How to make super awesome web apps

Deepak Thukral
@iapain

Europython 2011

Computer Science

Europython 2011

I <3 data

and

Python

Why Python?

Because its awesome

Awesome web frameworks

Europython 2011

Awesome web framework in Python

- Django
- Cherrypy
- Turbogears
- Pylons
- Zope
- Your fav web framework

Awesome web app formula

Europython 2011

Awesome Idea

+

Awesome web framework(s)/tools

+

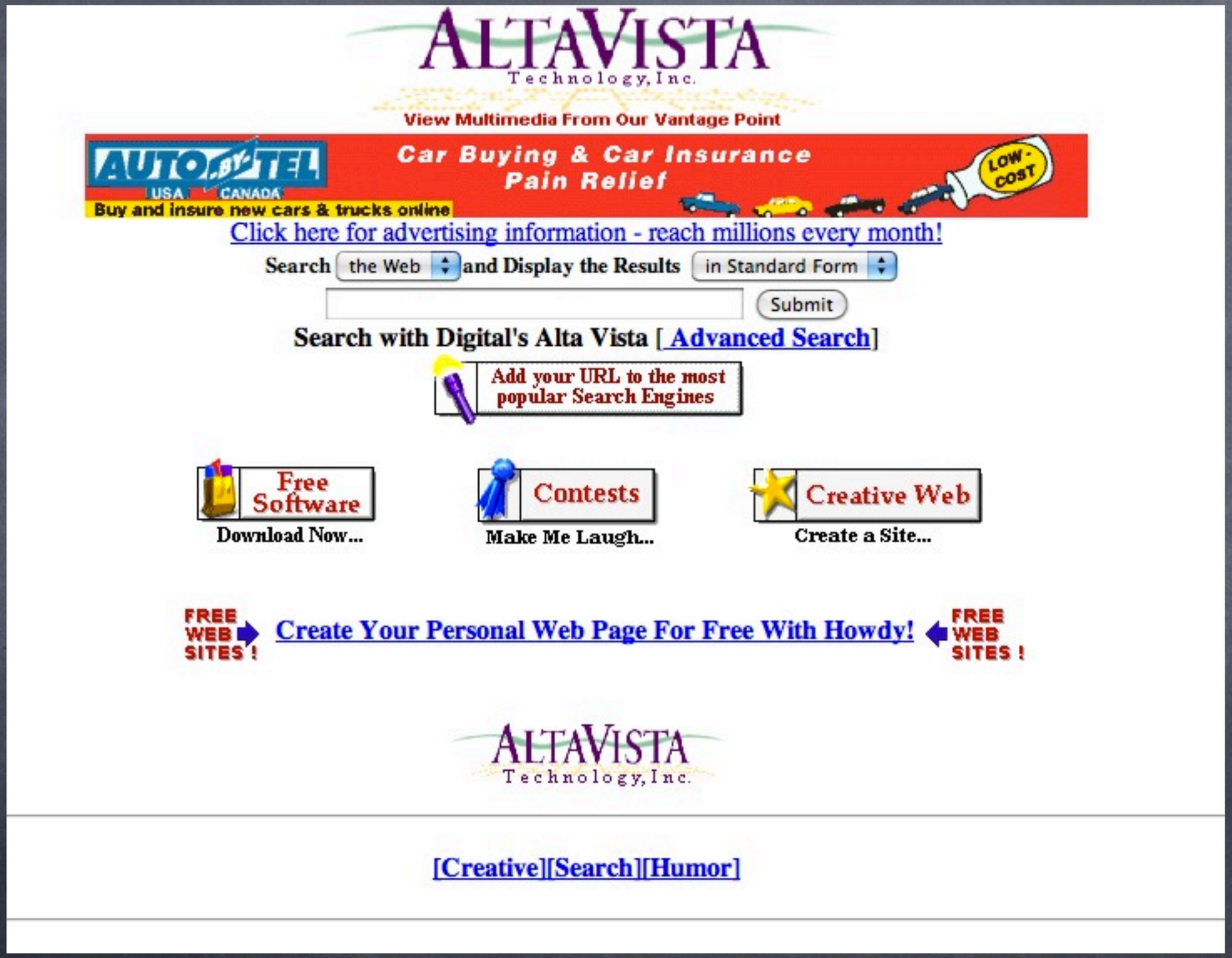
Awesome UI

=

Awesome Web app

But It was not so easy 15 years back

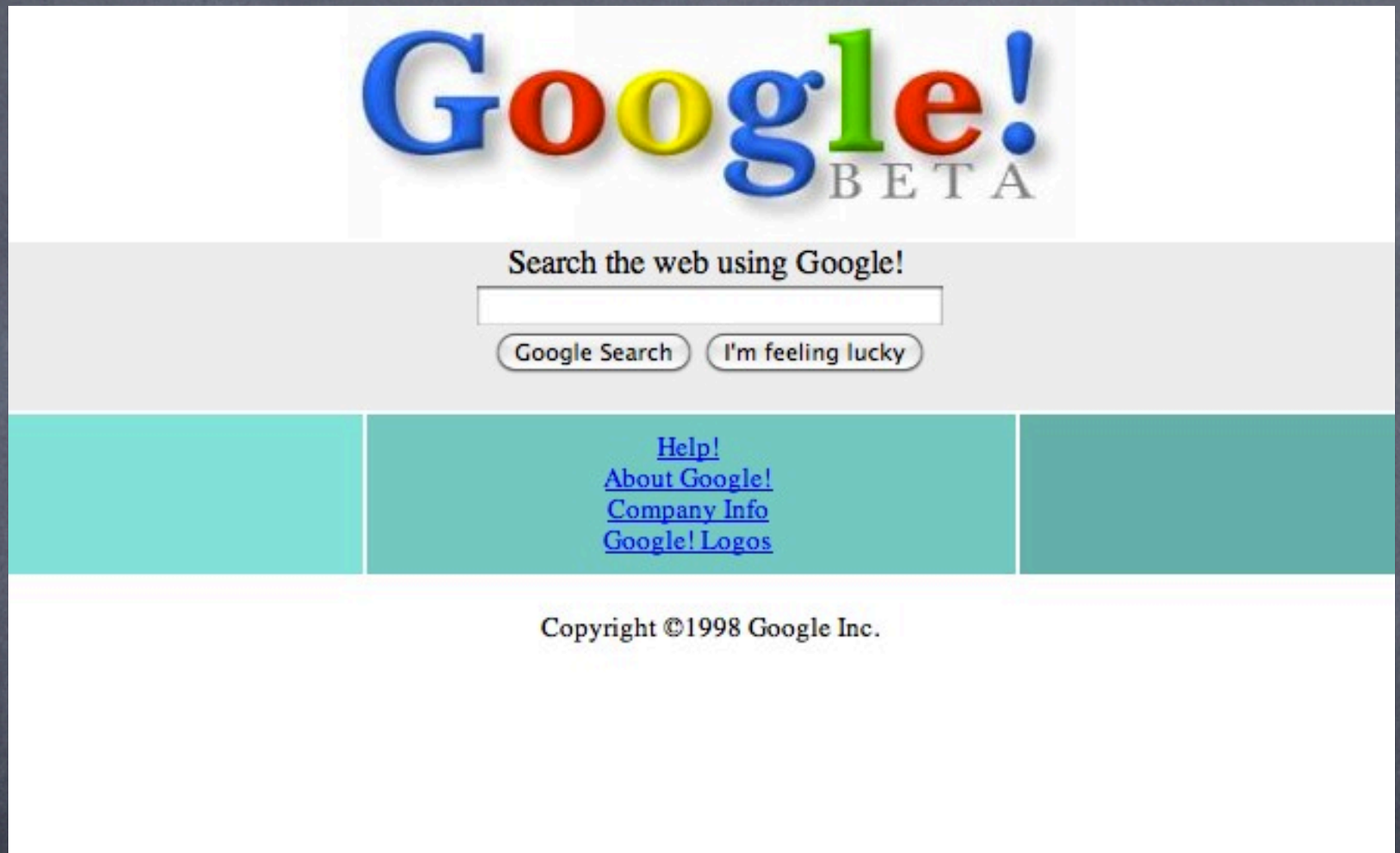
Awesome website in 1996



Source: Wayback machine

Two years later ... in 1998

It's super awesome and still it is



Why we need intelligence in web apps?

Information Overload

Massive Data: Wicked Hard

Source: Stefano Bussolon (flickr)

Information Overload rate

- ~ 130 M websites
- ~ 150 M tweets per day
- ~ 2 M pics on twitter per day
- ~ 4 M blogpost at WordPress per day

Why we need intelligence in web apps

- To find interesting information from data.
- To do something new in your web apps.
- To stay in competition in long run.
- Lots of opportunity.

What is Machine Learning (ML)?

What is Machine Learning (ML)?

Algorithms that allows machines to learn

Examples

- Spam detection (Classification)
- Google News (Clustering, Recommendation)
- Facebook friend recommendation (Recommendation/Collaborative filtering)
- Decide price for your products (Nearest Neighbor)
- and so on ...

Machine Learning

- Supervised
  - Classification
    - ANN, regression etc
- Unsupervised
  - Clustering
    - Self organized maps (SOM)

ML modules in Python

- Over 20 python modules (MDP, PyBrain, SciKit.learn, NLTK, Orange, LibSVM, Elefant etc)
- But ...

- None of them covers all.
- MDP is pure python.
- Scikit.learn looks more promising for real word problems.

Hardest part is to **extract features** that these modules can read it.

Clustering

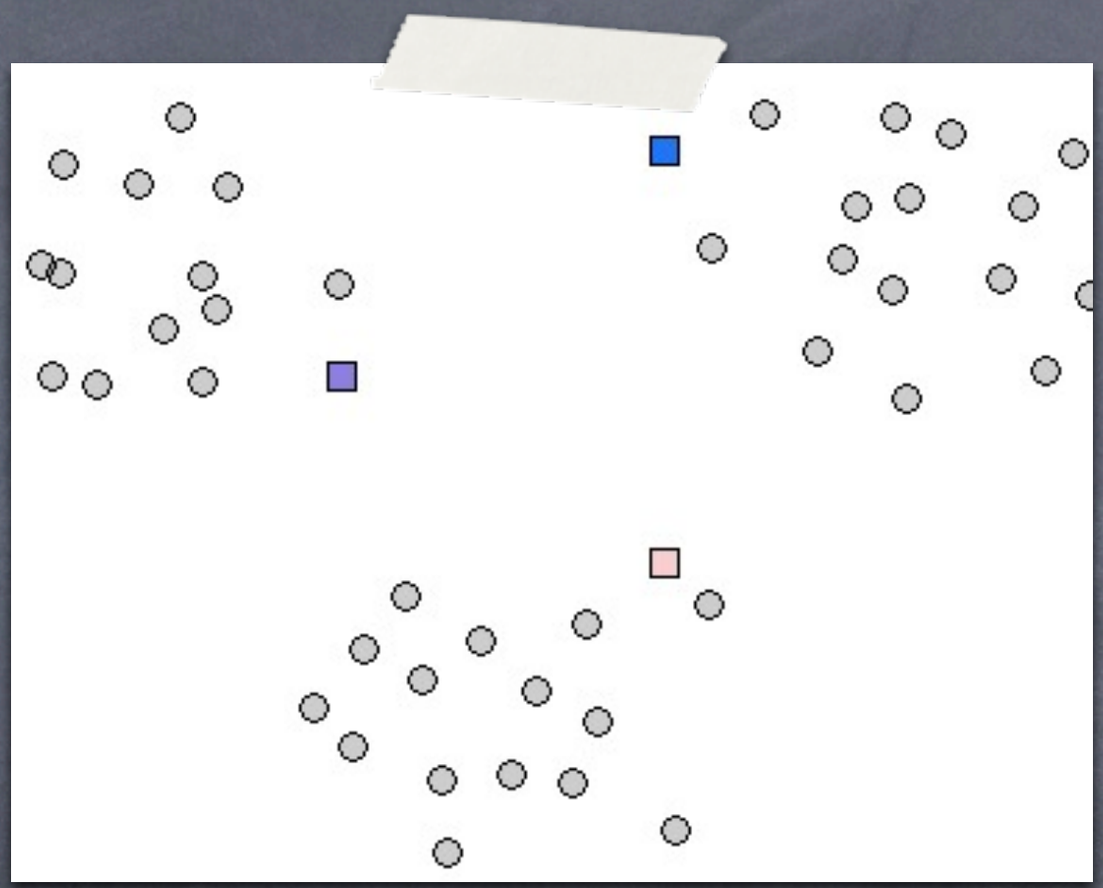
Source: NASA (NGC 869)

Clustering

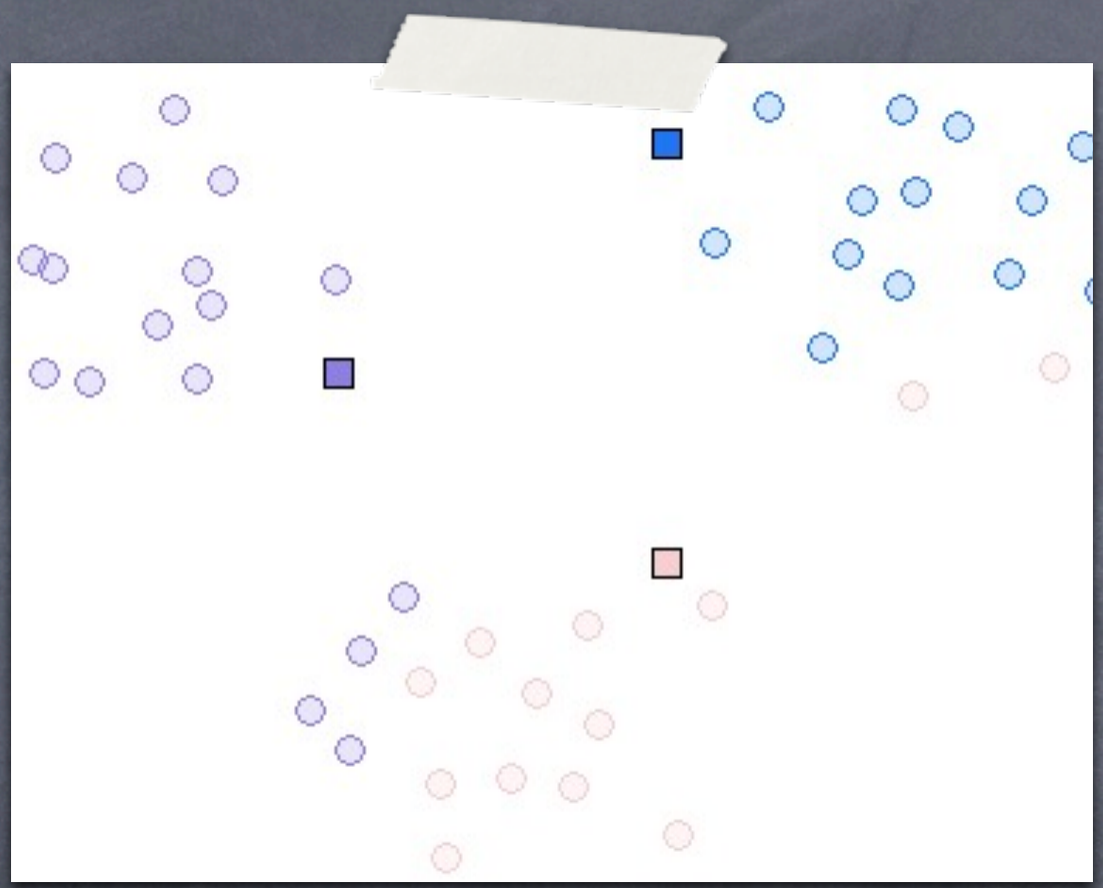
- Hierarchical
- k-means
- Quality Threshold
- Fuzzy c-means
- Locality sensitive hashing

Pick one: k-means

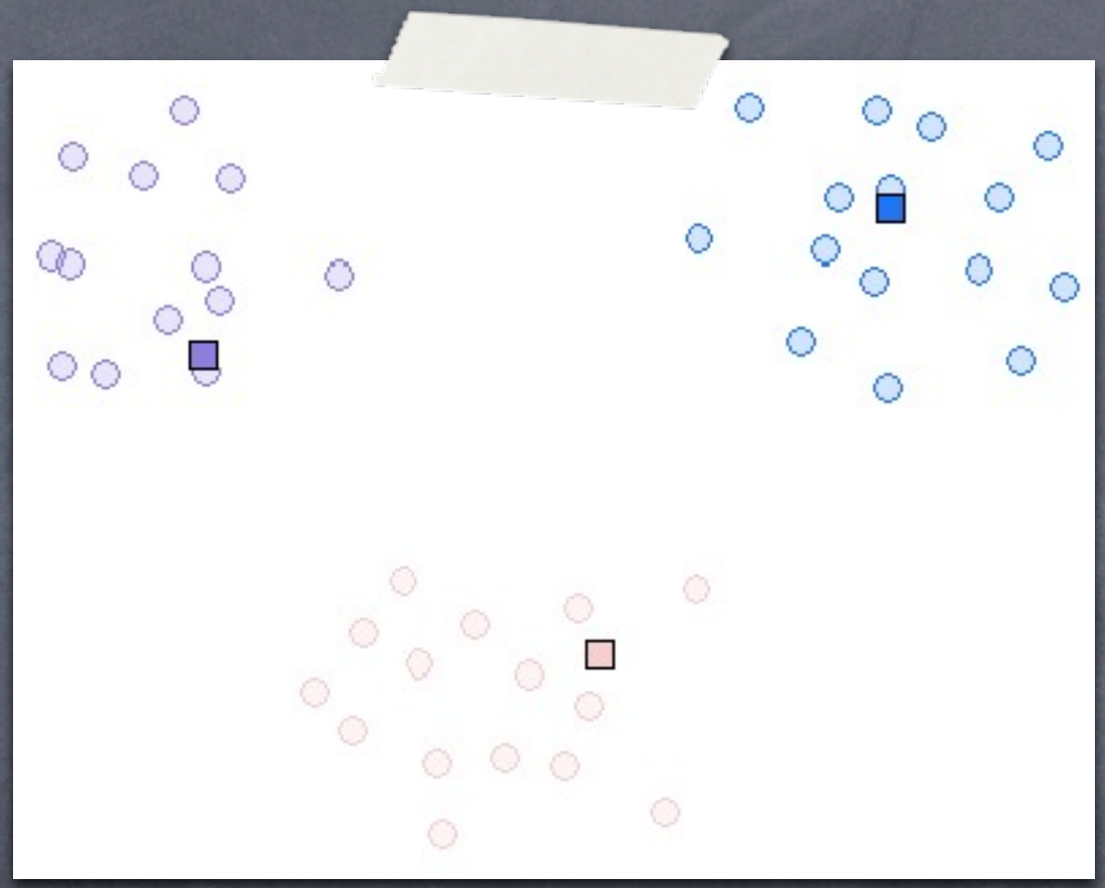
- We have N items and we need to cluster them into K groups.



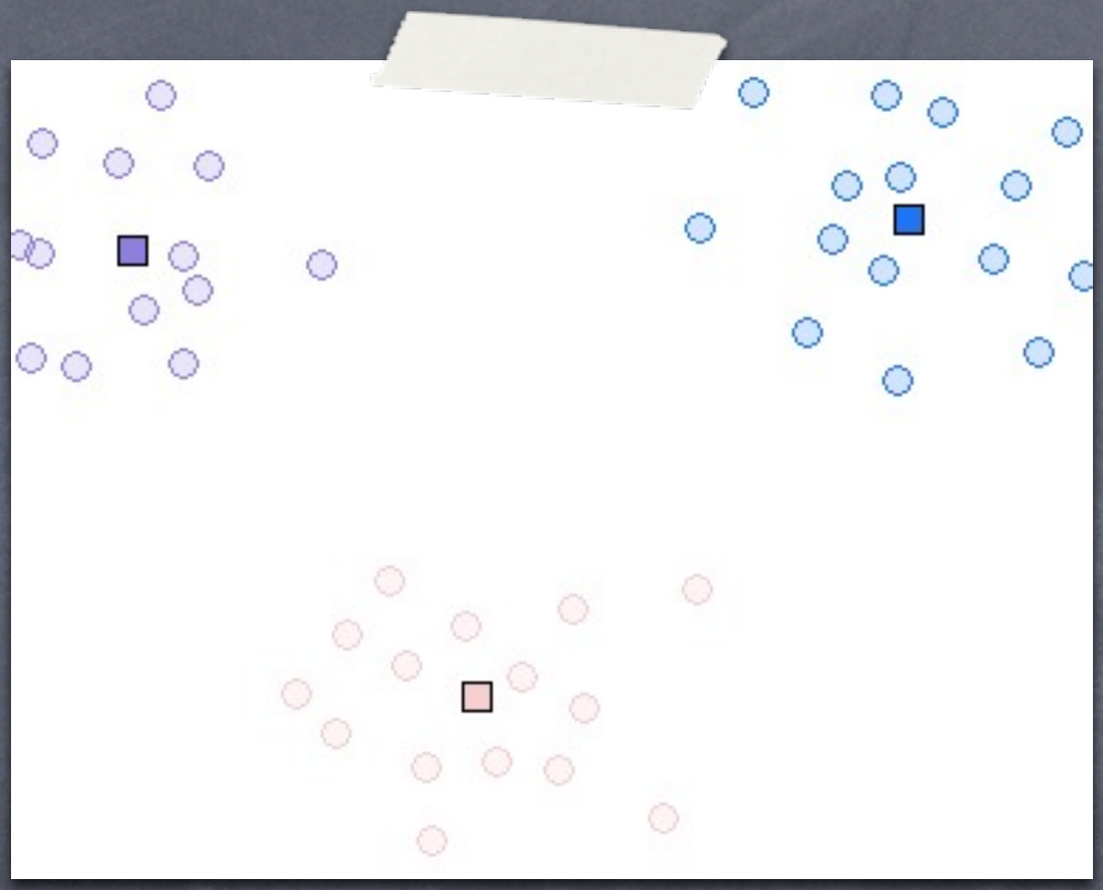
Randomly select k items as centroids



Compute distance between centroid and items and assign them to cluster



Adjust the centroids



Re-run until it converge

Example: News aggregator web app

- Example: Write a simple web app in Django which aggregate news articles from different sources.

Example: News aggregator web app

```
# models.py
class NewsSource(models.Model):
   name = models.CharField(max_length=50)
   url = models.URLField()


class News(models.Model)
   title = models.CharField(max_length=100)
   content = models.TextField()
   source = models.ForeignKey(“NewsSource”)
   cluster = models.ForeignKey(“Cluster”, null=True, blank=True)
   ...


class Cluster(models.Model)
   created = models.DateTimeField(auto_save_add=True)
   ...
```

And then an awesome UI - It's IMPORTANT

Let's make it super awesome by clustering news articles into related stories.

Step 1: Finding distance between stories

Story 1: Germany eases stance, boosting hope for Greek aid

Story 2: Private sector needed in Greek aid deal, Germany says

Looks similar, they should be in same cluster

Distance(A, B) = 1 - Similarity(A, B)

or

Distance(A, B) = 1 - n(A ∩ B)/n(A U B)

Also known as Jaccard Coefficient, you may also use minHash (it's quick when you have high dimensionality)

Story 1: Germany eases stance, boosting hope for Greek aid

Story 2: Private sector needed in Greek aid deal, Germany says

Distance(story1, story2) = 1 - 3/12 (ignore stop words)
= 0.75

Python code:

```
import re
from django.utils.stopwords import strip_stopwords

def jaccard_distance(item1, item2):
    """
    A simple distance function (curse of dimensionality applies)
    """
    #Tokenize string into bag of words

    feature1 = set(re.findall('\w+', strip_stopwords("%s %s" %
                    (item1.title.lower(), item1.content.lower())))[:100])
    feature2 = set(re.findall('\w+', strip_stopwords("%s %s" %
                    (item2.title.lower(), item2.content.lower())))[:100])

    similarity = 1.0*len(feature1.intersection(feature2))/len(feature1.union(feature2))

    return 1 - similarity
```

Brainstorming

- How many clusters are required?
- What kind of clustering is required?

Modify k-means

- Not every clusters have to contain items, we need quality (threshold)
- It allows us to have arbitrary k and we can discard empty clusters.
- It also allows us to maintain quality in clusters.
- Actually, it's more similar to Canopy Clustering.

Python code:

```
class Cluster(object):
    """
    Clustering class
    """
    def __init__(self, items, distance_function=jaccard_distance):
        self.distance = distance_function
        self.items = items

    def kmeans(self, k=10, threshold=0.80):
        "k is number of clusters, threshold is minimum acceptable distance"

        #pick k random stories and make then centroid
        centroids = random.sample(self.items, k)

        #remove centroid from collection
        items = list(set(self.items) - set(centroids))

        ...
```

Python code:

```
last_matches = None
        # Max. 50 iterations for convergence.
        for t in range(50):
            # Make k empty clusters
            best_matches = [[] for c in centroids]
            min_distance = 1 # it's max value of distance

            # Find which centroid is the closest for each row
            for item in items:
                best_center = 0
                min_distance = 1.0 #max
                minima = -1
                for centroid in centroids:
                    minima+=1
                    distance = self.distance(item, centroid)
                    if distance <= min_distance:
                        best_center = minima
                        min_distance = distance
                # maintain quality of your cluster
                if min_distance <= threshold:#threshold
                    best_matches[best_center].append(item)

            # If the results are the same as last time, this is complete
            if best_matches == last_matches:
                break
            last_matches = best_matches
        # Move best_matches to new centroids...
```

Sweet!

Syrian Troops Take Northwestern Town

Backed by tanks and helicopters, Syrian troops took over a northwestern town.

- Inside Syria: refugees in terror as tanks attack small town near Turkish border - Telegraph
- Syrian tanks roll into town near Turkish border - CBC
- Syrian Gunmen 'Burn Houses In Border Town' - Sky
- Syria forces storm border town near Turkey (Reuters) - Yahoo
- Syria troops 'raid border town' - BBC
- VIDEO: Army storms key border town in Syria - BBC
- Syrian troops, tanks attack town near Turkey - CBS

Fairway leeway: Can Obama and Boehner Ease Tensions?

- Can 'Golf Diplomacy' Ease Obama-Boehner Tensions? - VOA
- Obama and Boehner win golf summit - BBC
- Obama, Boehner emerge victorious in golf outing - CBS

UN condemns gay discrimination

The United Nations issued its first condemnation of discrimination against gays, lesbians and transgender peop

- UN backs gay rights for first time ever - LA Times

More examples

- Handwriting recognition
- Image Segmentation
- Marker Clustering on Maps
- Common patterns in your sale.

Classification

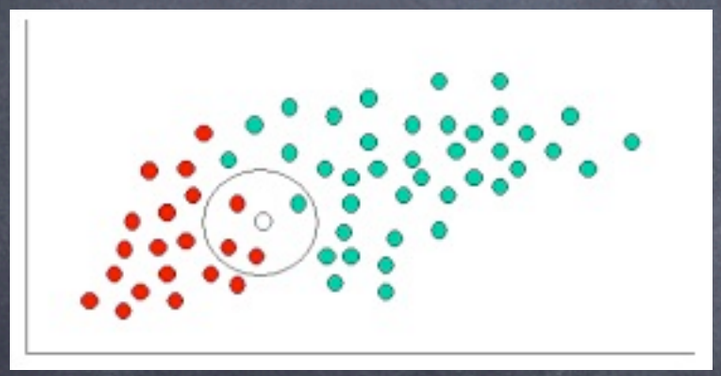
Source: Danny Nicholson (flickr)

Classification

- We present some observations and outcomes to classifier.
- Then we present an unseen observation and ask classifier to predict outcome.
- Essentially we want to classify a new record based on probabilities estimated from the training data.

Naïve Bayes Classifier

P(green) = 40/60 = 2/3
P(red) = 20/60 = 1/3



P(green in vicinity) = 1/40
P(red in vicinity) = 3/20

P(white|green) = 1/40*2/3 = 1/60
P(white|red) = 3/20*1/3 = 4/60

Example: Tweet Sentiment

- Example: Write a simple web app in Django which tells mood of the tweet (sentiment analysis).

Example: Tweet sentiment

```
# models.py
CHOICES = (('H', 'Happy'),
           ('S', 'Sad'),
           ('N', 'Neutral'),
           )

class Tweet(models.Model):
    author = models.CharField(max_length=50)
    tweet = models.TextField(max_length=140)
    sentiment = models.CharField(max_length=1, default='N', choices=CHOICES)
    url = models.URLField()
    ...
```

Example: Tweet sentiment

```
# classifier.py
from classifier import BayesClassifier

happy = [“happy”, “awesome”, “amazing”, “impressed”, “:)”]
sad = [“sad”, “sucks”, “problem”, “headache”, “:(“]

classifier = BayesClassifier()
classifier.train(happy, “happy”)
classifier.train(sad, “sad”)

classifier.guess(“Europython is awesome this year”) #happy
classifier.guess(“It’s sad I can’t make it to europython this year”) #sad
...
```

Happy

Pack done, tomorrow fly Frankfurt Hahn - Pisa . Ready for a great week to the @europython

Sad

Bag too small for running shoes. No running for me at #europython I guess

Enhancements: Tweet sentiment

- Bag of words
- Human computing

Recommendation (Collaborative Filtering)

User based or Item based?

User based recommendation

- Similar User: Relating user with other users. (Pearson correlation)
- Easy: Quite easy to implement.
- Slow: For large datasets It's slow.
- Make sense: For social sites.

Item based recommendation

- Items for user: Recommend items to a user.
- Faster: It's faster when dataset is huge (Amazon, Netflix etc).
- Make sense: Recommending products.

Why item based CF is fast?

Users changes very often but items does not.

Examples

- Last.fm
- Facebook, Twitter friend recommendation
- Google Ads, Facebook Ads etc.

Scared?

- scikit-learn (Python)
- Google Prediction API (Restful)
- Apache Mahout (Java / JPype)
- Amazon Mechanical Turk (Human computing)

Limitations

- Accuracy: Oh la la. Not always accurate. (Locality)
- Computation: In large dataset it might require lots of computation.
- Wicked Hard Problems: In some cases it's just wickedly hard.
- And many more...

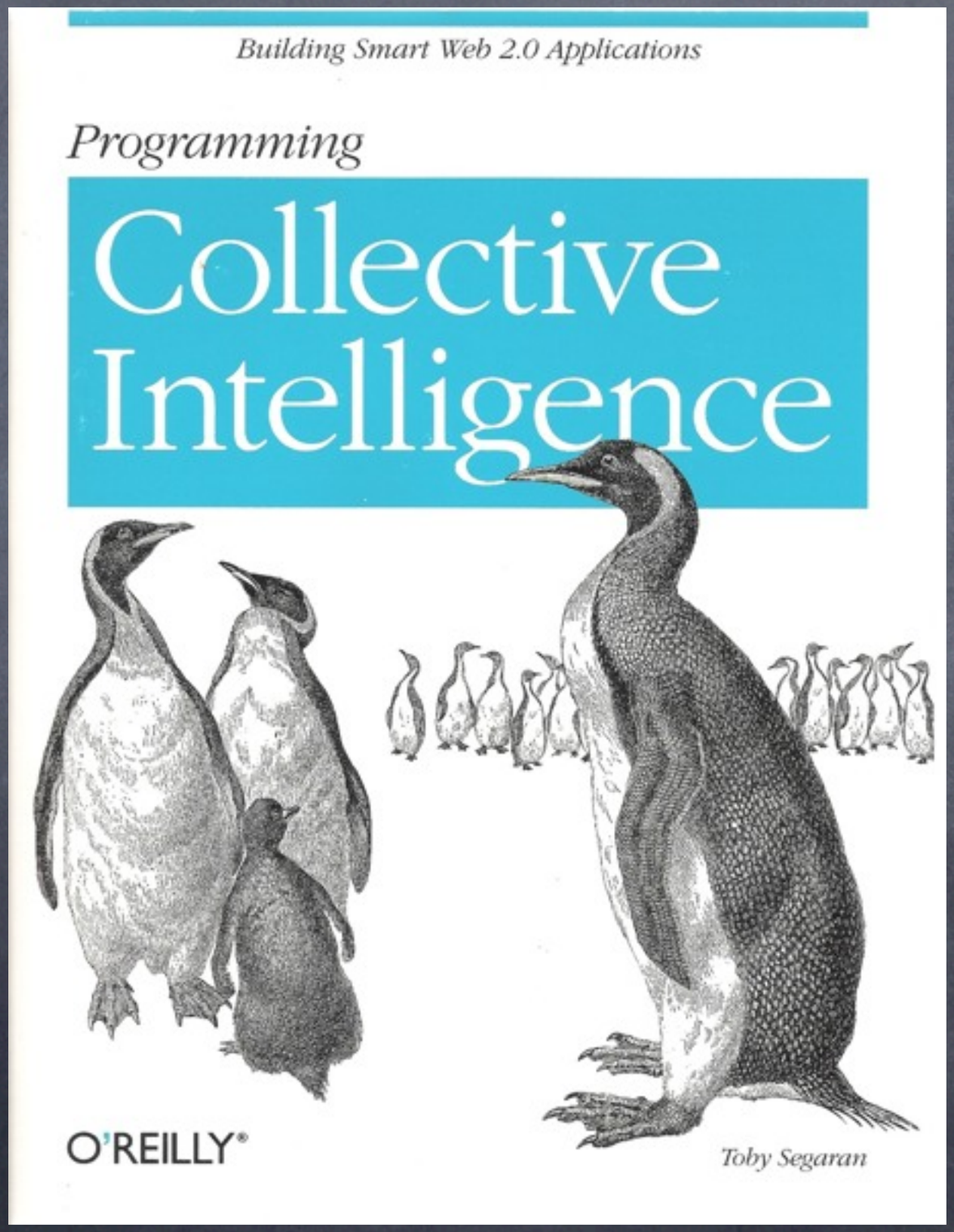
Awesome web app checklist

- An awesome idea
- An awesome architecture (scalable)
- An awesome web framework (probably Django)
- An awesome UI
- An awesome Cloud solution (if required)

Super awesome checklist

- An awesome idea
- An awesome architecture (scalable)
- An awesome web framework (probably Django)
- An awesome UI
- Do interesting thing with you data.
- An awesome Cloud solution (if required)

This books is awesome



Other useful techniques

- Support Vector Machines (Classification)
- Canopy Clustering
- Locality sensitive hashing
- k-dimensional trees

You're Awesome - Q A

https://github.com/iapain/smartwebapps