Machine Learning
An introduction of applications

Diep Nguyen, Ph.D.
Computer Science department
Faculty of Information and Technology, UET-VNU

Mail-to: ngocdiep@vnu.edu.vn

Applications in this lecture

- Finding similar users based on their preferences
- Recommending items to a user
- Building “customer viewed this also viewed” item recommendation
- Grouping things that people wants
- Discovering the buying patterns
- Predict gas consumptions
- Predict the quality of white wine
- Detect the wine class

Steps in developing a machine learning application

1. Collect data
2. Prepare the input data → make sure it's in a useable format
3. Analyze the input data (empty values, any patterns)
4. Define the evaluation metrics
5. Train the algorithm
6. Test the algorithm (evaluate using metrics)
7. Use it

Collaborative filtering

Sample applications:
Finding similar users based on their preferences
Recommending items to a user
Building "customer viewed this also viewed" item recommendation

Collaborative Filtering

- We often seek recommendations from people we trust (family, friends, colleagues, experts)
- But who to ask for a recommendation is also based on similarity in taste
  - People with my tastes are likely to recommend things I will like
- CF searches a large group of people for those that like the things you like, and it combines the things they like into a ranked list of suggestions

Movie ratings

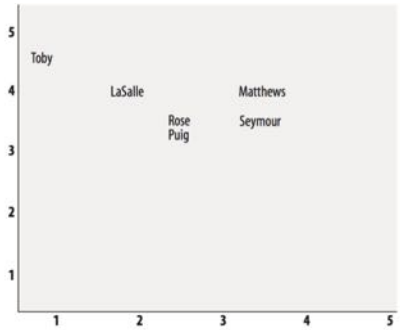
movies

critics

	Lady in the Water	Snakes on a Plane	Just My Luck	Superman Returns	You, Me and Dupree	The Night Listener
Lisa Rose	2.5	3.5	3.0	3.5	2.5	3.0
Gene Seymour	3.0	3.5	1.5	5.0	3.5	3.0
Michael Phillips	2.5	3.0	NaN	3.5	NaN	4.0
Mick LaSalle	3.0	4.0	2.0	3.0	2.0	3.0
Jack Matthews	3.0	4.0	NaN	5.0	3.5	3.0
Claudia Puig	NaN	3.5	3.0	4.0	2.5	4.5
Toby	NaN	4.5	NaN	4.0	1.0	NaN

Finding similar users

- A way to determine how similar people are in their tastes
- *Similarity score*



Euclidean distance

$$d(\mathbf{p},\mathbf{q}) = d(\mathbf{q},\mathbf{p}) = \sqrt{(q_1-p_1)^2 + (q_2-p_2)^2 + \cdots + (q_n-p_n)^2}$$
$$= \sqrt{\sum_{i=1}^{n}(q_i-p_i)^2}.$$

Example:

```
# euclidean distance between two vectors
def euclidean_distance(p, q):
    from math import sqrt
    sum_pow = sum([pow(p[i] - q[i], 2) for i in range(len(p))])
    return sqrt(sum_pow)

toby = (1.0, 4.5)
rose = (2.5, 3.5)
puig = (2.5, 3.5)
rose_toby = euclidean_distance(rose, toby)
rose_puig = euclidean_distance(rose, puig)

print("distance from rose to toby: ", rose_toby)
print("distance from rose to puig: ", rose_puig)
```

```
distance from rose to toby:  1.8027756377319946
distance from rose to puig:  0.0
```

Similarity score

```
# we want a function that returns higher values for those are similar
def similarity(p, q):
    dist = euclidean_distance(p, q)
    return 1/(1 + dist) # why 1+?
```

```
# calculate how much similar
rose_toby = similarity(rose, toby)
rose_puig = similarity(rose, puig)

print("similarity between rose and toby: ", rose_toby)
print("similarity between rose and puig: ", rose_puig)
```

```
similarity between rose and toby:  0.3567891723253309
similarity between rose and puig:  1.0
```

Ranking the Critics

- Given a person, finds the closest matches
- As known as k-nearest neighbors

Calculate the similarities

Rank similarities

Return n highest scores

```
def topMatches(prefs,person,n=5,similarity=sim_euclidean):
    scores=[(similarity(prefs,person,other),other)
                for other in prefs if other!=person]

    # Sort the list so the highest scores appear at the top
    scores.sort()
    scores.reverse()
    return scores[0:n]
```

Ranking example

	Lady in the Water	Snakes on a Plane	Just My Luck	Superman Returns	You, Me and Dupree	The Night Listener
Lisa Rose	2.5	3.5	3.0	3.5	2.5	3.0
Gene Seymour	3.0	3.5	1.5	5.0	3.5	3.0
Michael Phillips	2.5	3.0	NaN	3.5	NaN	4.0
Mick LaSalle	3.0	4.0	2.0	3.0	2.0	3.0
Jack Matthews	3.0	4.0	NaN	5.0	3.5	3.0
Claudia Puig	NaN	3.5	3.0	4.0	2.5	4.5
Toby	NaN	4.5	NaN	4.0	1.0	NaN

```
topMatches(critics, "Toby")
```

```
[(0.3076923076923077, 'Mick LaSalle'),
 (0.2857142857142857, 'Michael Phillips'),
 (0.23529411764705882, 'Claudia Puig'),
 (0.2222222222222222, 'Lisa Rose'),
 (0.11764705882352941, 'Jack Matthews')]
```

Dealing with NaN values

Is there any problem?

```
# Returns the best matches for person from the prefs dictionary.
# Number of results and similarity function are optional params.

def sim_euclidean(prefs, person1, person2):
    #get the list of shared_items
    si = {}
    for item in prefs[person1]:
        if item in prefs[person2]: si[item]=1

    # if they have no ratings in common, return 0
    if len(si)==0: return 0

    # Add up the squares of all the differences
    sum_of_squares=sum([pow(prefs[person1][item]-prefs[person2][item],2)
                        for item in prefs[person1] if item in prefs[person2]])

    return 1/(1+sum_of_squares)
```

Which distance/similarity score function?

- Euclidean
- Cosine
- Pearson correlation
- Jaccard coefficient
- Manhattan distance
- ...

Recommending items - Methods

- Method 1: Look at the person whose taste is the most similar and look for items in his but not yet mine
  - No way to rank the items
  - Would miss items that I might like because they are not in the list
  - Too sensitive to outliers
    - If the most similar person strangely liked an item, which had all bad reviews from all others returned by `topMatches`

Recommending items – Methods

- Weighted score for items
  - Take the votes of all the other critics and multiply how similar they are to me by the score they gave each movie

Table 2-2. Creating recommendations for Toby

Critic	Similarity	Night	S.xNight	Lady	S.xLady	Luck	S.xLuck
Rose	0.99	3.0	2.97	2.5	2.48	3.0	2.97
Seymour	0.38	3.0	1.14	3.0	1.14	1.5	0.57
Puig	0.89	4.5	4.02			3.0	2.68
LaSalle	0.92	3.0	2.77	3.0	2.77	2.0	1.85
Matthews	0.66	3.0	1.99	3.0	1.99		
Total			12.89		8.38		8.07
Sim. Sum			3.84		2.95		3.18
Total/Sim. Sum			3.35		2.83		2.53

If use Total, a movie reviewed by more people would have a big advantage

Sum of all the similarities for critics that reviewed that movie

This also implies what my rating for each movie would be

Matching products

Matching products - method

	Lady in the Water	Snakes on a Plane	Just My Luck	Superman Returns	You, Me and Dupree	The Night Listener
Lisa Rose	2.5	3.5	3.0	3.5	2.5	3.0
Gene Seymour	3.0	3.5	1.5	5.0	3.5	3.0
Michael Phillips	2.5	3.0	NaN	3.5	NaN	4.0
Mick LaSalle	3.0	4.0	2.0	3.0	2.0	3.0
Jack Matthews	3.0	4.0	NaN	5.0	3.5	3.0
Claudia Puig	NaN	3.5	3.0	4.0	2.5	4.5
Toby	NaN	4.5	NaN	4.0	1.0	NaN

Flip item and person

	Lisa Rose	Gene Seymour	Michael Phillips	Mick LaSalle	Jack Matthews	Claudia Puig	Toby
Lady in the Water	2.5	3.0	2.5	3.0	3.0	NaN	NaN
Snakes on a Plane	3.5	3.5	3.0	4.0	4.0	3.5	4.5
Just My Luck	3.0	1.5	NaN	2.0	NaN	3.0	NaN
Superman Returns	3.5	5.0	3.5	3.0	5.0	4.0	4.0
You, Me and Dupree	2.5	3.5	NaN	2.0	3.5	2.5	1.0
The Night Listener	3.0	3.0	4.0	3.0	3.0	4.5	NaN

Matching products - example

```
topMatches(movies, "Superman Returns")
```

```
[(0.16666666666666666, 'Snakes on a Plane'),
 (0.10256410256410256, 'The Night Listener'),
 (0.09090909090909091, 'Lady in the Water'),
 (0.06451612903225806, 'Just My Luck'),
 (0.05333333333333334, 'You, Me and Dupree')]
```

```
topMatches(movies, "Superman Returns", similarity=sim_pearson)
```

```
[(0.6579516949597695, 'You, Me and Dupree'),
 (0.4879500364742689, 'Lady in the Water'),
 (0.11180339887498941, 'Snakes on a Plane'),
 (-0.1798471947990544, 'The Night Listener'),
 (-0.42289003161103106, 'Just My Luck')]
```

How can we interpret these negative values?

Collaborative filtering: user-based vs item-based

Users

	Lady in the Water	Snakes on a Plane	Just My Luck	Superman Returns	You, Me and Dupree	The Night Listener
Lisa Rose	2.5	3.5	3.0	3.5	2.5	3.0
Gene Seymour	3.0	3.5	1.5	5.0	3.5	3.0
Michael Phillips	2.5	3.0	NaN	3.5	NaN	4.0
Mick LaSalle	3.0	4.0	2.0	3.0	2.0	3.0
Jack Matthews	3.0	4.0	NaN	5.0	3.5	3.0
Claudia Puig	NaN	3.5	3.0	4.0	2.5	4.5
Toby	NaN	4.5	NaN	4.0	1.0	NaN

Which is more changing?

Flip item and person

Items

	Lisa Rose	Gene Seymour	Michael Phillips	Mick LaSalle	Jack Matthews	Claudia Puig	Toby
Lady in the Water	2.5	3.0	2.5	3.0	3.0	NaN	NaN
Snakes on a Plane	3.5	3.5	3.0	4.0	4.0	3.5	4.5
Just My Luck	3.0	1.5	NaN	2.0	NaN	3.0	NaN
Superman Returns	3.5	5.0	3.5	3.0	5.0	4.0	4.0
You, Me and Dupree	2.5	3.5	NaN	2.0	3.5	2.5	1.0
The Night Listener	3.0	3.0	4.0	3.0	3.0	4.5	NaN

Item-based collaborative filtering

- Precompute the most similar items for each item
  - Build once, reuse every time later
- When making recommendations to a user:
  - Look at his top-rated items
  - Create a weighted list of the items most similar to those

User-based vs item-based recommendation

Table 2-3. Item-based recommendations for Toby

Movie	Rating	Night	R.xNight	Lady	R.xLady	Luck	R.xLuck
Snakes	4.5	0.182	0.818	0.222	0.999	0.105	0.474
Superman	4.0	0.103	0.412	0.091	0.363	0.065	0.258
Dupree	1.0	0.148	0.148	0.4	0.4	0.182	0.182
Total		0.433	1.378	0.713	1.764	0.352	0.914
Normalized			3.183		2.598		2.473

Compared to User-based:

Table 2-2. Creating recommendations for Toby

Critic	Similarity	Night	S.xNight	Lady	S.xLady	Luck	S.xLuck
Rose	0.99	3.0	2.97	2.5	2.48	3.0	2.97
Seymour	0.38	3.0	1.14	3.0	1.14	1.5	0.57
Puig	0.89	4.5	4.02			3.0	2.68
LaSalle	0.92	3.0	2.77	3.0	2.77	2.0	1.85
Matthews	0.66	3.0	1.99	3.0	1.99		
Total			12.89		8.38		8.07
Sim. Sum			3.84		2.95		3.18
Total/Sim. Sum			3.35		2.83		2.53

Need to compute these similarities every time

Exercises

- Using del.icio.us API, create a dataset of tags and items. Use this to calculate similarity between tags and see if you can find any that are almost identical
- Take a look at *http://www.audioscrobbler.net*, a dataset containing music preferences for a large set of users. Use their web services API to get a set of data for making and building a music recommendation system.

Data clustering

Unsupervised algorithm

Sample application: Grouping things that people wants

What clustering is for?

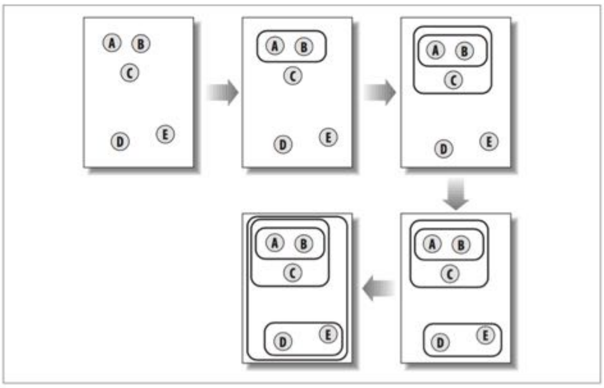
- Discovering groups of things, people, or ideas that are all closely related
- Examples:
  - Detecting groups of customers with similar buying patterns
  - Discovering different styles of fashion
  - In computational biology: Finding groups of genes that exhibit similar behavior, which might indicate that they respond to a treatment in the same way or are part of the same biological pathway

Sample problem: Clusters of Preferences

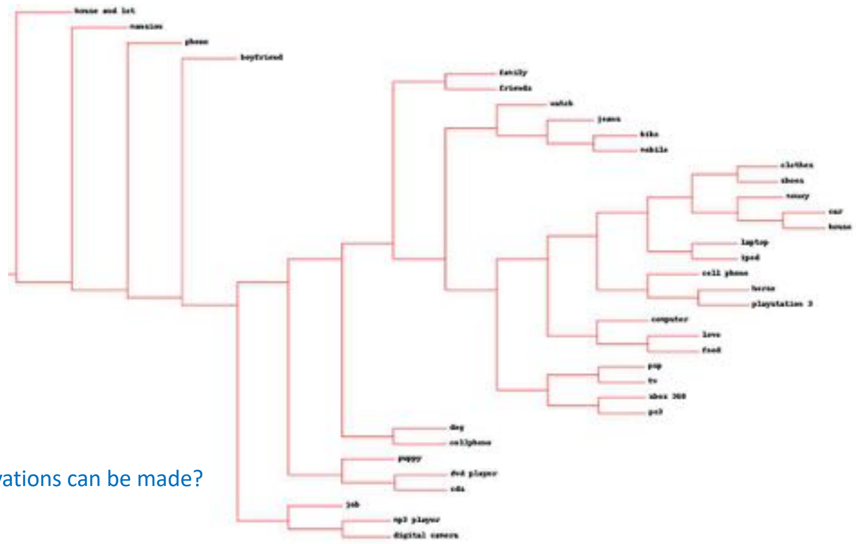
- @Zebo: People make lists of :
  - things that they own
  - things that they would like to own
- Task: Group items by how similarly they are preferred

Hierarchical clustering

- Continuously merge the two most similar groups:
  - Each group starts as a single item
  - In each iteration, the algorithm calculates the distances between every pair of groups, and the closest ones are merged together to form a new group.
  - This is repeated until there is only one group



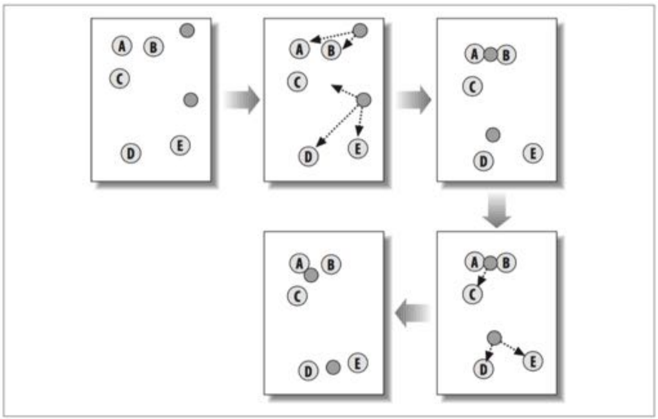
Hierarchical preference clusters



Which observations can be made?

K-means clustering

- Begins with k randomly placed centroids
- Assigns every item to the nearest one
- After the assignment, the centroids are moved to the average location of all the nodes assigned to them
- Repeat until the assignments stop changing



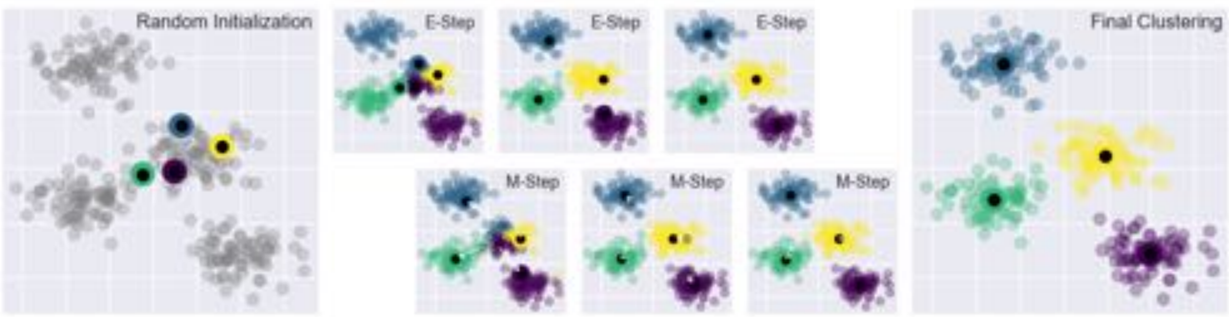
Expectation Maximization

1. Begins with k randomly placed centroids
2. Assigns every item to the nearest one
3. After the assignment, the centroids are moved to the average location of all the nodes assigned to them
4. Repeat until the assignments stop changing

E-step (expectation step): updating our expectation of which cluster each point belongs to

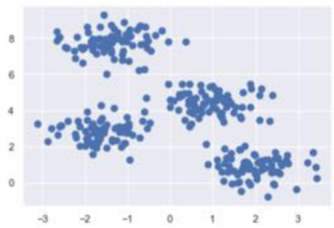
M-step (maximization step): maximizing some fitness function that defines the location of the cluster centers – in this case, it is accomplished by taking a simple mean of the data in each cluster

K-means example

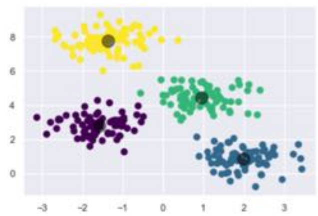


K-means example with different number of clusters

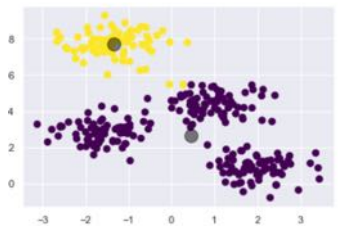
Data



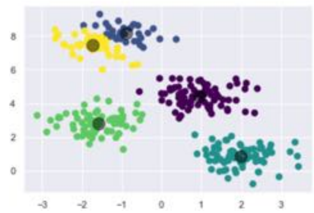
k=4



k=2

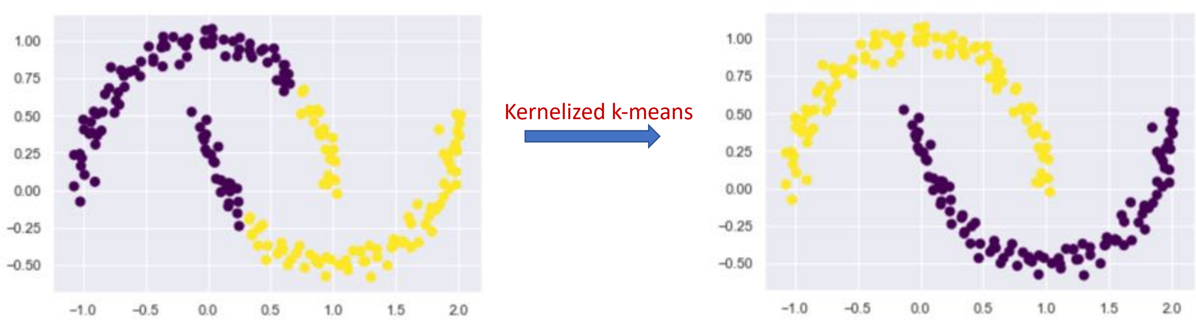


k=5



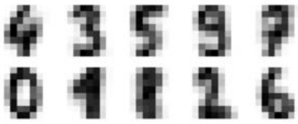
Notes on k-means

- The globally optimal result may not be achieved
- The number of clusters must be selected beforehand
- k-means is limited to linear cluster boundaries



Exercises

- K-means on digits



- K-means for color compression

Credit: PythonDataScienceHandbook

Association rule mining: Apriori algorithm

Sample application: Discovering the buying patterns

Association rule mining

- To identify underlying relations between different items
- For example: Super Market
  - Mothers with babies buy milk together with diapers
  - Bachelors buy beers and chips
- Knowing the relations to do intelligent marketing
  - If we know item A and B are often bought together, what can we do to promote this to customers?

Marketing example

- If item A and B are often bought together
  - A and B can be placed together so that when a customer buys one of the product he doesn't have to go far away to buy the other product.
  - People who buy one of the products can be targeted through an advertisement campaign to buy the other.
  - Collective discounts can be offered on these products if the customer buys both of them.
  - Both A and B can be packaged together.

Apriori algorithm - components

- Support = default popularity of an item

```
Support(B) = (Transactions containing (B))/(Total Transactions)
```

- Confidence = likelihood that an item B is also bought if item A is bought

```
Confidence(A→B) = (Transactions containing both (A and B))/(Transactions containing A)
```

- Lift (A->B) = increase in the ratio of sale of B when A is sold

```
Lift(A→B) = (Confidence (A→B))/(Support (B))
```

Apriori algorithm

1. Set a minimum value for support and confidence. This means that we are only interested in finding rules for the items that have certain default existence (e.g. support) and have a minimum value for co-occurrence with other items (e.g. confidence).
2. Extract all the subsets having higher value of support than minimum threshold.
3. Select all the rules from the subsets with confidence value higher than minimum threshold.
4. Order the rules by descending order of Lift.

Association rule - Example

```
from apyori import apriori

association_rules = apriori(
                        records,
                        min_support=0.0045,
                        min_confidence=0.2,
                        min_lift=3,
                        min_length=2
)
```

```
Rule: chicken -> light cream
Support: 0.0045327289694707 37
Confidence: 0.2905982905982905 7
Lift: 4.84395061728395
```

Number of transactions containing light cream divided by total number of transactions

Out of all the transactions that contain "light cream", 29.05% of the transactions also contain chicken

Chicken is 4.84 times more likely to be bought by the customers who buy light cream compared to the default likelihood of the sale of chicken

Regression

Supervised learning algorithm

Sample applications:

- Predict gas consumptions
- Predict the quality of white wine

Problem statement

- Predict the gas consumptions (in millions of gallons) in 48 US states based upon gas taxes (in cents), per capita income (dollars), paved highways (in miles) and the proportion of population that has a drivers license.

Linear regression

$$y_i = \beta_0 + \beta_1 x_{i1} + \cdots + \beta_p x_{ip} + \varepsilon_i = \mathbf{x}_i^{\mathsf{T}} \boldsymbol\beta + \varepsilon_i, \qquad i = 1, \ldots, n,$$

Matrix form:

$$\mathbf{y} = X\boldsymbol\beta + \boldsymbol\varepsilon,$$

where

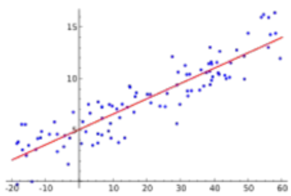
$$\mathbf{y} = \begin{pmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{pmatrix},$$

$$X = \begin{pmatrix} \mathbf{x}_1^{\mathsf{T}} \\ \mathbf{x}_2^{\mathsf{T}} \\ \vdots \\ \mathbf{x}_n^{\mathsf{T}} \end{pmatrix} = \begin{pmatrix} 1 & x_{11} & \cdots & x_{1p} \\ 1 & x_{21} & \cdots & x_{2p} \\ \vdots & \vdots & \ddots & \vdots \\ 1 & x_{n1} & \cdots & x_{np} \end{pmatrix},$$

Intercept

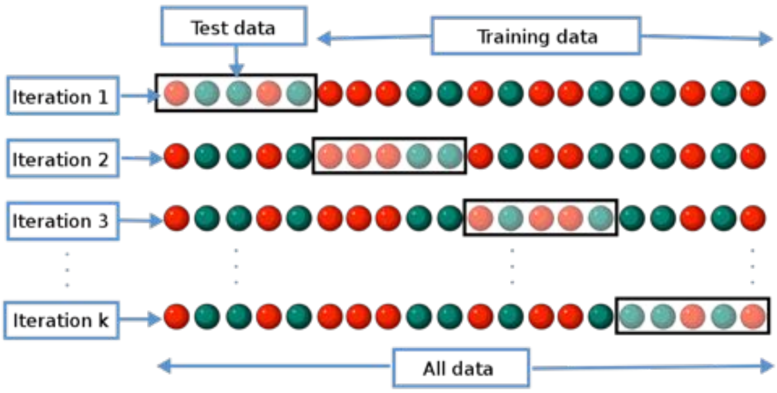
Effects or regression coefficients

$$\boldsymbol\beta = \begin{pmatrix} \beta_0 \\ \beta_1 \\ \beta_2 \\ \vdots \\ \beta_p \end{pmatrix}, \quad \boldsymbol\varepsilon = \begin{pmatrix} \varepsilon_1 \\ \varepsilon_2 \\ \vdots \\ \varepsilon_n \end{pmatrix}.$$



Source: wikipedia

Cross-validation

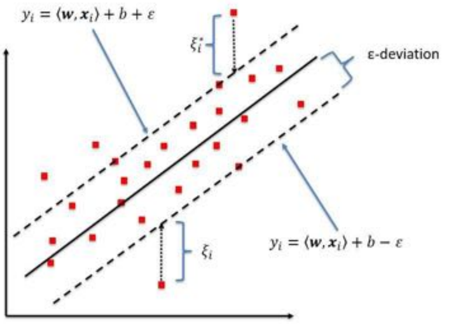


Source: wikipedia

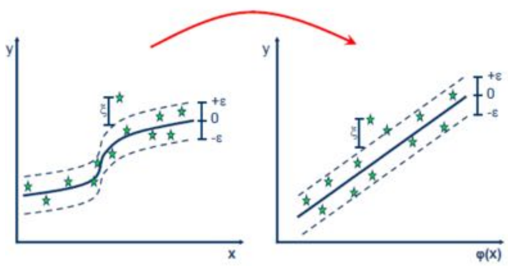
Problem 2 - Description

- Predict the quality of white wines on a scale given chemical measures of each wine
  - Fixed acidity
  - Volatile acidity
  - Citric acid
  - Residual sugar
  - Chlorides
  - Free sulfur dioxide
  - Total sulfur dioxide
  - Density
  - pH
  - Sulphates
  - Alcohol
  - Quality (score between 0 and 10)

Support Vector Regression



With kernel



- Minimize:

$$\frac{1}{2}\|w\|^2 + C\sum_{i=1}^{N}\left(\xi_i + \xi_i^*\right)$$

- Constraints:

$$y_i - wx_i - b \le \varepsilon + \xi_i$$
$$wx_i + b - y_i \le \varepsilon + \xi_i^*$$
$$\xi_i, \xi_i^* \ge 0$$

Source: http://www.saedsayad.com/support_vector_machine_reg.htm

Other regression datasets

- https://archive.ics.uci.edu/ml/datasets.php?task=reg
- More on cross validation:
  - https://scikit-learn.org/stable/modules/cross_validation.html

Classification

Supervised learning algorithms: SVM, NaiveBayes, LogisticRegression

Sample application: Detect the wine class

Problem description

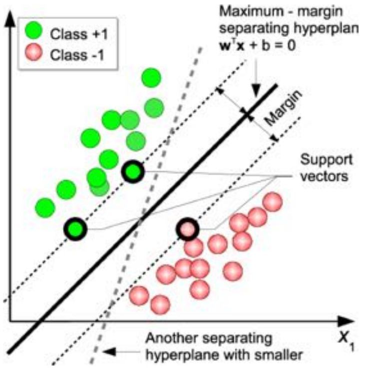
- Predict labels of wine based on its features

https://archive.ics.uci.edu/ml/datasets/wine

Data Set Characteristics:	Multivariate	**Number of Instances:**	178	**Area:**	Physical
Attribute Characteristics:	Integer, Real	**Number of Attributes:**	13	**Date Donated**	1991-07-01
Associated Tasks:	Classification	**Missing Values?**	No	**Number of Web Hits:**	1243736

1) Alcohol
2) Malic acid
3) Ash
4) Alcalinity of ash
5) Magnesium
6) Total phenols
7) Flavanoids
8) Nonflavanoid phenols
9) Proanthocyanins
10)Color intensity
11)Hue
12)OD280/OD315 of diluted wines
13)Proline

Classification with SVM



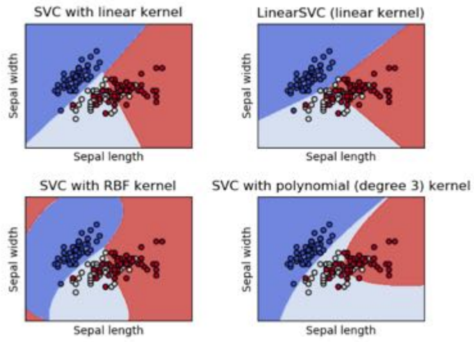
"One-against-one" for multi-class classification

If n_class is the number of classes, then n_class * (n_class - 1) / 2 classifiers are constructed and each one trains data from two classes.

Multi-class classification

- Strategies
  - "one-against-rest"
  - "one-against-one"
    - If n_class is the number of classes, then n_class * (n_class - 1) / 2 classifiers are constructed and each one trains data from two classes.

sklearn.svm.SVC()



Train SVM model

data

	alcohol	malic acid	ash	alcalinity of ash	magnesium	total phenols	flavanoids	nonflavanoid phenols	proanthocyanins	color intensity	hue	OD280/OD315	proline
0	13.49	3.59	2.19	19.5	88.0	1.62	0.48	0.58	0.88	5.70	0.81	1.82	580.0
1	12.51	1.73	1.98	20.5	85.0	2.20	1.92	0.32	1.48	2.94	1.04	3.57	672.0
2	12.33	0.99	1.95	14.8	136.0	1.90	1.85	0.35	2.76	3.40	1.06	2.31	750.0
3	13.28	1.64	2.84	15.5	110.0	2.60	2.68	0.34	1.36	4.60	1.09	2.78	880.0
4	12.29	2.83	2.22	18.0	88.0	2.45	2.25	0.25	1.99	2.15	1.15	3.30	290.0

```
#train model
model.fit(x_train, y_train)
```

```
#predict test set
y_pred = model.predict(x_test)

# calculate accuracy score
score = accuracy_score(y_test, y_pred)
print("accuracy: ", score)
```

```
accuracy:  0.42592592592592593
```

Why so low?

Importance of Feature Scaling

Data after scaling so that all features are centered around zero and have unit variance

	alcohol	malic acid	ash	alcalinity of ash	magnesium	total phenols	flavanoids	nonflavanoid phenols	proanthocyanins	color intensity	hue	OD280/OD315	proline
0	0.628447	1.081206	-0.652127	0.000000	-0.841477	-1.003358	-1.517062	1.711448	-1.230771	0.333174	-0.641378	-1.070901	-0.518219
1	-0.540882	-0.612994	-1.427534	0.288180	-1.037487	-0.112585	-0.086751	-0.350476	-0.195036	-0.933495	0.346530	1.330768	-0.215063
2	-0.755657	-1.287031	-1.538306	-1.354445	2.294697	-0.573329	-0.156280	-0.112562	2.014532	-0.722384	0.432435	-0.398434	0.041960
3	0.377877	-0.694972	1.747940	-1.152719	0.595936	0.501741	0.668135	-0.191866	-0.402183	-0.171658	0.561293	0.246586	0.470333
4	-0.803385	0.388952	-0.541355	-0.432270	-0.841477	0.271369	0.241029	-0.905609	0.685339	-1.296056	0.819008	0.960225	-1.473819

Train with scaled data

```
#build scaler
scaler = preprocessing.StandardScaler().fit(x_train)

x_train_scaled = scaler.transform(x_train)
x_test_scaled = scaler.transform(x_test)

model = svm.SVC()
model.fit(x_train_scaled, y_train)

y_pred = model.predict(x_test_scaled)

score = accuracy_score(y_test, y_pred)
print("Accuracy on scaled data: ", score)
```

```
Accuracy on scaled data:  0.9814814814814815
```

Naive Bayes

$$P(y|X) = \frac{P(X|y)P(y)}{P(X)}$$

$$P(y|x_1, ..., x_n) = \frac{P(x_1|y)P(x_2|y)...P(x_n|y)P(y)}{P(x_1)P(x_2)...P(x_n)}$$

$$= \frac{P(y)\prod_{i=1}^{n} P(x_i|y)}{P(x_1)P(x_2)...P(x_n)}$$

=> $$P(y|x_1, ..., x_n) \propto P(y)\prod_{i=1}^{n} P(x_i|y)$$

$$y = argmax_y P(y)\prod_{i=1}^{n} P(x_i|y)$$

Naive Bayes classification

```
from sklearn.naive_bayes import GaussianNB

model = GaussianNB()

#fit model
model.fit(x_train_scaled, y_train)
#predict
y_pred = model.predict(x_test_scaled)
#evaluate
score = accuracy_score(y_test, y_pred)
print("Accuracy on scaled data: ", score)
```

```
Accuracy on scaled data:  1.0
```

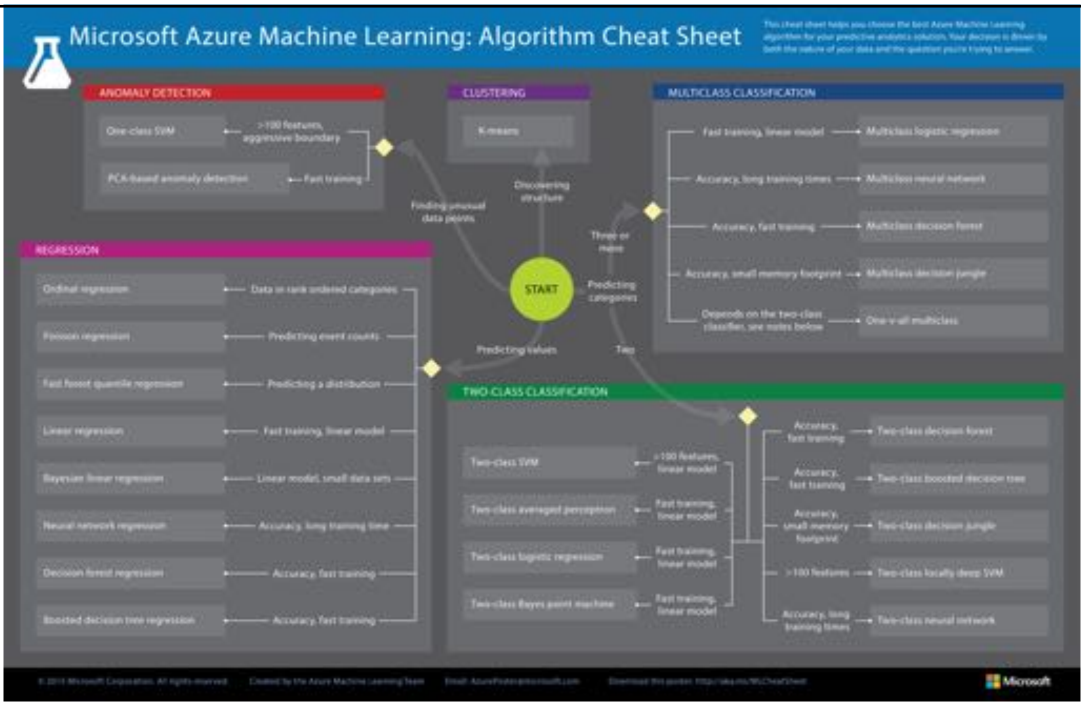
Logistic regression

```
from sklearn.linear_model import LogisticRegression

model = LogisticRegression()
model.fit(x_train, y_train)
y_pred = model.predict(x_test)
#evaluate
score = accuracy_score(y_test, y_pred)
print("Accuracy on scaled data: ", score)
```

```
Accuracy on scaled data:  1.0
```

Summary



Top 10 algorithms in data mining -2007

1. C4.5 (trees)
2. K-means
3. Support vector machines
4. Apriori
5. Expectation maximization
6. PageRank
7. AdaBoost
8. K-Nearest Neighbors
9. Naive Bayes
10. CART

Ref: Journal of Knowledge and Information Systems, 2007

Top 10 machine learning algorithms - 2018

1. Linear regression
2. Logistic regression
3. CART
4. Naive Bayes
5. KNN
6. Apriori
7. K-means
8. PCA
9. Bagging with random forests
10. Boosting with AdaBoost

Ref: https://www.dataquest.io/blog/top-10-machine-learning-algorithms-for-beginners/

References

- Segaran, Toby. *Programming collective intelligence: building smart web 2.0 applications*. " O'Reilly Media, Inc.", 2007.
- Pedregosa, Fabian, et al. "Scikit-learn: Machine learning in Python." *Journal of machine learning research* 12.Oct (2011): 2825-2830.