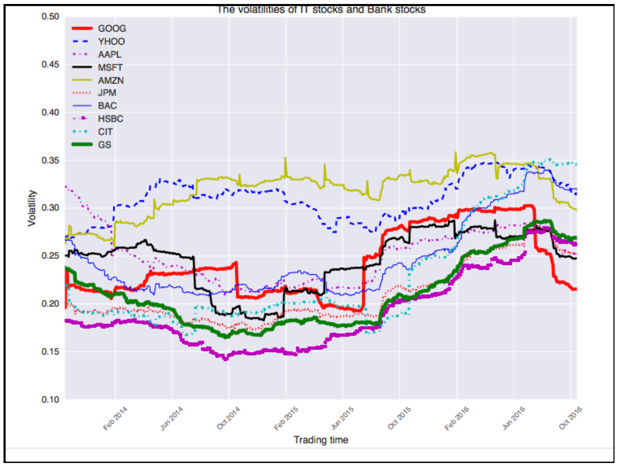
Henry Han Ph.D.
Department of Computer and Information Science
Fordham University, New York NY 10023

Homework 1 feedback

All students did not give correct plots in visualization!

The key is to let people understand/distinguish your data instead of draw all data

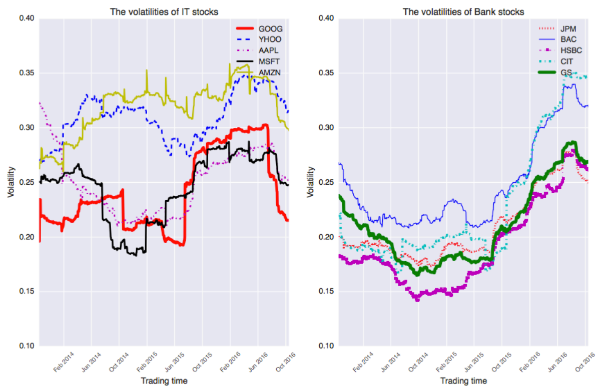




Corresponding code segments

```
## plot 2: volatility of two type of stocks

fig2 = pylab.figure(figsize = (10,6))

pylab.plot(stock_data['GOOG']['Volatility'],  'r-',  label='GOOG', linewidth=3.5)
pylab.plot(stock_data['YHOO']['Volatility'],  'b--', label='YHOO', linewidth=2.0)
pylab.plot(stock_data['AAPL']['Volatility'],  'm-.', label='AAPL', linewidth=2.0)
pylab.plot(stock_data['MSFT']['Volatility'],  'k-',  label='MSFT', linewidth=2.2)
pylab.plot(stock_data['AMZN']['Volatility'],  'y-',  label='AMZN', linewidth=2.0)

pylab.plot(stock_data['JPM']['Volatility'],   'r:',  label='JPM', linewidth=2.5)
pylab.plot(stock_data['BAC']['Volatility'],   'b-',  label='BAC', linewidth=1.0)
pylab.plot(stock_data['HSBC']['Volatility'],  'm-.o', label='HSBC', markersize=4,
linewidth=1.0)
pylab.plot(stock_data['CIT']['Volatility'],   'c-.', label='CIT', linewidth=3.0)
pylab.plot(stock_data['GS']['Volatility'],    'g-',  label='GS', linewidth=4.0)
```

```
pylab.legend(loc='upper left')
pylab.xlabel('Trading time')
pylab.setp(pylab.gca().get_xticklabels(), FontSize=8, rotation=45)
pylab.ylabel('Volatility')

pylab.ylim(0.1,0.5)
pylab.title('The volatilities of IT stocks and Bank stocks')

# save figure
filename2 = 'ITandBankStockVolatility.eps'
fig2.savefig(filename2, dpi=300)
print(" " + filename2 + "  is saved!\n")
```



All codes can be found in cisc5352.lecture.6.demoDataRetrivalVisAndVol.py

Update your homework visualization part by using the visualization codes I provide

Last class review

Last class review

- Implied volatility prediction methods: model based approaches (The model is assumed as BSM model
  1. Bisection (no derivative needed $q_A=1$)
  2. Muller-bisection (no derivative needed $q_A=1.84$)
  3. Newton (need f' (vega in BSM)→ Quadratic convergence $q_A=2$)
  4. Halley's Irrational Formula (need f'' (vomma)→ cublic convergence $q_A=3$)

Last class review

- Implied volatility prediction methods: model based approaches (The model is assumed as BSM model
  1. Bisection (no derivative needed $q_A=1$)
  2. Muller-bisection (no derivative needed $q_A=1.84$)
  3. Newton (need f' (vega in BSM)→ Quadratic convergence $q_A=2$)
  4. Halley's Irrational Formula (need f'' (vomma)→ cublic convergence $q_A=3$)

Note: In the real implied volatility prediction, it does not mean the larger q_A will be fast. Initial points selection or even fit of data may also play a role.

Max (Yanzhe) Li 's codes for Newton Method (I edited a little bit)

It's written in a class way.

I will post your codes if they are good.

See *cisc5352.Lecture.6.MaxLi.NewtonMethodIVPrediction.py for all details*

```python
class bsmNewtonMethod():
    def __init__(self, S, K, T, r, sigma, cStar, optionType, iter):
        self.S = S
        self.K = K
        self.T = T
        self.r = r
        self.sigma = sigma
        self.cStar = cStar

        self.optionType = optionType
        self.iter     = iter


    def bsmVega(self):
        d1 = (e.log(self.S / self.K) + (self.r + 0.5 * self.sigma ** 2) * self.T) / (self.sigma * e.sqrt(self.T))
        vega = self.S * stats.norm.pdf(d1) * e.sqrt(self.T)
        return vega
```

The self in the constructor is equivalent to this in C++

Code segments

```python
##########################################################
## newton method for implied volatility prediction
##########################################################

    def bsmIVprediction(self):
        max_iter     = self.iter
        tolerance    = 0.000000001
        for i in range(max_iter):
            f      = self.bsmValue() - self.cStar  # objective function
            f_prime = self.bsmVega()               # compute f_prime

            old_sigma  = self.sigma
            self.sigma = self.sigma - f/f_prime
            if (e.fabs(self.sigma - old_sigma) < tolerance):
                print("total {:d}".format(i) + " iterations in newton method\n")
                return self.sigma

        return self.sigma
```

Most students miss these part including Max. It is probably because I did not include it in previous newton method codes. Such a tolerance is $|X_{n+1} - X_n|$

```
Today's date is 2016-10-12
Wait, Newton method is predicting option price for you...
total 2 iterations in newton method

Here is a call option.
The strike price is $16.00 and option price is $0.80.
The predicted implied volatility is --> 26.66%

total 3 iterations in newton method

Here is a call option.
The strike price is $17.00 and option price is $0.38.
The predicted implied volatility is --> 26.04%

total 2 iterations in newton method

Here is a put option.
The strike price is $16.00 and option price is $0.74.
The predicted implied volatility is --> 30.28%

total 3 iterations in newton method

Here is a put option.
The strike price is $17.00 and option price is $1.32.
The predicted implied volatility is --> 30.01%
```

Code output

Can you improve it further by using another tolerance to code Newton method?

That is, the objective function fabs(f) < 0.000001 (|f| <0.000001)

What are the pros and cons to use such a tolerance in newton method?

Can you improve it further by using another tolerance to code Newton method?

That is, the objective function fabs(f) < 0.000001 (|f| <0.000001)

What are the pros and cons to use such a tolerance in newton method?

Do we need to include both in Newton method coding?

Do we need to include both in Newton method coding?

Yes, we should: we will know which points Newton methods can't converge and skip the possible re-setting for the initial point

Last class review cont'd

K-NN method
Instance method but with good performance

It is a classification method and also a regression method.

What are the differences between classification and regression?

What are the differences between classification and regression?

Classification: the decision function f(x) outputs predicted labels of test data (e.g. 1 (stock price up), 0 (stock price down)

Regression: the decision function outputs an exact value for test data (e.g. the predicted stock price)

We need to use kNN regression for Implied volatility prediction

Sklearn is a good machine learning library written in python. It also has a corresponding spark-version

https://pypi.python.org/pypi/spark-sklearn

To use k-NN from Sklearn, you need to include k-NN modules as follows

```
from sklearn import neighbors
```

Conduct k-NN via sklearn.neighbors

Three steps

1. Specify k-NN structures/parameters (the value of k, distance, algorithm…)

k: you can try to find the optimal k but there is no method applied to all data

algorithm: you can let kNN make an auto decision according to data: algorithm='auto'

Weights: 'uniform' or 'distance' (distance is recommended in most cases)
distance is by default is Euclidean distance, you can also use more general Minkovsi distance or others

$$\mathrm{Dis}_p(\mathbf{x},\mathbf{y}) = \left(\sum_{j=1}^{d} |x_j - y_j|^p\right)^{1/p} = ||\mathbf{x}-\mathbf{y}||_p$$

KNeighborsClassifier(n_neighbors, weights, algorithm)

Conduct k-NN via sklearn.neighbors Cont'd

Three steps

2. Training:
   fit(training_data, training_data_label)

3. Test (prediction)
   test_data_label=predict(test_data)

```
from sklearn import neighbors

training_data  = [[10], [11], [12], [30], [40], [88]]
training_label = [0, 0, 0, 1, 1, 1]

kNN = neighbors.KNeighborsClassifier(n_neighbors=3)
kNN.fit(training_data, training_label)

test_data =[[15.8], [98.38]]

test_data_label=kNN.predict(test_data)
print("The pedicted labels for test data is:\n")
print(test_data_label)
```

KNN for IRIS data

IRIS data:

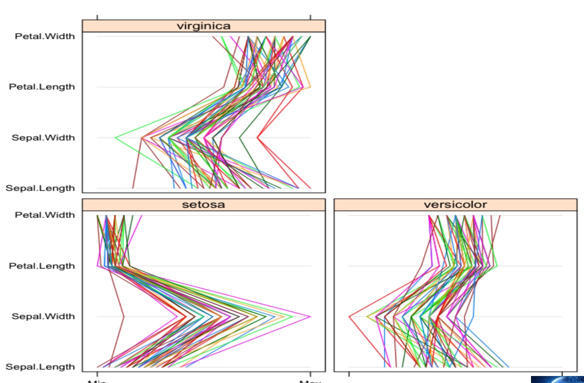
- It gives the measurements in centimeters of the variables <u>Sepal length and width and petal length and width</u>, respectively, for 50 flowers from each of 3 species of iris: *setosa*, *versicolor*, and *virginica*.
- **150 observations and 5 variables**
- **The 5th variable is not a numeric type: 4 useful variables**
  - **n=150 and p=4**

The first 6 samples

	Sepal.Length	Sepal.Width	Petal.Length	Petal.Width	Species
1	5.1	3.5	1.4	0.2	setosa
2	4.9	3.0	1.4	0.2	setosa
3	4.7	3.2	1.3	0.2	setosa
4	4.6	3.1	1.5	0.2	setosa
5	5.0	3.6	1.4	0.2	setosa
6	5.4	3.9	1.7	0.4	setosa

Total data summary

Sepal.Length	Sepal.Width	Petal.Length	Petal.Width	Species
Min. :4.300	Min. :2.000	Min. :1.000	Min. :0.100	setosa :50
1st Qu.:5.100	1st Qu.:2.800	1st Qu.:1.600	1st Qu.:0.300	versicolor:50
Median :5.800	Median :3.000	Median :4.350	Median :1.300	virginica :50
Mean :5.843	Mean :3.057	Mean :3.758	Mean :1.199	
3rd Qu.:6.400	3rd Qu.:3.300	3rd Qu.:5.100	3rd Qu.:1.800	
Max. :7.900	Max. :4.400	Max. :6.900	Max. :2.500	



Data visualization (R package used)

We check the performance of k-NN for this data set

```
import numpy as np
import matplotlib.pyplot as plt
from matplotlib.colors import ListedColormap
from sklearn import neighbors
from sklearn.datasets  import load_iris
import time
import seaborn as sb
```

cisc5352.lecture.6.demoKNN.py
Some Credits to https://docs.scipy.org/doc/

```
# import some data to play with
# The typical sample data in machine learning/ data minining
# is IRIS data

iris = load_iris()

# It is Bunch object-- a dict object
#  attributes:
#  data: samples

# feature names:
#
#   ['sepal length (cm)',
#   'sepal width (cm)',
#   'petal length (cm)',
#   'petal width (cm)'],


#  target: label of each sample-->  (0,1,2)

#  targe_names: label names:
#           total three (label names):
#           'setosa', (0) 'versicolor' (1), 'virginica' (2)
#
```

```
print("total row numbers in data:
{:d}".format(iris.data.size))
print("\n checking this sample data\n")
time.sleep(1)
print(iris)
time.sleep(2)

n=10
print("\n The first {:d}".format(n) + "
samples \n")
print(iris.data[0:n,0:n])
time.sleep(1)

print("\n Their corresponding label
information\n")
print(iris.target[0:n])
time.sleep(1)

print("\n")
```

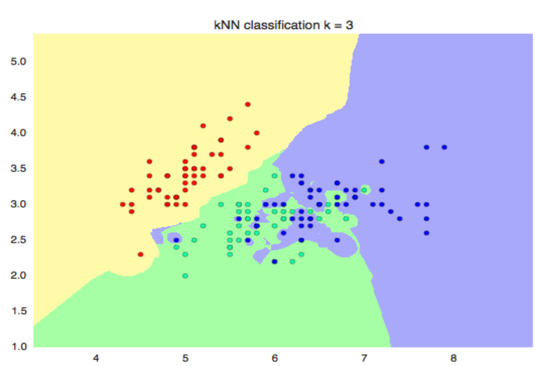
```
###########################################
## data and targets for k-NN
## training data and training data label
## Also represented as X and y
###########################################

training_data          = iris.data[:, 0:2]  # we only take the first two features
training_data_label = iris.target


# Create color maps

cmap_light = ListedColormap(['#FFFAAA', '#AAFFAA', '#AAAAFF']) # plot decision regions

cmap_bold = ListedColormap(['#FF0000', '#00FFAA', '#0000FF'])  # plot training data
```

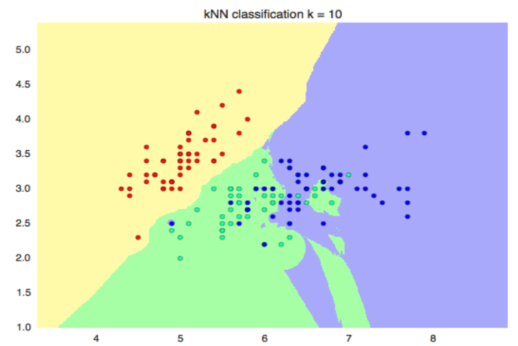
```
####################
# k to be selected
####################

k_list=[3,5,7,10, 15]


## create test data
x_min, x_max = training_data[:, 0].min() - 1, training_data[:, 0].max() + 1
y_min, y_max = training_data[:, 1].min() - 1, training_data[:, 1].max() + 1
h = .01
xx, yy = np.meshgrid(np.arange(x_min, x_max, h), np.arange(y_min, y_max, h))

Test_data = np.c_[xx.ravel(), yy.ravel()]
print("\n This is test data:\n")
print(Test_data)
time.sleep(2)
```

```
##################################################
## k_NN for different k values
##################################################

for nb in k_list:
## 1. specify the strcutures/parameters of KNN
   kNN = neighbors.KNeighborsClassifier(nb, weights='distance', algorithm='auto')

## 2. training kNN with training data
   kNN.fit(training_data, training_data_label)

## 3. prediction for test data
   test_data_labels = kNN.predict(Test_data)
   print("\nUnder k={:d}".format(nb) + " the predicted labels are\n")
   print("\t" + str(test_data_labels) + "\n")
   time.sleep(1)
```

```
# NOTE: still in the for loop block

# Put the classification result into a color plot
    test_data_labels = test_data_labels.reshape(xx.shape)
    plt.figure()
    plt.pcolormesh(xx, yy, test_data_labels, cmap=cmap_light)

    # Plot training data
    plt.scatter(training_data[:, 0], training_data[:, 1], c=training_data_label, cmap=cmap_bold)
    plt.xlim(xx.min(), xx.max())
    plt.ylim(yy.min(), yy.max())
    plt.title('kNN classification k = '+ '{:d}'.format(nb) )

plt.show()
```

Which k will lead to a better classification (checking your output)?





How about k-NN regression?

KNeighborsRegressor(n_neighbors, weights, algorithm='auto',…)

Its parameters are same as those of k-NN classification

```
from sklearn.neighbors import KNeighborsRegressor

training_data  = [[10], [11], [12], [30], [40], [88]]
training_response = [0., 0., 0.14, 1.0, 1.1, 1.5]


kNN = KNeighborsRegressor(n_neighbors=3)
kNN.fit(training_data, training_response)

test_data =[[57.27], [20.88]]


test_data_response=kNN.predict(test_data)
print("The pedicted response for test data is:\n")
print(test_data_response)
```

How can we employ k-NN to predict implied volatility?

Suppose we have data like

Strike	Expiry	Type	Symbol	Last	Bid	Ask	Chg	PctChg	Vol	Open_Int	IV	Root	Nonstandar	Underlying	derlying_Pr	Quote_Time
60	2015-10-16 00:00:00	call	AAPL151016C00060000	67.6	65.05	68.4	0	0.00%	19	619	58.20%	AAPL	0	AAPL	126.6	2015-06-20 00:00:00
		put	AAPL151016P00060000	0.01	0	0.02	0	0.00%	5	2252	47.27%	AAPL	0	AAPL	126.6	2015-06-20 00:00:00
65	2015-10-16 00:00:00	call	AAPL151016C00065000	65.86	60.1	62.15	0	0.00%	1	25	66.65%	AAPL	0	AAPL	126.6	2015-06-20 00:00:00
		put	AAPL151016P00065000	0.05	0.01	0.04	0	0.00%	5	611	45.70%	AAPL	0	AAPL	126.6	2015-06-20 00:00:00
70	2015-10-16 00:00:00	call	AAPL151016C00070000	59.6	55.1	56.9	0	0.00%	2	111	53.91%	AAPL	0	AAPL	126.6	2015-06-20 00:00:00
		put	AAPL151016P00070000	0.05	0.03	0.06	0	0.00%	2	1579	42.97%	AAPL	0	AAPL	126.6	2015-06-20 00:00:00
75	2015-10-16 00:00:00	call	AAPL151016C00075000	55.5	50.05	51.85	0	0.00%	2	32	46.88%	AAPL	0	AAPL	126.6	2015-06-20 00:00:00
		put	AAPL151016P00075000	0.08	0.06	0.08	0	0.00%	10	1857	39.84%	AAPL	0	AAPL	126.6	2015-06-20 00:00:00
80	2015-10-16 00:00:00	call	AAPL151016C00080000	46.8	46.25	46.95	-2.9	-5.84%	2	293	44.29%	AAPL	0	AAPL	126.6	2015-06-20 00:00:00
		put	AAPL151016P00080000	0.1	0.08	0.11	-0.04	-28.57%	4	2844	36.91%	AAPL	0	AAPL	126.6	2015-06-20 00:00:00
85	2015-10-16 00:00:00	call	AAPL151016C00085000	43.05	41.5	41.8	0	0.00%	36	539	35.55%	AAPL	0	AAPL	126.6	2015-06-20 00:00:00
		put	AAPL151016P00085000	0.13	0.13	0.15	0	0.00%	12	3769	34.08%	AAPL	0	AAPL	126.6	2015-06-20 00:00:00

Stock_Price	Strike_Price	Time_to_Ma	Interest_Rat	Option_Price	Option_Type	Volatility	nplied Volatility
21.67	20	182	0.0026	2.4	0	0.40376384	0.419

from sklearn.neighbors import KNeighborsClassifier

kNN = KNeighborsClassifier(n_neighbors=5, weights='distance', algorithm='auto')

kNN.fit(train_data, train_data_response)

Does this one work?

test_data_response= kNN.predict(test_data)

	Stock_Price	Strike_Price	Time_to_Ma	Interest_Rat	Option_Price	Option_Type	Volatility	nplied Volatility
2	29.21	30	35	0.0026	2.5	0	0.17797755	0.6099
3	29.21	35	35	0.0026	0.17	0	0.17797755	0.4815
4	29.21	30	126	0.0026	2.89	0	0.17797755	0.3247
5	29.21	35	126	0.0026	0.22	0	0.17797755	0.3066
6	29.21	30	126	0.0026	1.1	1	0.17797755	0.6519
7	29.21	35	126	0.0026	5.6	1	0.17797755	0.6636

which variables should we input to a machine learning model is essential for the success of prediction!

Feature engineering is not recommended generally.

It will enhance learning results for certain data, but it may decrease the generalization problem or lead to overfitting.

Overfitting: a learning machine is only good at few datasets

Support vector machine (SVM) (1995) dominated machine learning more than 10 years!

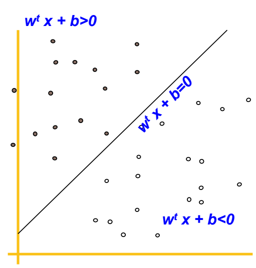
Main idea: seek to construct an optimal hyperplane in a high-dimensional space with kernel trick learning tricks.

Its applications can be found in ANYWHERE! From trading, text mining to disease diagnosis, web mining, pattern recognition.

SVM has a clear geometric interpretation

denotes +1 (stock up)

denotes -1 (stock down)



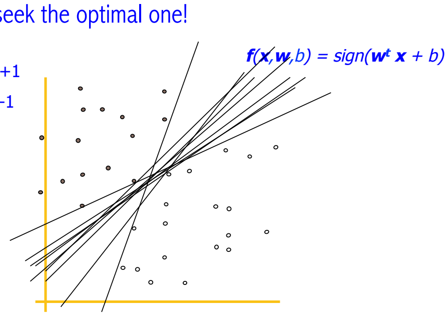
Classification in 2D→ a find an optimal line to partition two groups of data.

If we extend it to dimesnional >=3, it means we need to find an optimal hyperlane to partition data

There are many hyperplanes to partition data, we only seek the optimal one!

denotes +1

denotes -1



SVM decision function: a linear function

$f(\mathbf{x},\mathbf{w},b) = sign(\mathbf{w}^t\,\mathbf{x} + b)$

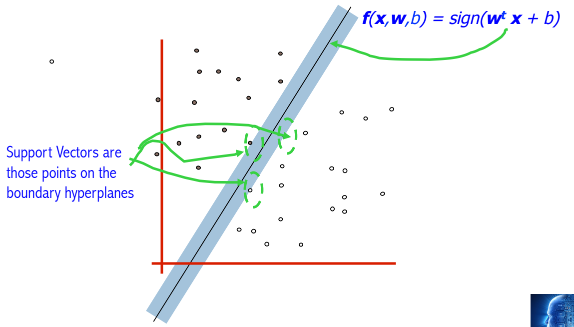
If we remove sign→ $f(x,w,b)=w^tx+b$ is the prediction function for the its regression version

How to get the optimal hyperplane?

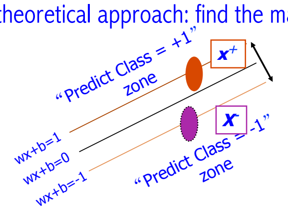
How to get the optimal hyperplane?

Seek the maximum margin!

Maximum Margin: the maximum distance between two hyperplanes seperating data



More theoretical approach: find the maximum margin



① $w^t . x^+ + b = +1$

② $w^t . x^- + b = -1$

③ $w^t . (x^+ - x^-) = 2$

$$M = \frac{(x^+ - x^-) \cdot w}{\| w \|} = \frac{2}{\| w \|}$$

M=Margin Width

We need the optimal hyperplane: classify two g groups of data with the maximum margin $M = \frac{2}{\| w \|}$

- This is equivalent to minimize $\min \Phi(w) = \frac{1}{2} w' w$

  under the conditions $y_i (w x_i + b) \geq 1$

y_i is the label for x_i and it only had two possible values {+1, -1}

How to solve this nonlinear programming problem (quadratic programming problem)

How to solve this nonlinear programming problem (quadratic programming problem)?

It is a well-known mathematical programming problem

Many classic algorithms are available

Lagrange multiplier methods: Solve a corresponding dual problem by using *Lagrange multipliers* α *for each constraint*

Find $\alpha_1 \ldots \alpha_N$ such that
$Q(\alpha) = \Sigma \alpha_i - ½ \Sigma \Sigma \alpha_i \alpha_j y_i y_j \mathbf{x_i}^T \mathbf{x_j}$ is maximized

Under the following conditions
(1) $\Sigma \alpha_i y_i = 0$
(2) $\alpha_i \geq 0$

SVM Solution

The solution has the form:

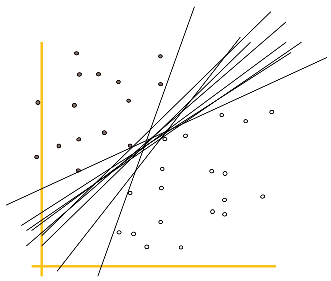
$\boxed{\mathbf{w} = \Sigma \alpha_i y_i \mathbf{x_i} \qquad b = y_k - \mathbf{w}^T \mathbf{x_k} \text{ for any } \mathbf{x_k} \text{ such that } \alpha_k \neq 0}$

Each non-zero α_i indicates that corresponding $\mathbf{x_i}$ is a support vector

Then the classifying function will have the form:

$\boxed{f(\mathbf{x}) = \Sigma \alpha_i y_i \mathbf{x_i}^T \mathbf{x} + b}$

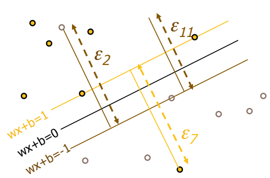
The previous margin is called hard margin that assumes data is linearly separable: we can find a hyperplane to partition data completely (100%)



linearly separable data.
Not all data are linearly separable data

Soft Margin is needed for

Slack variables ξ_i can be added to allow misclassification of difficult or noisy examples. The current optimization poblem is changed a little bit



Find $\mathbf{w}$ and b such that
$\mathbf{\Phi}(\mathbf{w}) = \frac{1}{2}\, \mathbf{w}^T\mathbf{w} + C\Sigma\xi_i$ is minimized and for all $\{(\mathbf{x}_i, y_i)\}$
$y_i\,(\mathbf{w}^T\mathbf{x}_i + b) \geq 1 - \xi_i$ and $\xi_i \geq 0$ for all i

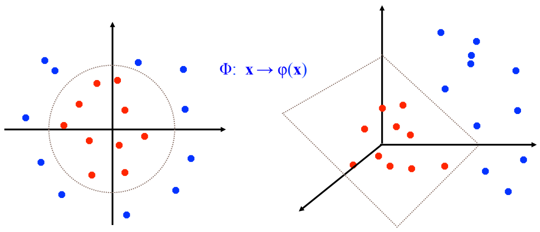
How about nonlinear data?

We can't just see we can separate or not→their nonlinear property prevents the "separable".

The best contribution of SVM to machine learning is to solve this problem using kernel tricks

SVM maps nonlinear data to a high-dimensional feature space to conduct classification

We don't need to know the mapping function Φ but we can still do classification in the high-dimensional space.



The high-dimensional space does not bring high complexity. All computing can be done in input space via using kernel tricks.

Any input data point is mapped into high-dimensional space via a transformation Φ: $x \rightarrow \phi(x)$, we can use a kernel function to evaluate computing in the feature space by assume all computing can be written in inner-product forms. The correctness of such an approach can be guaranteed by Mercer's theorem

$K(x_i,x_j) = \phi(x_i)^T \phi(x_j)$

All kernel matrices should be semi-positive definite matrices.

General Kernel Functions (you can build you own kernels)

- Linear: $K(x_i,x_j) = x_i^T x_j$
- Polynomial of power p: $K(x_i,x_j) = (1 + x_i^T x_j)^p$
- Gaussian ('rbf' kernel):

$$K(\mathbf{x_i}, \mathbf{x_j}) = \exp\left(-\frac{\|\mathbf{x_i} - \mathbf{x_j}\|^2}{2\sigma^2}\right)$$

- Sigmoid: $K(x_i,x_j) = \tanh(\beta_0 x_i^T x_j + \beta_1)$

Non-linear SVMs

- Dual problem formulation:

Find $\alpha_1 \ldots \alpha_N$ such that
$Q(\alpha) = \Sigma\alpha_i - \frac{1}{2}\Sigma\Sigma\alpha_i\alpha_j y_i y_j K(x_i, x_j)$ is maximized and
(1) $\Sigma\alpha_i y_i = 0$
(2) $\alpha_i \geq 0$ for all α_i

- The solution is:

$$f(x) = \Sigma\alpha_i y_i K(x_i, x_j) + b$$

```
from sklearn import svm

X = [[0, 0], [-2,0], [1, 1], [10,1]]
y = [0, 0,1,1]
clf = svm.SVC()  # default kernel ‘rbf’
clf.fit(X, y)

predicted_label=clf.predict([[2.5, 8.]])
print("\n The predicted label is--->" +str(predicted_label))
```