



CS 8803 Social Computing: Data Mining Review

Munmun De Choudhury

munmund@gatech.edu

Week 5 | September 17, 2014

Regression

What if we need prediction?

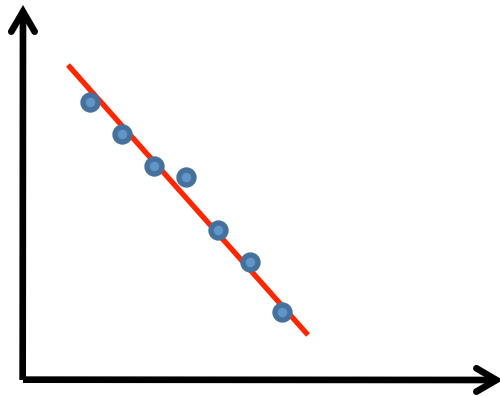
- *Answer: Regression*
- In a way, prediction is similar to classification
 - First, construct a model
 - Second, use model to predict unknown value
- Major method for prediction is regression
 - Linear and multiple regression
 - Non-linear regression
- But prediction is **different** from classification
 - Classification refers to predict categorical class label
 - Prediction models continuous-valued functions

Dependent and Independent Variables in Linear Regression

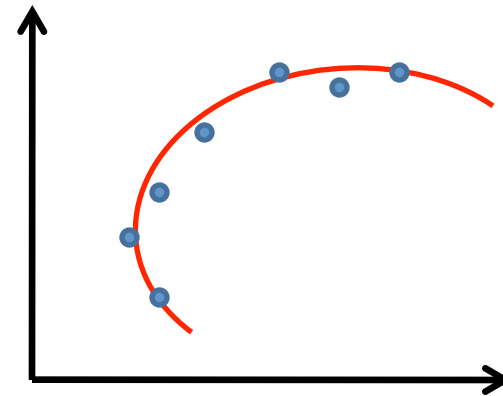
- **independent variable (denoted as X)** - the variable that is used to explain changes
- **dependent variable (denoted as Y)** - the variable that is to be explained.
- **Linear regression** involves the use of one variable to make a prediction about other variable. It also involves testing hypotheses about the relation between the two variables and quantifying the strength of relationship between the two variables.

Types of Regression Models

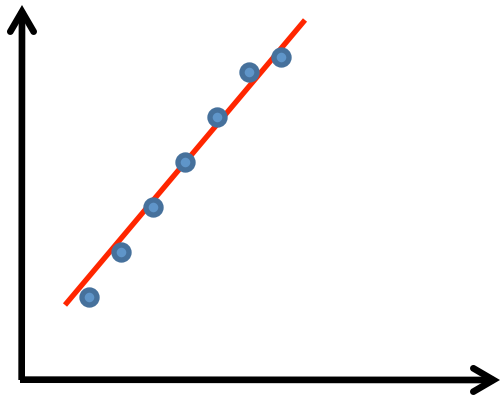
Negative Linear Relationship



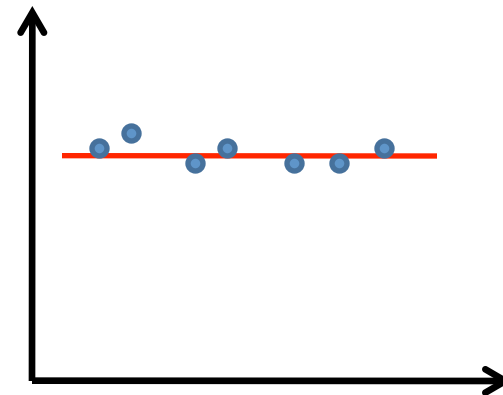
Relationship NOT Linear



Negative Linear Relationship



No Relationship



Regression Analysis and Log-Linear Models in Prediction

Linear regression: $Y = \alpha + \beta X$

- Two parameters, α and β specify the line and are to be estimated by using the data at hand.
- using the least squares criterion to the known values of $Y_1, Y_2, \dots, X_1, X_2, \dots$

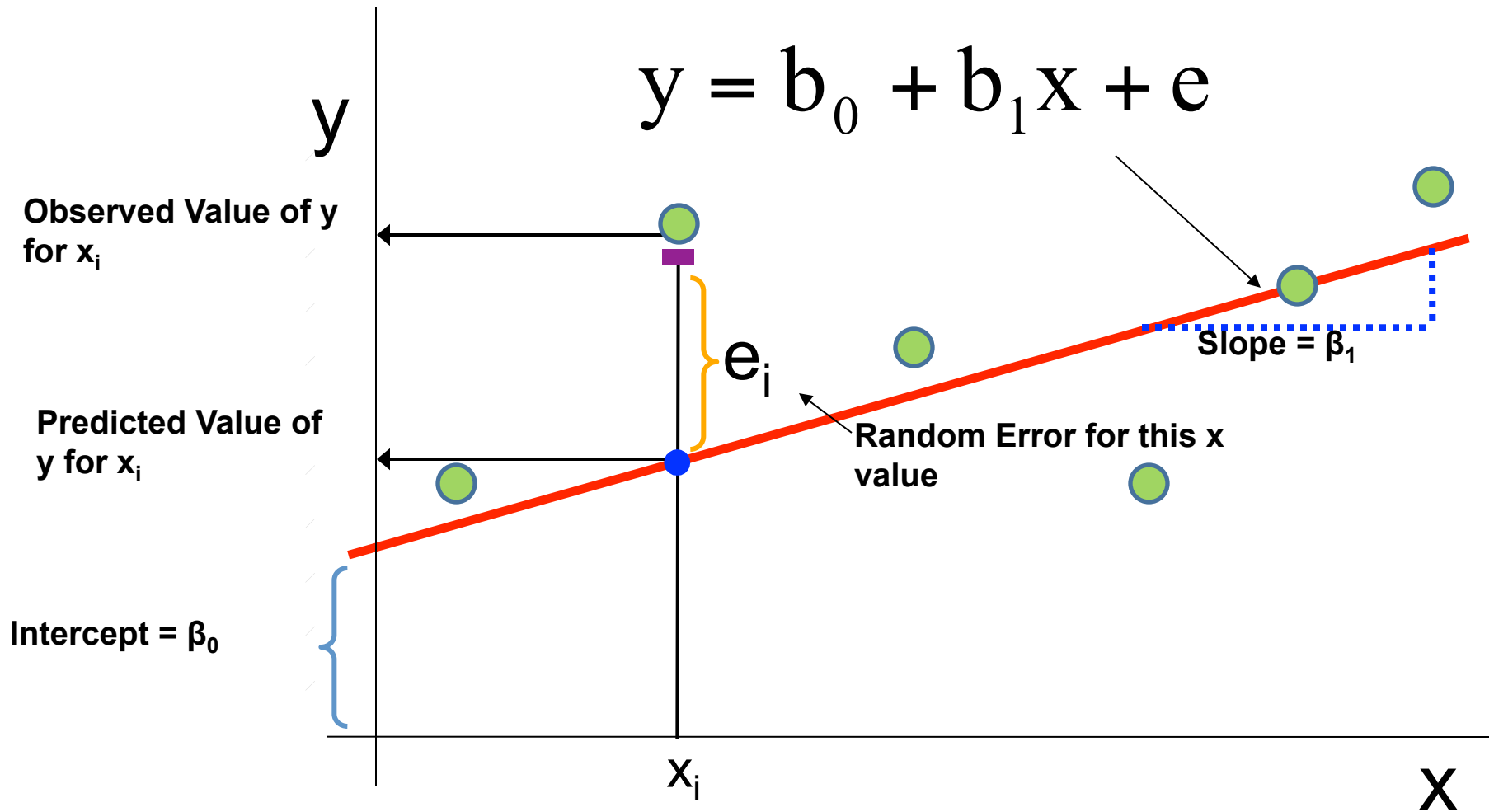
Multiple regression: $Y = b_0 + b_1 X_1 + b_2 X_2.$

- Many nonlinear functions can be transformed into the above.

Log-linear models:

- The multi-way table of joint probabilities is approximated by a product of lower-order tables.
- Probability: $p(a, b, c, d) = \alpha_{ab} \beta_{ac} \chi_{ad} \delta_{bcd}$

Sample Regression Function



Assumptions of Multiple Regression Model

- There exists a linear relationship between the dependent and independent variables.
- The expected value of the error term, conditional on the independent variables is zero.
- The error terms are homoskedastic, i.e. the variance of the error terms is constant for all the observations.
- The expected value of the product of error terms is always zero, which implies that the error terms are uncorrelated with each other.
- The error term is normally distributed.
- The independent variables doesn't have any linear relationships between each other.

Standard Error of Estimate

Standard Error of Estimate (also called the **standard error of regression**) - used to measure how accurately a regression model fits the data.

Formula:

$$SEE = \left(\frac{\sum_{i=1}^n (Y_i - \hat{b}_0 - \hat{b}_1 X_i)^2}{n-2} \right)^{1/2} = \left(\frac{\sum_{i=1}^n (\hat{\epsilon}_i)^2}{n-2} \right)^{1/2}$$

Calculating ANOVA in Regression Analysis

Analysis of Variance (ANOVA) - a statistical procedure that is used to determine how well the independent variable or variables explain the variation in the dependant variable.

F-test - the statistical test that is used in the analysis of the variance

F-test

- A F-statistic is used to test whether the slope coefficients in a linear regression are equal to 0 or not.
- In a regression equation with one independent variable:
 - Null Hypothesis $H_0 : b_1 = 0$ (for multiple regression model, all the slope coefficients are equal to 0)
 - Alternative Hypothesis $H_a : b_1 \neq 0$ (for multiple regression model, at least one slope coefficient is not equal to 0)
- Things required to undertake an F-test
 - the total number of observations
 - the total number of parameters to be estimated
 - the sum of squared errors (SSE)
 - regression sum of squares (RSS)

F-test (one independent variable)

Formula: SSE

$$\sum_{i=1}^n (Y_i - \hat{Y}_i)^2$$

Formula: RSS

$$\sum_{i=1}^n (\hat{Y}_i - \bar{Y})^2$$

Formula: Total Variation

$$(TSS) = SSE + RSS$$

Formula: F-statistic in a regression with one independent variable

$$F = \frac{RSS / 1}{SSE / (n - 2)}$$

F-statistic in Multiple Regression Analysis

Things required for F-test

- Total number of observations (n).
- Total number of regression coefficients to be estimated (k +1) where k is number of slope coefficients.
- Sum of squared errors (SSE) (Unexplained Variation)

$$\sum_{i=1}^n (Y_i - \hat{Y}_i)^2 = \sum_{i=1}^n \hat{\epsilon}_i^2$$

- Regression sum of squares (RSS) (Explained Variation)

$$\sum_{i=1}^n (\hat{Y}_i - \bar{Y})^2$$

F-statistic in Multiple Regression Analysis

Calculating the F-statistic

$$F = \frac{\frac{RSS}{k}}{\frac{SSE}{[n - (k + 1)]}} = \frac{\text{Mean Regression Sum of squares}}{\text{Mean squared error}}$$

Degrees of freedom in the test

- 1) k (numerator degrees of freedom)
- 2) $n - (k + 1)$ (denominator degrees of freedom)

Multicollinearity in Regression Analysis

Multicollinearity - a violation of the regression assumption that there is no exact linear relationship between two or more independent variables

Consequences of Multicollinearity

Estimates of regression coefficients become unreliable.

It is not possible to ascertain how individual independent variables affect dependent variables.

Model Misspecification in Regression Analysis

Model specification - the set of variables that are included in the regression and the regression equation's functional form

Misspecified Functional Form

It omits one or more important variables from regression.

One or more regression variables are required to be transformed before estimating the regression.

Data has been pooled from different samples that are not to be pooled.

Calculating the Predicted Trend Value for a Time Series

Linear Trend Models - the dependent variable changes at a constant rate with time

Formula: $y_t = b_0 + b_1t + \varepsilon_t$, $t = 1, 2, \dots, T$

Where: y_t - value of the time series at time t

b_0 - the y-intercept term

b_1 - the slope coefficient (trend coefficient)

t - time (independent variable)

ε_t - a random error term

Calculating the Predicted Trend Value for a Time Series

Log-Linear Trend Models - used when the time series tends to grow at a constant rate

Formula: $\ln y_t = b_0 + b_1 t + \varepsilon_t, t = 1, 2, \dots, T$

Predicted trend value of y_t is $e^{\hat{b}_0 + \hat{b}_1 t}$

Feature Engineering

Data pre-processing

- Lower case & remove accentuated chars:

```
import unicodedata
```

```
s = ".join(c for c in unicodedata.normalize('NFD', s.lower())  
         if unicodedata.category(c) != 'Mn')
```

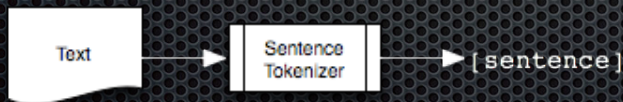
- Extract only word tokens of at least 2 chars
 - Using NLTK tokenizers & stemmers
 - Using a simple regexp:

```
re.compile(r"\b\w\w+\b", re.U).findall(s)
```

Sentence Tokenization

```
>>> from nltk.tokenize import sent_tokenize
>>> sent_tokenize("Hello SF Python. This is NLTK.")
['Hello SF Python.', 'This is NLTK.']

>>> sent_tokenize("Hello, Mr. Anderson. We missed you!")
['Hello, Mr. Anderson.', 'We missed you!']
```



Constructing features (text)

- Tokenize document: list of **uni-grams**

`['the', 'quick', 'brown', 'fox', 'jumps', 'over', 'the', 'lazy', 'dog']`

- **Binary occurrences / counts:**

`{'the': True, 'quick': True...}`

- **Frequencies:**

`{'the': 0.22, 'quick': 0.11, 'brown': 0.11, 'fox': 0.11...}`

- **TF-IDF**

`{'the': 0.001, 'quick': 0.05, 'brown': 0.06, 'fox': 0.24...}`

Constructing features (text)

- Term Frequency

$$tf_{i,j} = \frac{n_{i,j}}{\sum_k n_{k,j}}$$

- Inverse Document Frequency

$$idf_i = \log \frac{|D|}{|\{d : t_i \in d\}|}$$

- Non informative words such as “the” are scaled down

Word Tokenization

```
>>> from nltk.tokenize import word_tokenize
>>> word_tokenize('This is NLTK.')
['This', 'is', 'NLTK', '.']
```



Constructing features (text)

- **bi-grams of words:**
 - “New York”, “very bad”, “not good”
- **n-grams of chars:**
 - “the”, “ed ”, “ a ” (useful for language guessing)
- **Combine with:**
 - **Binary occurrences**
 - **Frequencies**
 - **TF-IDF**

Feature extraction in Python

Unigram features

```
def word_features(words):  
    return dict((word, True) for word in words)
```

Bigram Collocations

```
from nltk.collocations import BigramCollocationFinder  
from nltk.metrics import BigramAssocMeasures as BAM  
from itertools import chain
```

```
def bigram_features(words, score_fn=BAM.chi_sq):  
    bg_finder = BigramCollocationFinder.from_words(words)  
    bigrams = bg_finder.nbest(score_fn, 100000)  
    return dict((bg, True) for bg in chain(words, bigrams))
```

Complete sentences are composed of two or more “phrases”.

Noun phrase:

- **Jack and Jill** went up the hill

Prepositional phrase:

- Contains a noun, preposition and in most cases an adjective
- **The book is on the table** but perhaps it is best kept in a bookshelf

Gerund Phrase:

- Phrases that contain “-ing” verbs
- Jack fell down and broke his crown and **Jill came tumbling after**

Why Part-of-Speech Tag?

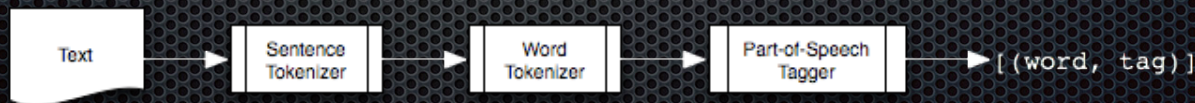
- ✦ word definition lookup (WordNet, WordNik)
- ✦ fine-grained text analytics
- ✦ part-of-speech specific keyword analysis
- ✦ chunking & named entity recognition (NER)



Part-of-Speech Tagging

```
>>> words = word_tokenize("And now for something completely  
different")  
>>> from nltk.tag import pos_tag  
>>> pos_tag(words)  
[('And', 'CC'), ('now', 'RB'), ('for', 'IN'), ('something', 'NN'), ('completely',  
'RB'), ('different', 'JJ')]
```

Tags List: http://www.ling.upenn.edu/courses/Fall_2003/ling001/penn_treebank_pos.html



Take the following sentence

Jack and Jill *went* up **the hill**

```
graph TD; A["Jack and Jill went up the hill"] --> B["Noun phrase"]; A --> C["Noun Phrase"];
```

Noun phrase

Noun Phrase

Chunkers will get us this far:

[Jack and Jill] went up [the hill]

Chunk tokens are non-recursive – meaning, there is no overlap when chunking

The recursive form for the same sentence is:

(Jack and Jill went up (the hill))

Verb phrase chunking

Jack and Jill **went up the hill to fetch a pail of water**



Verb Phrase

Verb Phrase


```
from nltk.chunk import *
from nltk.chunk.util import *
from nltk.chunk.regexp import *
from nltk import word_tokenize, pos_tag
```

```
text = '''
Jack and Jill went up the hill to fetch a pail of water
'''
```

```
tokens = pos_tag(word_tokenize(text))
```

```
chunk = ChunkRule("<.*>+", "Chunk all the text")
```

```
chink = ChinkRule("<VBD|IN|\.>", "Verbs/Props")
```

```
split = SplitRule("<DT><NN>", "<DT><NN>", "determiner+noun")
```

```
chunker = RegexChunkParser([chunk, chink, split], chunk_node='NP')
```

```
chunked = chunker.parse(tokens)
```

```
chunked.draw()
```

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... x +

http://scikit-learn.sourceforge.net/ Google

scikits learn

Download Support User Guide Examples Development

Google™ Custom Search Search x

scikits.learn: machine learning in Python

News

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

Videos

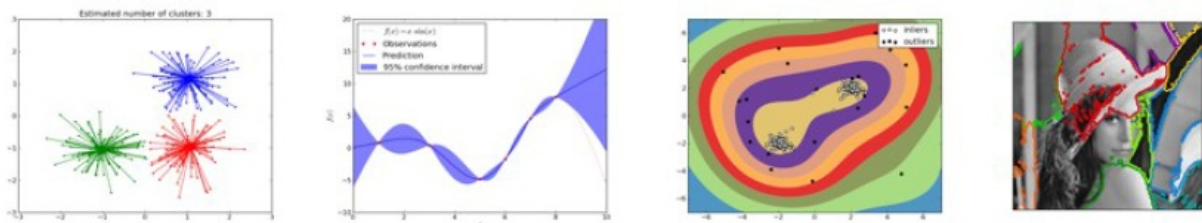
Watch the 2010 ICML Introductory Video by Gaël Varoquaux.

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
Development



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*
- **Planned:** 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 mangé du kangourou ce midi,"  
        u" c'était pas très 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)
```


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)
```

```
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

- $precision = \frac{tp}{tp+fp}$

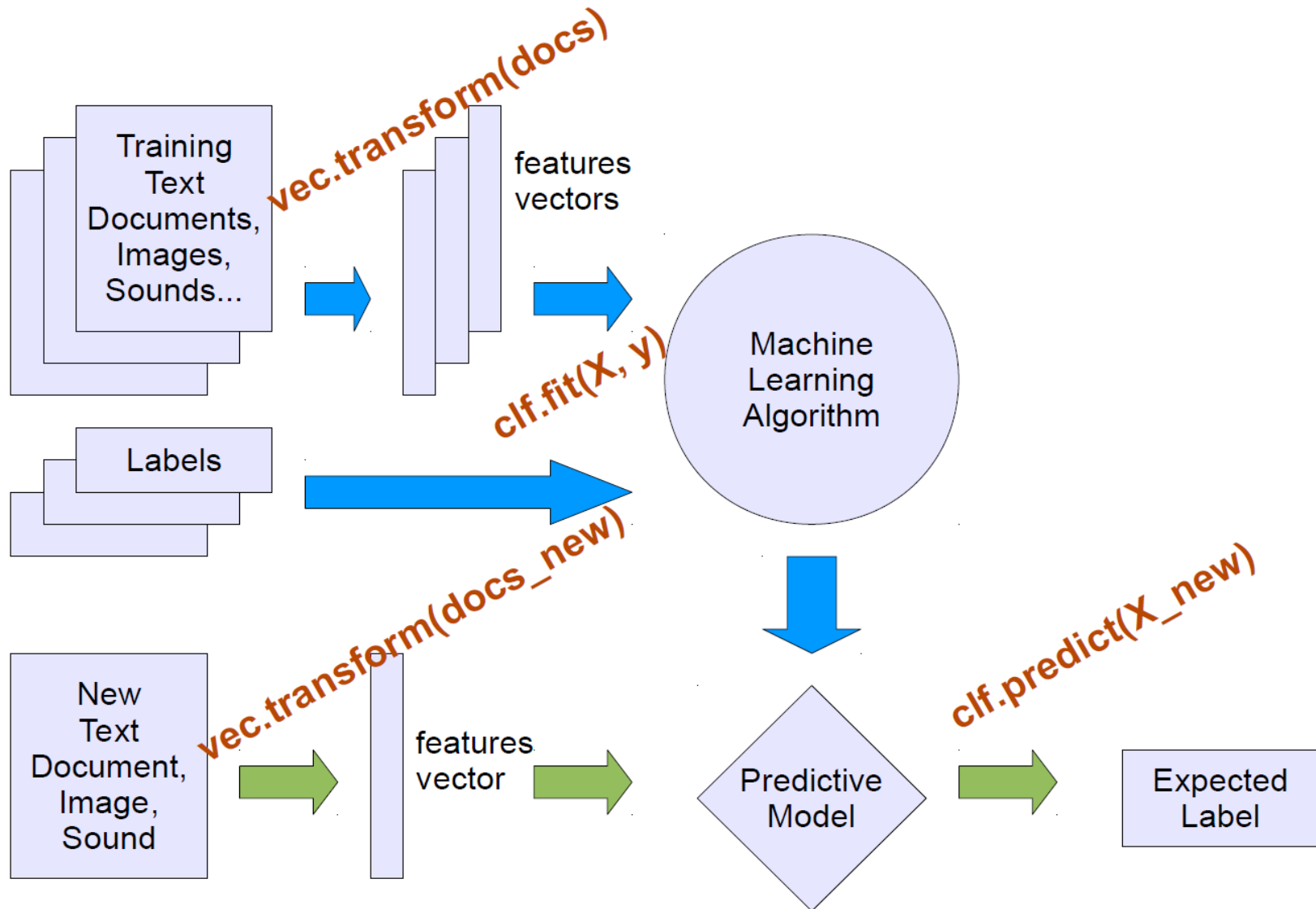
- $recall = \frac{tp}{tp+fn}$

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

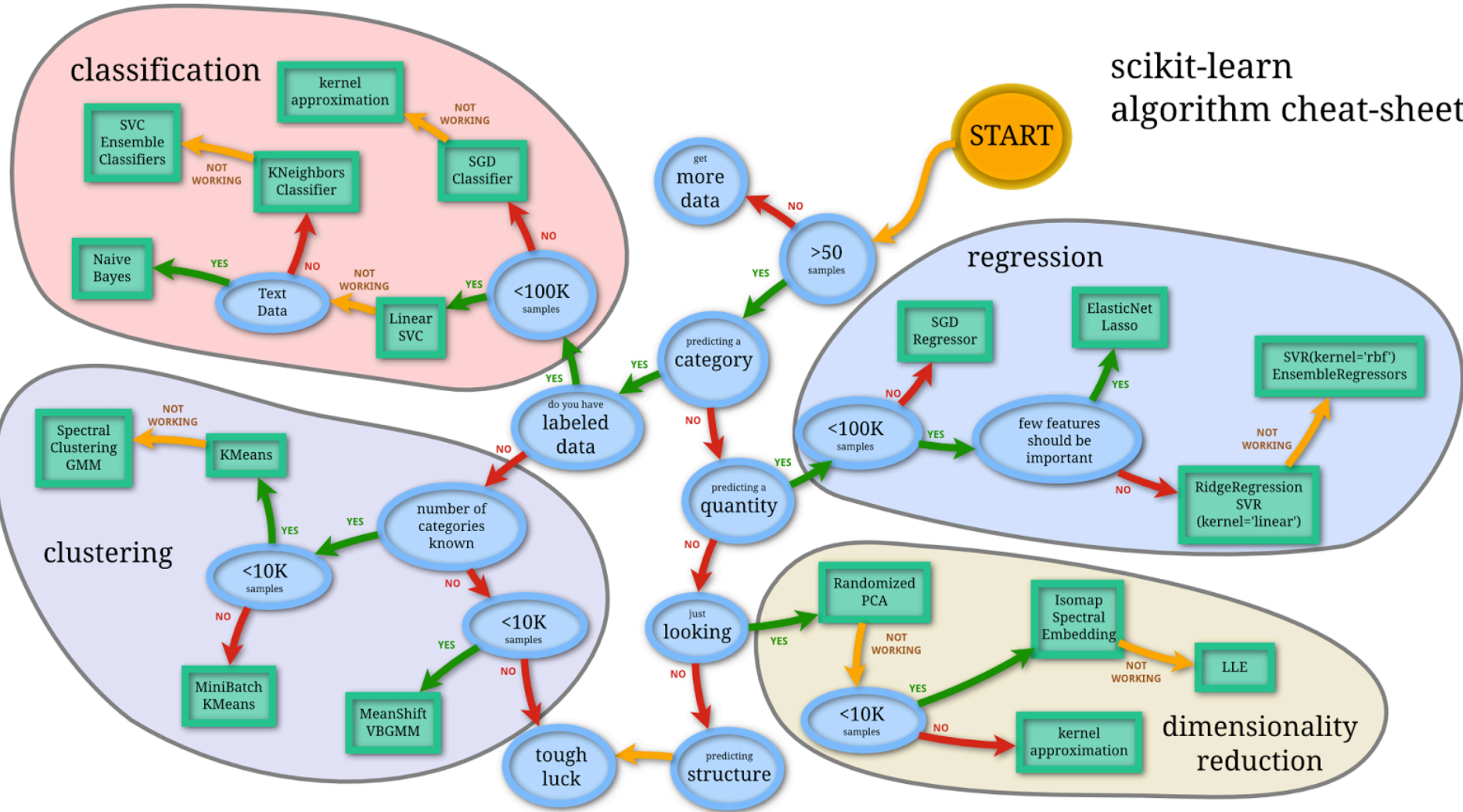
- sklearn.metrics
 - precision_score
 - recall_score
 - f1_score

	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



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>

Next class

- Monday 9/22 (topic: Text—Polarity/Affect)
- **There are assigned readings on the topic; check class website**