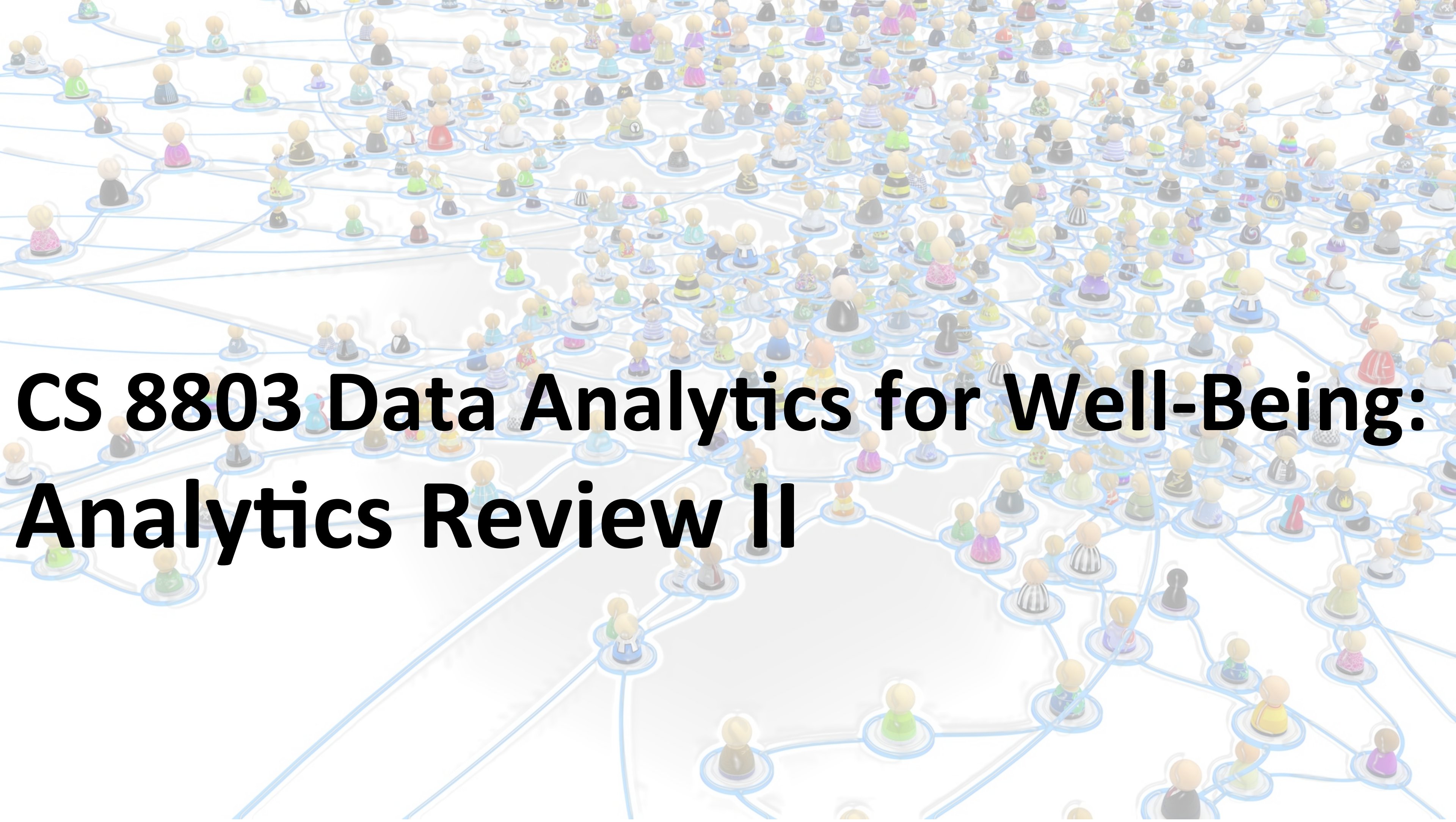


CS 8803 Data Analytics for Well-Being: Analytics Review II

Munmun De Choudhury

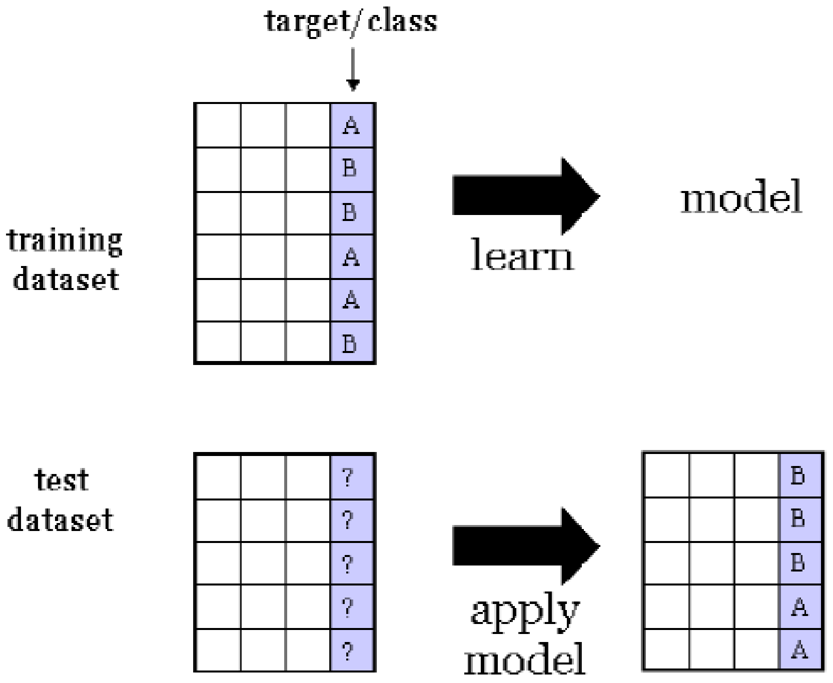
munmund@gatech.edu

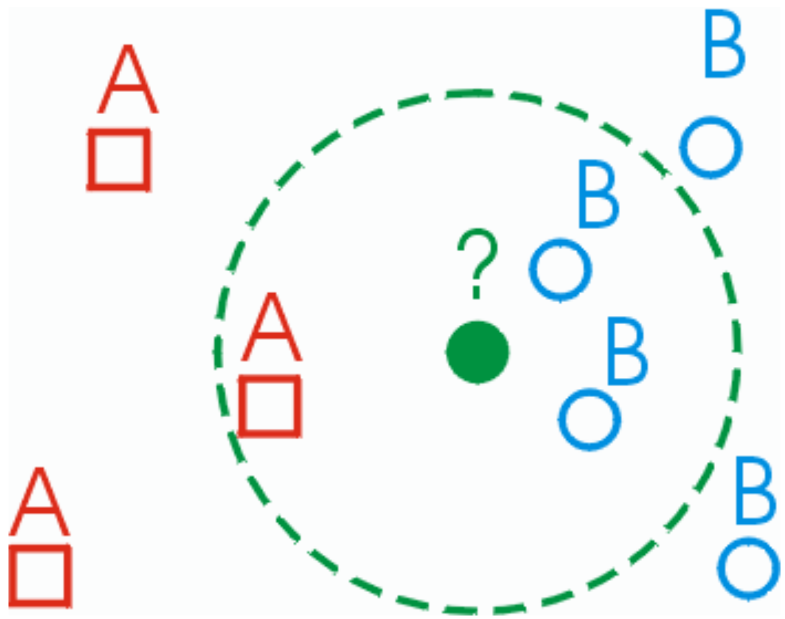
Week 7 | February 24, 2016

Classification

Simple example of a classification model ("classifier"):

- Use the label of the past training point which is most similar to the new test point, and return that as the prediction ("*k* nearest-neighbor").





Classification: *k*-NN Algorithm

- The *k*-NN algorithm for *continuous-valued target functions*
- Calculate the mean values of the *k* nearest neighbors
- Distance-weighted nearest neighbor algorithm
- Weight the contribution of each of the k neighbors according to their distance to the query point x_q
  - giving greater weight to closer neighbors

$$w \equiv \frac{1}{d(x_q, x_i)^2}$$

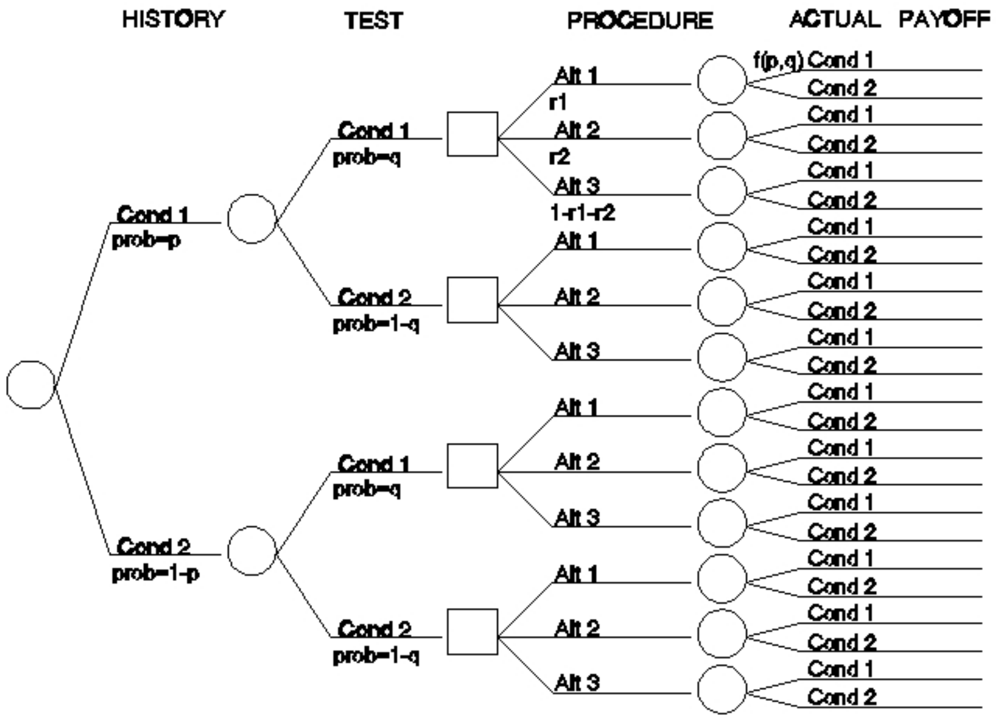
- Similarly, for real-valued target functions
- Robust to noisy data by averaging k-nearest neighbors
- Curse of dimensionality: distance between neighbors could be dominated by irrelevant attributes.
- To overcome it, axes stretch or elimination of the least relevant attributes.

Classification: Decision trees

- A decision tree is a decision support tool that uses a tree-like graph or model of decisions and their possible consequences, including chance event outcomes, resource costs, and utility.
- A decision tree is a flowchart-like structure in which internal node represents a "test" on an attribute (e.g. whether a coin flip comes up heads or tails), each branch represents the outcome of the test and each leaf node represents a class label (decision taken after computing all attributes).
- The paths from root to leaf represents classification rules.

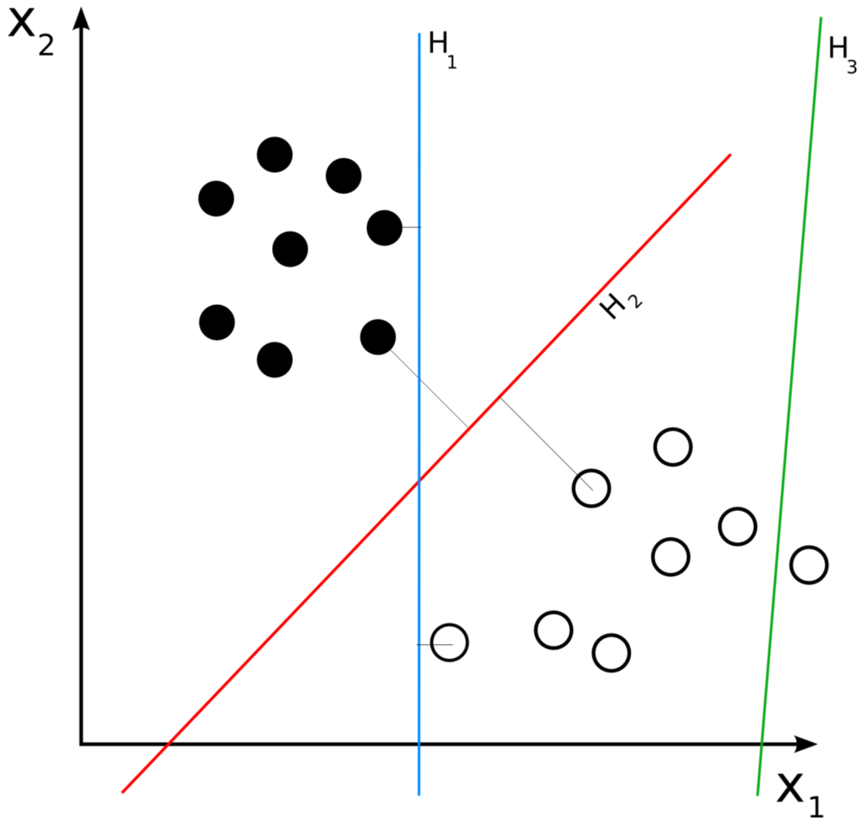
Classification: Decision trees

- A decision tree consists of 3 types of nodes:
  - Decision nodes - commonly represented by squares
  - Chance nodes - represented by circles
  - End nodes - represented by triangles
- Example: Should you recommend smoking cessation content to B?



Classification: linear classifier

- A linear classifier achieves this by making a classification decision based on the value of a linear combination of the characteristics.



Classification: linear classifier

- **Naive Bayes classifiers** are a family of simple probabilistic classifiers based on applying Bayes' theorem with strong (naive) independence assumptions between the features.
- A naive Bayes classifier assumes that the value of a particular feature is unrelated to the presence or absence of any other feature, given the class variable.

- Example: a fruit may be considered to be an apple if it is red, round, and about 3" in diameter. A naive Bayes classifier considers each of these features to contribute independently to the probability that this fruit is an apple, regardless of the presence or absence of the other features.

Bayesian Classification: Why?

- Probabilistic learning: Calculate explicit probabilities for hypothesis, among the most practical approaches to certain types of learning problems
- Incremental: Each training example can incrementally increase/ decrease the probability that a hypothesis is correct. Prior knowledge can be combined with observed data.
- Probabilistic prediction: Predict multiple hypotheses, weighted by their probabilities
- Standard: Even when Bayesian methods are computationally intractable, they can provide a standard of optimal decision making against which other methods can be measured

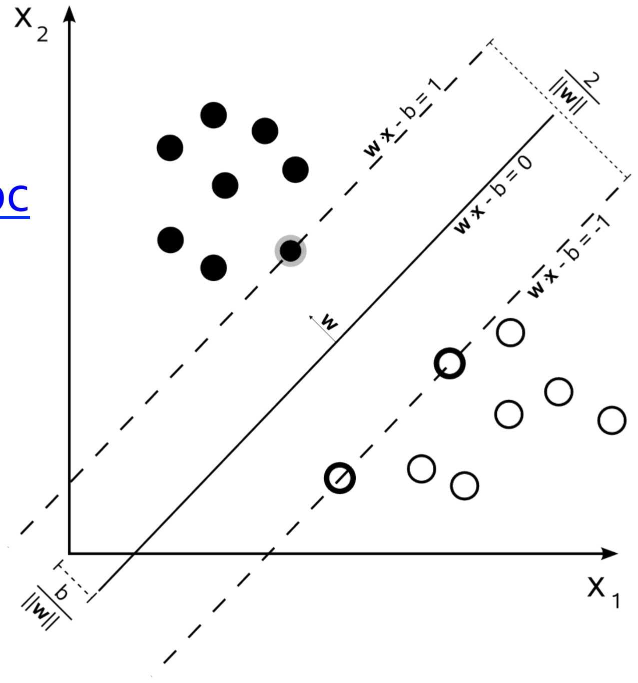
Classification: linear classifier

- A **logistic regression** model predicts a binary response from a binary predictor
- It is used for predicting the outcome of a categorical dependent variable (i.e., a class label) based on one or more predictor variables (features). That is, it is used in estimating the parameters of a qualitative response model.
- The probabilities describing the possible outcomes of a single trial are modeled, as a function of the explanatory (predictor) variables, using a logistic function.

$$F(x) = \frac{1}{1 + e^{-(\beta_0 + \beta_1 x)}}.$$

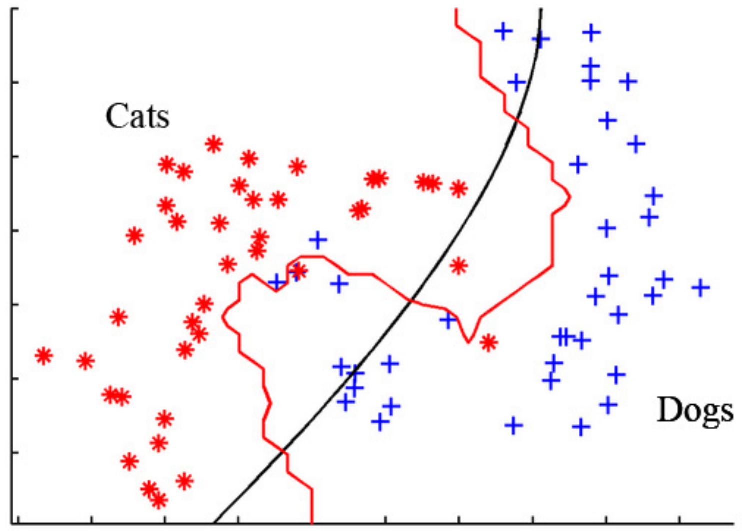
Classification: Support Vector Machine

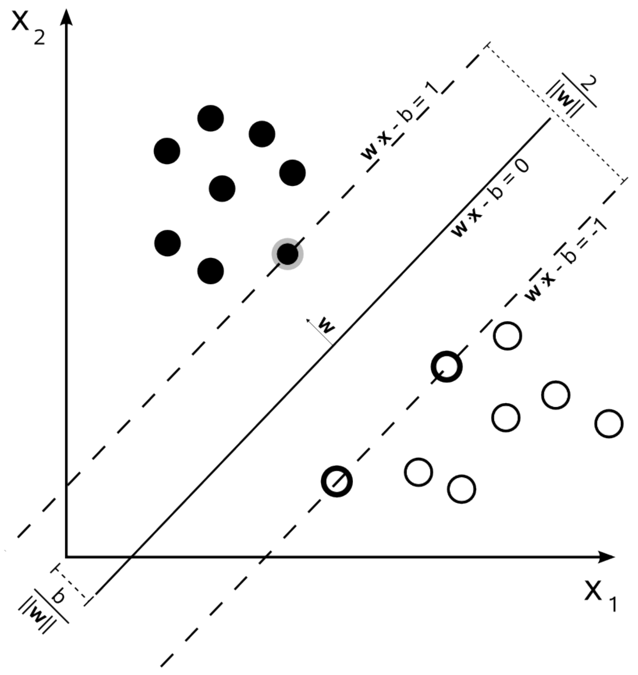
- A support vector machine constructs a hyperplane or set of hyperplanes in a high- or infinite-dimensional space, which can be used for classification, regression, or other tasks.
- Intuitively, a good separation is achieved by the hyperplane that has the largest distance to the nearest training data point of any class (so-called functional margin), since in general the larger the margin the lower the generalization error of the classifier.
- Video explaining how SVM works (in detail): https://www.youtube.com/watch?v=1NxnPkZM9bc


x_2
x_1
$\mathbf{w}\cdot\mathbf{x}-b=1$
$\mathbf{w}\cdot\mathbf{x}-b=0$
$\mathbf{w}\cdot\mathbf{x}-b=-1$
$\frac{2}{\|\mathbf{w}\|}$
$\frac{b}{\|\mathbf{w}\|}$
$\mathbf{w}$

Classification: Support Vector Machine

- But data is not always linearly separable!
- SVMs can efficiently perform a non-linear classification using what is called the *kernel trick*, implicitly mapping their inputs into high-dimensional feature spaces.
- How it works: https://www.youtube.com/watch?v=3liCbRZPrZA




x_2
x_1
$\mathbf{w}\cdot\mathbf{x} - b = 1$
$\mathbf{w}\cdot\mathbf{x} - b = 0$
$\mathbf{w}\cdot\mathbf{x} - b = -1$
$\frac{2}{\|\mathbf{w}\|}$
$\frac{b}{\|\mathbf{w}\|}$
$\mathbf{w}$

Discriminative Classifiers

Advantages

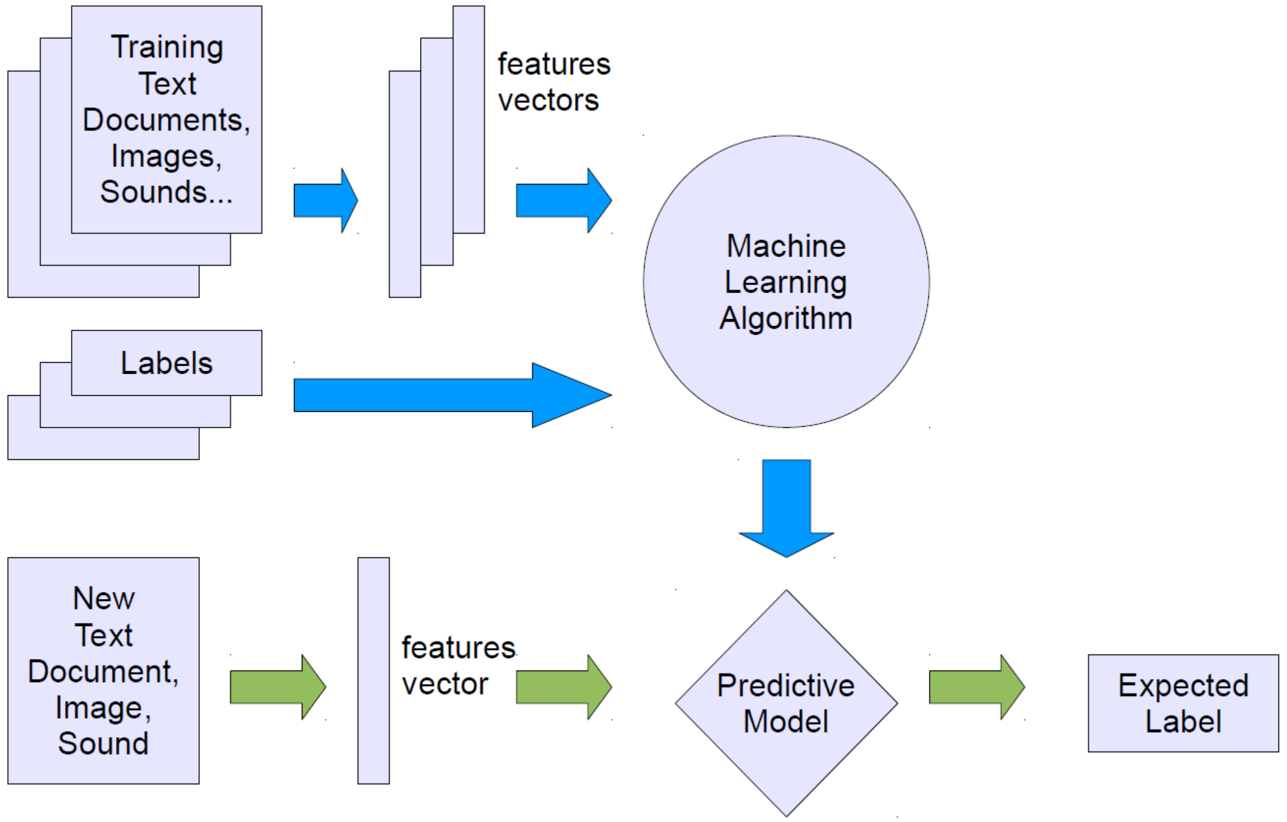
- prediction accuracy is generally high
  - (as compared to Bayesian methods – in general)
- robust, works when training examples contain errors
- fast evaluation of the learned target function
  - (Bayesian networks are normally slow)

Criticism

- long training time
- difficult to understand the learned function (weights)
  - (Bayesian networks can be used easily for pattern discovery)
- not easy to incorporate domain knowledge
  - (easy in the form of priors on the data or distributions)

Classification Summary

- Convert training data to a set of vectors of features:
  - Build a model based on the statistical properties of features in the training set, e.g. Naïve Bayesian Classifier, Logistic Regression, Support Vector Machines
  - For each new text document to classify: (1) Extract features; (2) Ask model to predict the most likely outcome


Training Text Documents, Images, Sounds...
features vectors
Labels
Machine Learning Algorithm
New Text Document, Image, Sound
features vector
Predictive Model
Expected Label

Class Exercise II

- You have created a supervised learner (e.g. a binary classifier) to distinguish between flu infected individuals and non-flu infected individuals on Twitter.
- What features would you use for the purpose?
- What might happen to training error and test error (decrease / increase / no change) if you:
  - Engineer better features?
  - Double the number of model parameters?
  - Double the amount of training data?
  - Double the amount of test data?

How to choose the right classifier?

- Predictive accuracy
- Speed and scalability
  - time to construct the model
  - time to use the model
- Robustness
  - handling noise and missing values
- Scalability
  - efficiency for high dimension, large-scale data
- Interpretability
  - understanding and insight provided by the model
- Goodness of rules
  - decision tree size
  - compactness of classification rules

Clustering

- "Show me the sub-groups in the data."
- Why show sub-groups in the data? Sometimes:
    - Computational reasons (e.g. use cluster centers instead of the dataset)
    - Statistical reasons (e.g. identify/remove outliers)
    - Mainly: Visualization/understanding reasons

- Cite examples where you'll apply clustering to study a social computing problem?

Clustering: *K* means

- The *K*-means method is as follows:
  - First initialize the means μ_k somehow, for example by choosing *K* different points randomly. Then:
  - Assign each point according to

    $C(i) = \arg\min_k ||x_i - \mu_k||$.
  - Recompute each μ_k according to the new assignments.
  - Stop when no assignments change.
  - However, it does not necessarily obtain the global optimum. In practice, this is done, say, 10 times and the result with the lowest sum-of-squares is used.

Clustering: how to choose *K*?

- A (heuristic) approach (out of many that have been proposed) uses the *gap statistic* – it chooses the *K* where the data look most clustered when compared to uniformly-distributed data.
  - For each value of *K*, compute the log of within-cluster scatter, $\log W_K$ for the best clustering using that *K*.
  - For each value of *K*, also compute this quantity for *m* clusterings using uniformly-distributed data – call this $\log W'_K$ and its standard deviation s_K. it the next cluster center.
  - Compute $G(K) = |\log W_K - \log W'_K|$.
  - Choose the *K* such that $G(K) \geq G(K+1) - s_K \sqrt{(1 + 1/m)}$, i.e. the smallest *K* producing a gap within one standard deviation of the gap at $K + 1$.

Clustering: hierarchical clustering

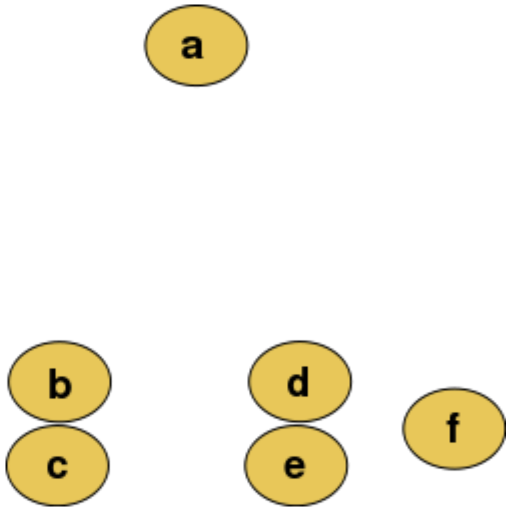
- Hierarchical clustering is a method of cluster analysis which seeks to build a hierarchy of clusters. Strategies for hierarchical clustering generally fall into two types:
  - Agglomerative: This is a "bottom up" approach: each observation starts in its own cluster, and pairs of clusters are merged as one moves up the hierarchy.
  - Divisive: This is a "top down" approach: all observations start in one cluster, and splits are performed recursively as one moves down the hierarchy.

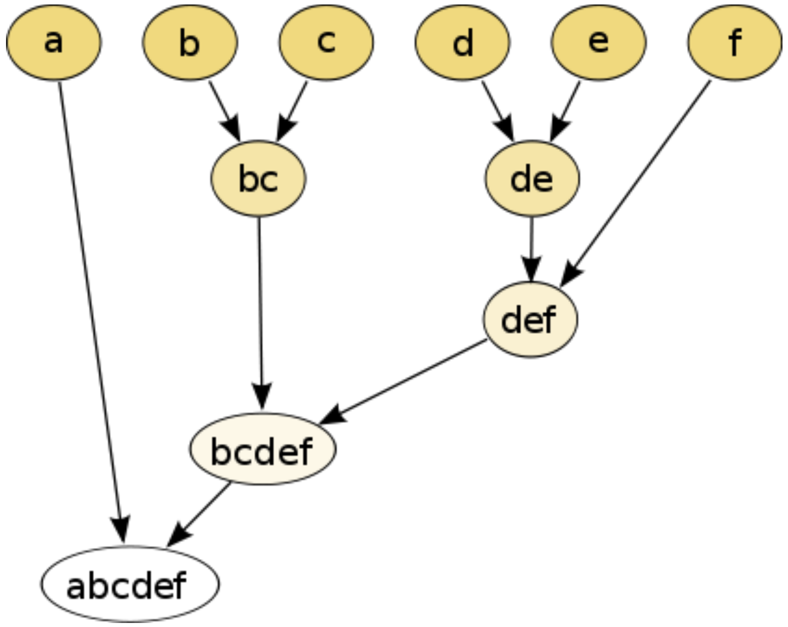
Clustering: hierarchical clustering

In order to decide which clusters should be combined (for agglomerative), or where a cluster should be split (for divisive), a measure of dissimilarity between sets of observations is required.

Euclidean distance $\|a - b\|_2 = \sqrt{\sum_i (a_i - b_i)^2}$

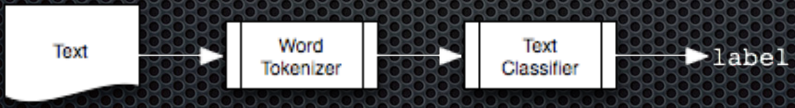
Cosine similarity $\frac{a \cdot b}{\|a\| \|b\|}$





Classification Algos in NLTK

- Naive Bayes
- Maximum Entropy / Logistic Regression
- Decision Tree
- SVM (coming soon)



Classification using NLTK

```
from nltk.classify import NaiveBayesClassifier

neg_examples = [(features(reviews.words(i)), 'neg') for i in neg_ids]
pos_examples = [(features(reviews.words(i)), 'pos') for i in pos_ids]
train_set = pos_examples + neg_examples

classifier = NaiveBayesClassifier.train(train_set)
```

Other NLTK Features

- clustering
- metrics
- parsing
- stemming
- WordNet
- ... and a lot more

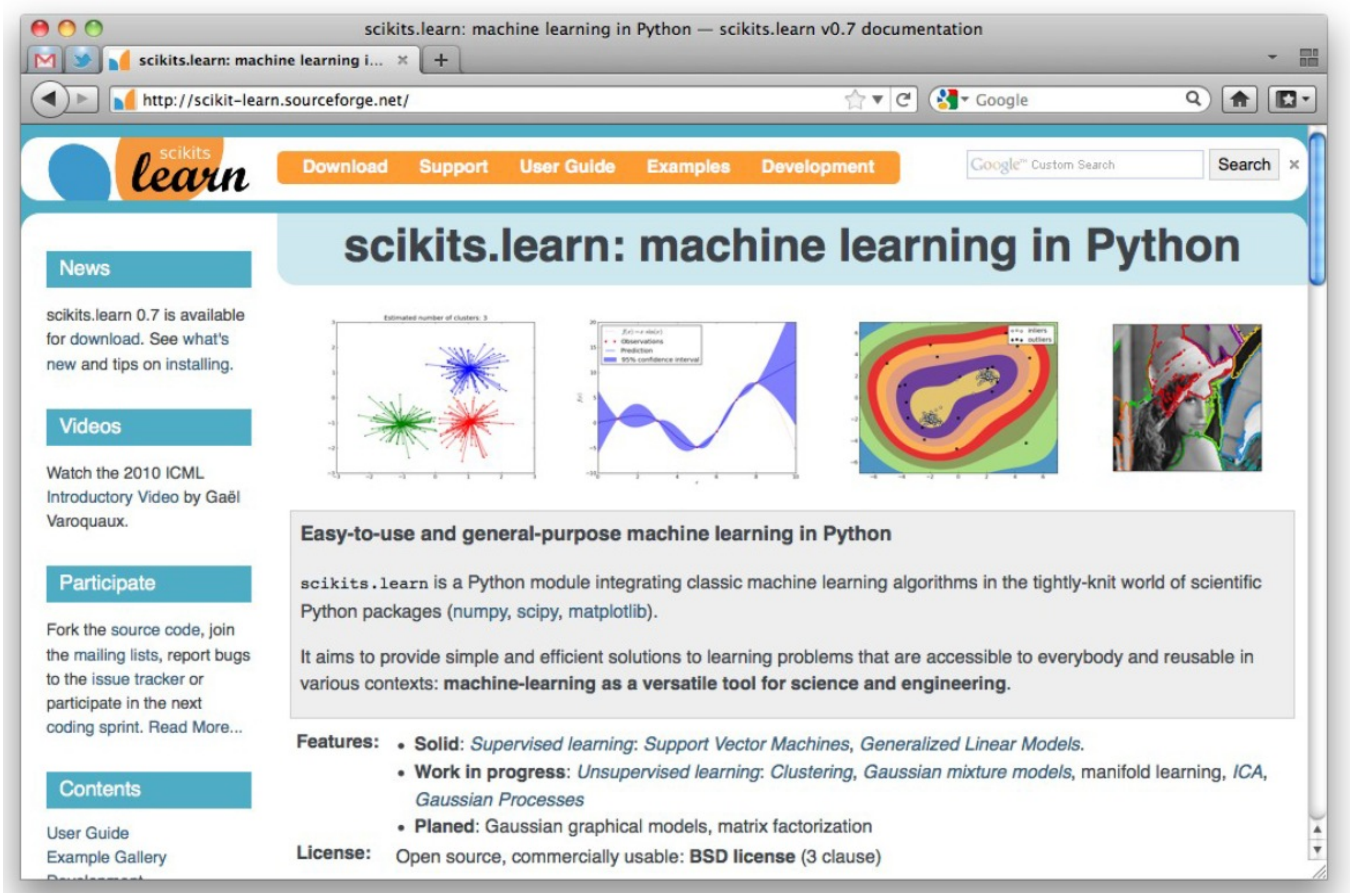
Notable Included Corpora

- movie_reviews: pos & neg categorized IMDb reviews
- treebank: tagged and parsed WSJ text
- treebank_chunk: tagged and chunked WSJ text
- brown: tagged & categorized english text
- 60 other corpora in many languages

Other Python NLP Libraries

- pattern: http://www.clips.ua.ac.be/pages/pattern
- scikits.learn: http://scikit-learn.sourceforge.net/stable/
- fuzzywuzzy: https://github.com/seatgeek/fuzzywuzzy

Doing slightly advanced data mining


scikits.learn: machine learning in Python — scikits.learn v0.7 documentation
scikits.learn: machine learning i...
http://scikit-learn.sourceforge.net/
Google
scikits learn
Download
Support
User Guide
Examples
Development
Google Custom Search
Search

scikits.learn: machine learning in Python

News

scikits.learn 0.7 is available for download. See what's new and tips on installing.

Videos

Watch the 2010 ICML Introductory Video by Gaël Varoquaux.

Participate

Fork the source code, join the mailing lists, report bugs to the issue tracker or participate in the next coding sprint. Read More...

Contents

User Guide
Example Gallery

Easy-to-use and general-purpose machine learning in Python

`scikits.learn` is a Python module integrating classic machine learning algorithms in the tightly-knit world of scientific Python packages (numpy, scipy, matplotlib).

It aims to provide simple and efficient solutions to learning problems that are accessible to everybody and reusable in various contexts: **machine-learning as a versatile tool for science and engineering.**

Features:
- **Solid**: *Supervised learning: Support Vector Machines, Generalized Linear Models.*
- **Work in progress**: *Unsupervised learning: Clustering, Gaussian mixture models*, manifold learning, *ICA*, *Gaussian Processes*
- **Planed**: Gaussian graphical models, matrix factorization

License: Open source, commercially usable: **BSD license** (3 clause)

Feature extraction using scikits.learn

```
from scikits.learn.features.text import WordNGramAnalyzer

text = (u"J'ai mang\xe9 du kangourou  ce midi,"

        u" c'\xe9tait pas tr\xeas bon.")


WordNGramAnalyzer(min_n=1, max_n=2).analyze(text)

[u'ai', u'mange', u'du', u'kangourou', u'ce', u'midi', u'etait',
u'pas', u'tres', u'bon', u'ai mange', u'mange du', u'du
kangourou', u'kangourou ce', u'ce midi', u'midi etait', u'etait
pas', u'pas tres', u'tres bon']
```

```
from scikits.learn.features.text import CharNGramAnalyzer


analyzer = CharNGramAnalyzer(min_n=3, max_n=6)

char_ngrams = analyzer.analyze(text)
```

Ref. Statistical Machine Learning for Text Classification by Olivier Grisel

TF-IDF features and SVM

```
from scikits.learn.features.text.sparse import Vectorizer
from scikits.learn.sparse.svm.sparse import LinearSVC


vec = Vectorizer(analyzer=analyzer)
features = vec.fit_transform(list_of_documents)
clf = LinearSVC(C=100).fit(features, labels)


clf2 = pickle.loads(pickle.dumps(clf))
predicted_labels = clf2.predict(features_of_new_docs)
```

Ref. Statistical Machine Learning for Text Classification by Olivier Grisel

```
import numpy as np
from sklearn.svm import SVR
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import StandardScaler

x,y = np.load('data.npz')
x_test = np.linspace(0, 200)

model = Pipeline([
   ('standardize', StandardScaler()),
   ('svr', SVR(kernel='rbf', verbose=0, C=5e6,
                  epsilon=20)) ])
model.fit(x[::, np.newaxis], y)
y_test = model.predict(x_test[::, np.newaxis])
```

regularization term

Clustering using scikits.learn

```
from sklearn import datasets
from sklearn.cluster import KMeans
from numpy.random import RandomState

rng = RandomState(42)
k_means = KMeans(3, random_state=rng)
boston = datasets.load_boston()
X = boston.data
k_means.fit(X)
```

Text Feature extraction in sklearn

- sklearn.feature_extraction.text
- CountVectorizer
  - Transform articles into token-count matrix
- TfidfVectorizer
  - Transform articles into token-TFIDF matrix
- Usage:
  - fit(): construct token dictionary given dataset
  - transform(): generate numerical matrix

Text Feature extraction

- Analyzer
  - Preprocessor: str -> str
    - Default: lowercase
    - Extra: strip_accents – handle unicode chars
  - Tokenizer: str -> [str]
    - Default: re.findall(ur"(?u)\b\w\w+\b“, string)
  - Analyzer: str -> [str]
    1. Call preprocessor and tokenizer
    2. Filter stopwords
    3. Generate n-gram tokens

Feature Selection

- Decrease the number of features:
  - Reduce the resource usage for faster learning
  - Remove the most common tokens and the most rare tokens (words with less information):
    - Parameter for Vectorizer:
      - max_df
      - min_df
      - max_features

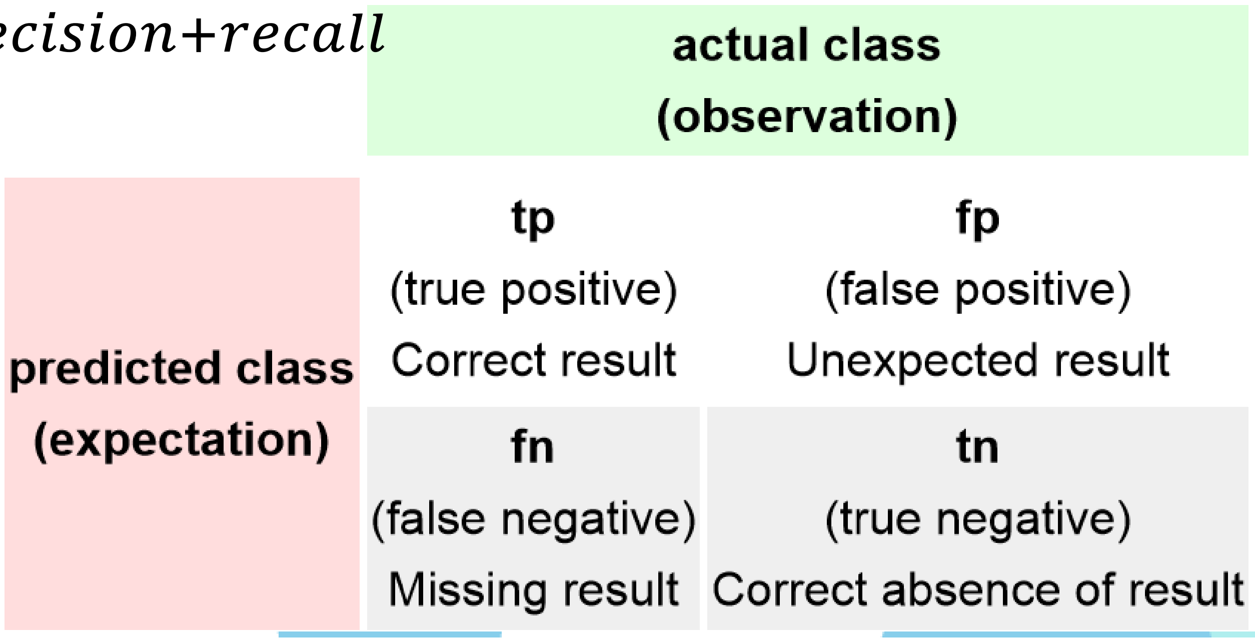
Cross Validation

- When tuning the parameters of model, let each article as training and testing data alternately to ensure the parameters are not dedicated to some specific articles.
  - from sklearn.cross_validation import KFold
  - for train_index, test_index in KFold(10, 2):
    - train_index = [5 6 7 8 9]
    - test_index = [0 1 2 3 4]

Performance Evaluation

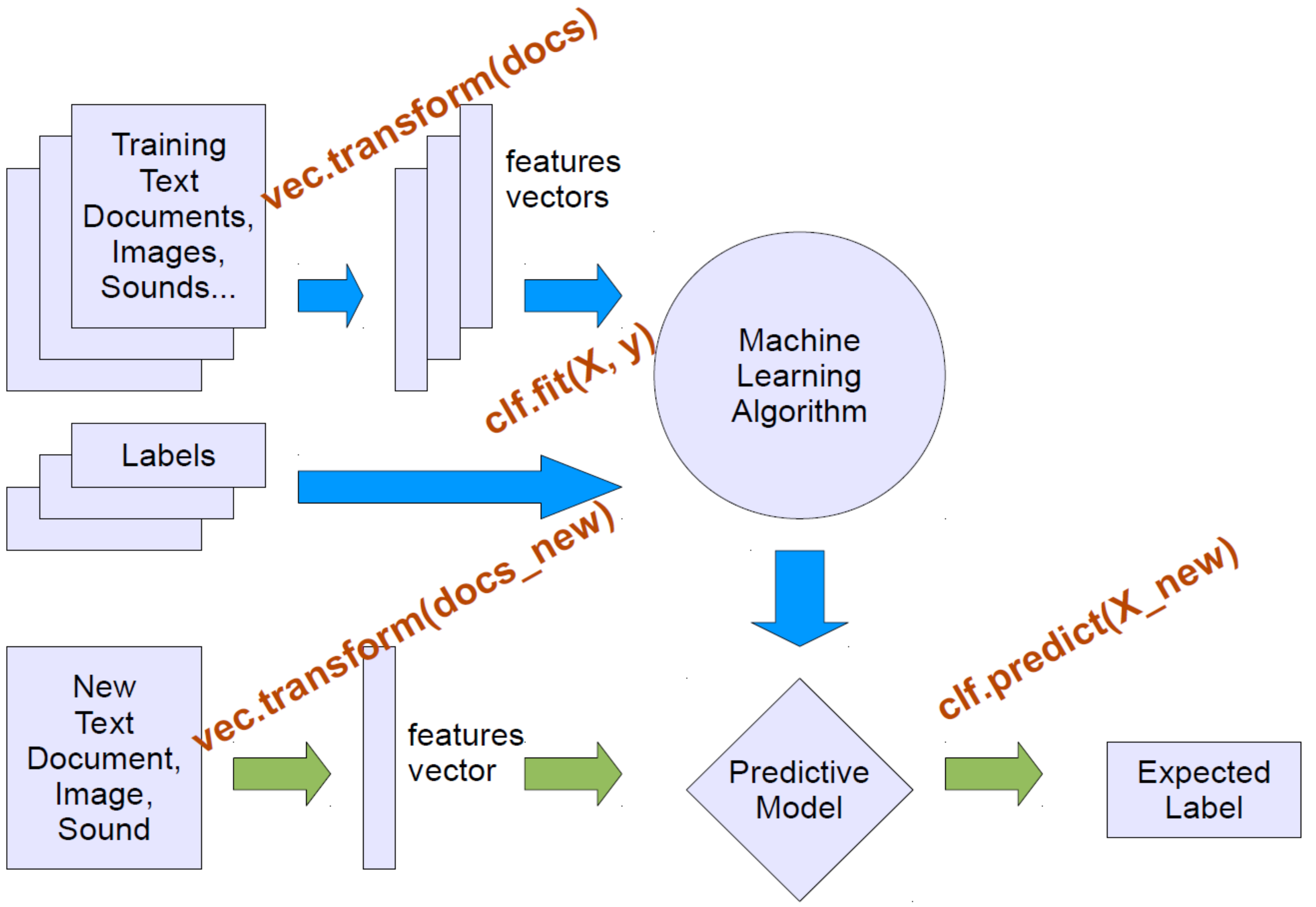
- $precision = \frac{tp}{tp+fp}$
- $recall = \frac{tp}{tp+fn}$
- $f1score = 2\frac{precision \times recall}{precision + recall}$

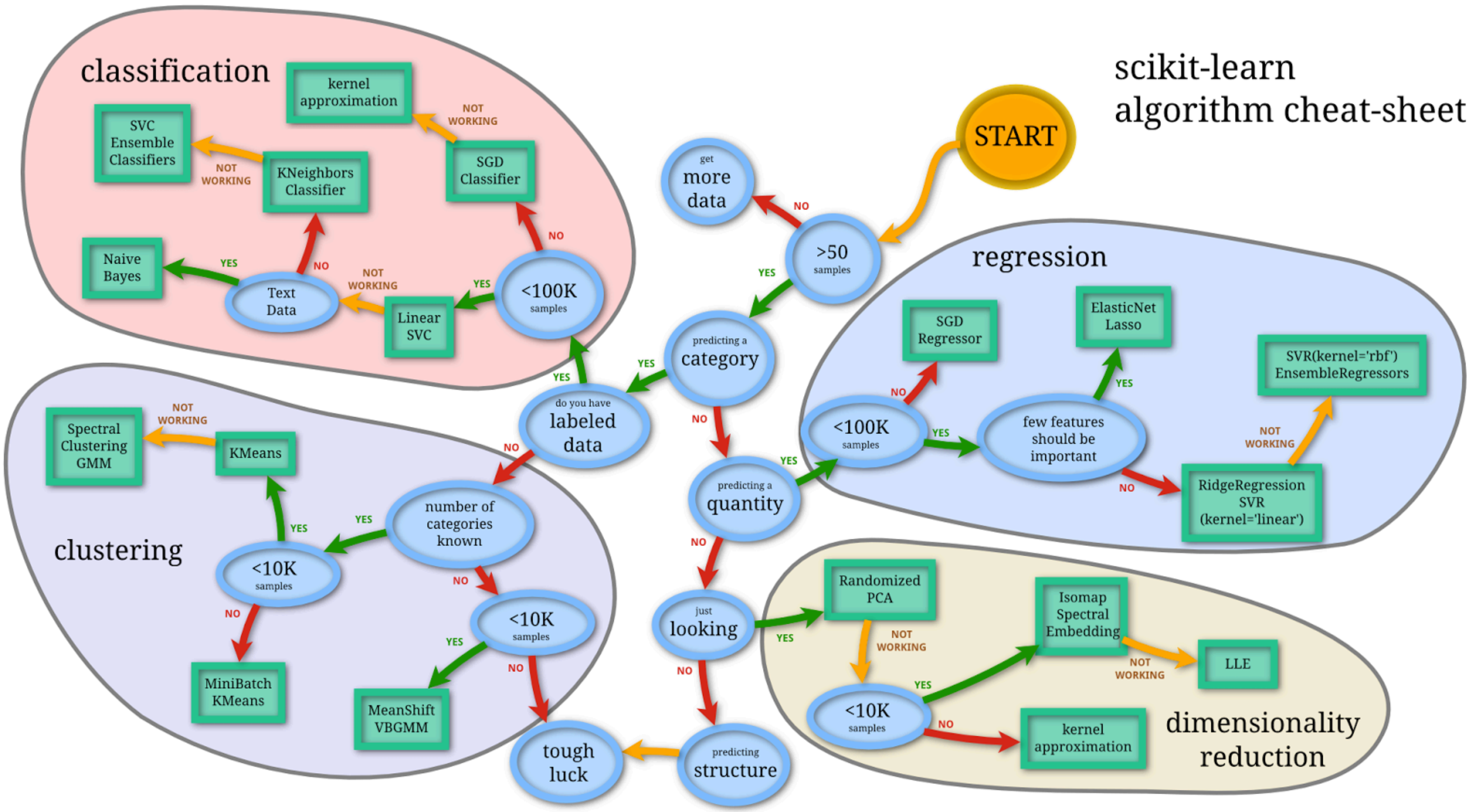
- sklearn.metrics
  - precision_score
  - recall_score
  - f1_score



	actual class (observation)	
predicted class (expectation)	**tp** (true positive) Correct result	**fp** (false positive) Unexpected result
	fn (false negative) Missing result	**tn** (true negative) Correct absence of result

Using scikits.learn Summary





scikit-learn
algorithm cheat-sheet
START
>50 samples
get more data
predicting a category
predicting a quantity
just looking
predicting structure
tough luck
do you have labeled data
classification
<100K samples
Linear SVC
Text Data
Naive Bayes
KNeighbors Classifier
SVC Ensemble Classifiers
SGD Classifier
kernel approximation
clustering
number of categories known
<10K samples
KMeans
Spectral Clustering GMM
MiniBatch KMeans
MeanShift VBGMM
regression
<100K samples
SGD Regressor
few features should be important
ElasticNet Lasso
RidgeRegression SVR (kernel='linear')
SVR(kernel='rbf') EnsembleRegressors
dimensionality reduction
Randomized PCA
<10K samples
Isomap Spectral Embedding
LLE
kernel approximation
YES
NO
NOT WORKING

A few notes

-The quality of your input data will affect the accuracy of your classifier.

- The threshold value that determines the sample size of the feature set will need to be refined until it reaches its maximum accuracy. This will need to be adjusted if training data is added, changed or removed.

Sentiment Classification w/ Python

- Sentiment Classifier using Word Sense Disambiguation using wordnet and word occurrence statistics from movie review corpus nltk.
- Classifies into positive and negative categories.

```
pip install sentiment_classifier
python setup.py install

cd sentiment_classifier/src/senti_classifier/
python senti_classifier.py -c reviews.txt
```

```
from senti_classifier import senti_classifier
sentences = ['The movie was the worst movie', 'It was the worst acting by the actors']
pos_score, neg_score = senti_classifier.polarity_scores(sentences)
print pos_score, neg_score
```

Some pointers

- http://scikit-learn.sf.net doc & examples
  http://github.com/scikit-learn code
- http://www.nltk.org code & doc & PDF book
- http://streamhacker.com/
  - Jacob Perkins' blog on NLTK & APIs
- https://github.com/japerk/nltk-trainer