CMSC5733 Social Computing

Tutorial VI: Dato and NLTK

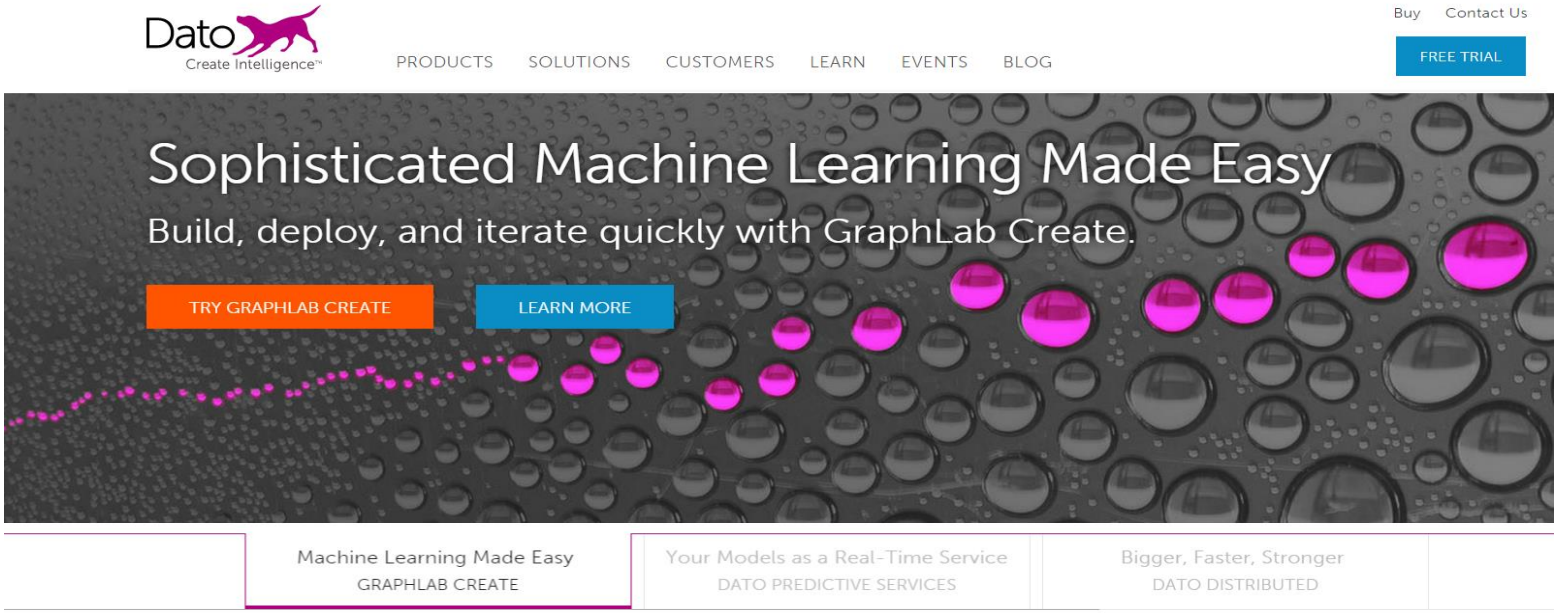
Shenglin Zhao

The Chinese University of Hong Kong

slzhao@cse.cuhk.edu.hk

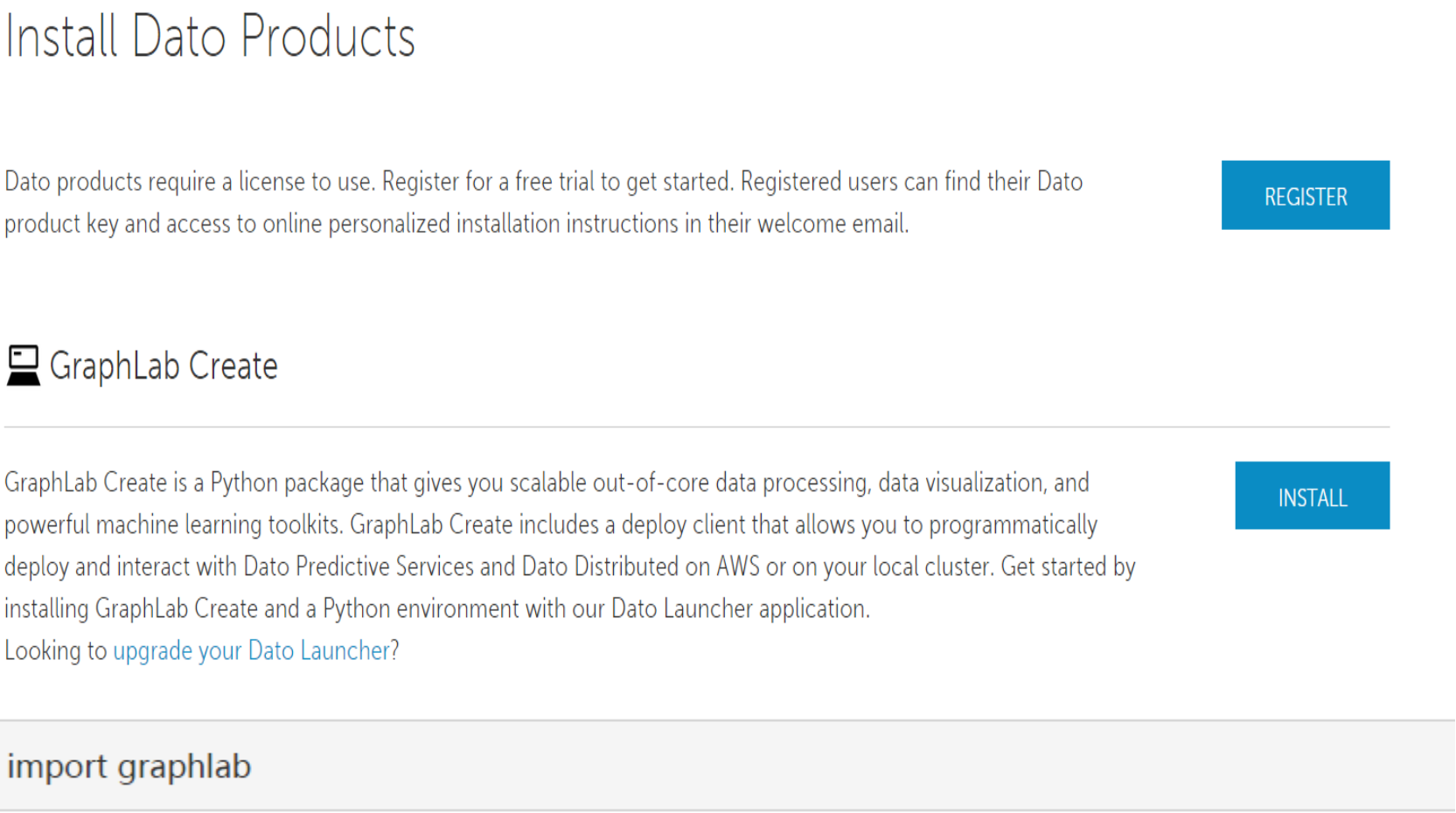
Dato (Graphlab)

- GraphLab Create is a Python package that gives you scalable out-of-core data processing, data visualization, and powerful machine learning toolkits. https://dato.com/



Install

- https://dato.com/download/install.html



Features

- Data Engineering
  - Data extraction, transformation, and loading (ETL)
  - Visualization
- Data Intelligence
  - Building recommenders
  - Analyzing images with deep learning
  - Analyzing unstructured text
  - Outcome prediction
- Deployment
  - Creating predictive services

Quick Start for Dato

- Importing and parsing data

```
url = 'http://s3.amazonaws.com/dato-datasets/millionsong/song_data.csv'
songs = graphlab.SFrame.read_csv(url)
```

- Visualizations

```
songs.show()
```

- Computation on columns

```
songs['year'].mean()
```

Quick Start for Dato

- Transforming columns

```
songs['num_words'] = songs['title'].apply(lambda x: len(x.split(' ')))
```

- Aggregations

```
songs.groupby('artist_name', {'total': graphlab.aggregate.COUNT})
```

- Create models

```
url = 'http://s3.amazonaws.com/dato-datasets/regression/Housing.csv'
x = graphlab.SFrame.read_csv(url)
m = graphlab.linear_regression.create(x, target='price')
```

Build Recommenders

- Three steps:
  - load data
  - create a recommender model
  - start making recommendations

Toy Example

Demo

```
data = graphlab.SFrame({'user_id': ["Ann", "Ann", "Ann", "Brian", "Brian", "Briar
                        'item_id': ["Item1", "Item2", "Item4", "Item2", "Item3"
                        'rating': [1, 3, 2, 5, 4, 2]})
m = graphlab.factorization_recommender.create(data, target='rating')

recommendations = m.recommend()
```

Explanation-SFrame

graphlab.SFrame

class `graphlab.` `SFrame` *(data=list(), format='auto')*

A tabular, column-mutable dataframe object that can scale to big data. The data in SFrame is stored column-wise on the GraphLab Server side, and is stored on persistent storage (e.g. disk) to avoid being constrained by memory size. Each column in an SFrame is a size-immutable `SArray`, but SFrames are mutable in that columns can be added and subtracted with ease. An SFrame essentially acts as an ordered dict of SArrays.

Currently, we support constructing an SFrame from the following data formats:

- csv file (comma separated value)
- sframe directory archive (A directory where an sframe was saved previously)
- general text file (with csv parsing options, See `read_csv()`)
- a Python dictionary
- pandas.DataFrame
- JSON
- Apache Avro
- PySpark RDD

and from the following sources:

- your local file system
- the GraphLab Server's file system
- HDFS
- Amazon S3
- HTTP(S).

Only basic examples of construction are covered here. For more information and examples, please see the User Guide, API Translator, How-Tos, and data science Gallery.

More example: https://dato.com/products/create/docs/generated/graphlab.SFrame.html

Explanation-Recommender

item similarity models

`item_similarity_recommender.create`	Create a recommender that uses item-item similar
`item_similarity_recommender.get_default_options`	Get the default options for the toolkit `ItemSimilarity`
`item_similarity_recommender.ItemSimilarityRecommender`	A model that ranks an item according to its similari

factorization recommenders

`factorization_recommender.create`	Create a FactorizationRecommender that learns laten
`factorization_recommender.get_default_options`	Get the default options for the toolkit `FactorizationRec`
`factorization_recommender.FactorizationRecommender`	A FactorizationRecommender learns latent factors for

factorization recommenders for ranking

`ranking_factorization_recommender.create`	Create a RankingFactorizationRecomme
`ranking_factorization_recommender.get_default_options`	Get the default options for the toolkit `Ra`
`ranking_factorization_recommender.RankingFactorizationRecommender`	A RankingFactorizationRecommender le

More: https://dato.com/products/create/docs/graphlab.toolkits.recommender.html

More Tasks

- Making recommendations for specific users

```
recommendations = m.recommend(users=['Brian'])
```

- Making recommendations for specific users and items

```
country = 'United States'
m.recommend(users=users['user_id'][users['country']==country].unique(),
            items=items['item_id'][items['country']==country])
```

Build Real-life Recommenders

- Load data

```
# Download data if you haven't already
import os
if os.path.exists('movie_ratings'):
    data = graphlab.SFrame('movie_ratings')
else:

    data = graphlab.SFrame.read_csv("http://s3.amazonaws.com/dato-datasets/movie_
    data .save('movie_ratings')

data.head()
```

```
data = graphlab.SFrame.read_csv("http://s3.amazonaws.com/dato-datasets/movie_ratings/training_data.csv", column_type_hints={"rating":int})
```

Build Real-life Recommenders

- Data format

user	movie	rating
Jacob Smith	Flirting with Disaster	4
Jacob Smith	Indecent Proposal	3
Jacob Smith	Runaway Bride	2
Jacob Smith	Swiss Family Robinson	1
Jacob Smith	The Mexican	2
Jacob Smith	Maid in Manhattan	4
Jacob Smith	A Charlie Brown Thanksgiving / The ...	3
Jacob Smith	Brazil	1
Jacob Smith	Forrest Gump	3
Jacob Smith	It Happened One Night	4

Build Real-life Recommenders

- Create a default recommender model

```
# The data needs to contain at least three columns: user, movie, and rating.
model = graphlab.recommender.create(data,
                                    user_id="user",
                                    item_id="movie",
                                    target="rating")
```

Build Real-life Recommenders

- Make recommendations

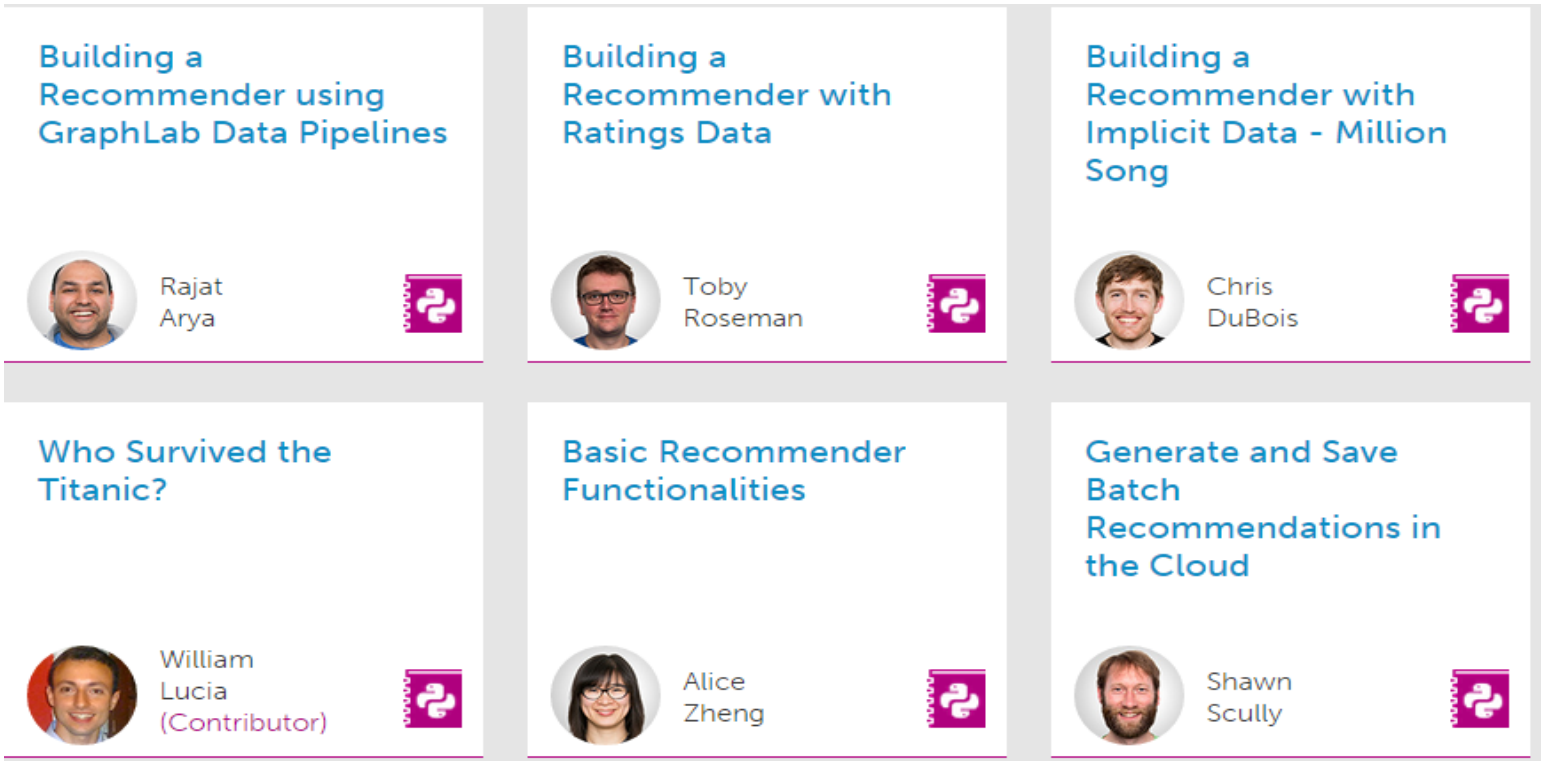
```
# You can now make recommendations for all the users you've just trained on
results = model.recommend(users=None, k=5)
results.head()
```

- Save the model

```
# Save the model for later use
model.save("my_model")
```

More examples

- https://dato.com/learn/gallery/index.html?tagsApplied=Recommender%20Systems&mediaKindsApplied=notebooks



NLTK

- NLTK is a leading platform for building Python programs to work with human language data. It provides easy-to-use interfaces to over 50 corpora and lexical resources such as WordNet, along with a suite of text processing libraries for classification, tokenization, stemming, tagging, parsing, and semantic reasoning, wrappers for industrial-strength NLP libraries, and an active discussion forum.
- http://www.nltk.org/

Installing NLTK

- Install NLTK
  - sudo pip install -U nltk
- Install Numpy (optional)
  - sudo pip install -U numpy
- Test installation:
  - run python then type import nltk

Installing NLTK Data

- Download data, e.g., corpus in NLTK

```
>>> import nltk
>>> nltk.download()
```

- Download via a proxy

```
>>> nltk.set_proxy('http://proxy.example.com:3128', ('USERNAME', 'PASSWORD'))
>>> nltk.download()
```

- Test for a package

```
>>> from nltk.corpus import brown
>>> brown.words()
['The', 'Fulton', 'County', 'Grand', 'Jury', 'said', ...]
```

Sentiment Analysis

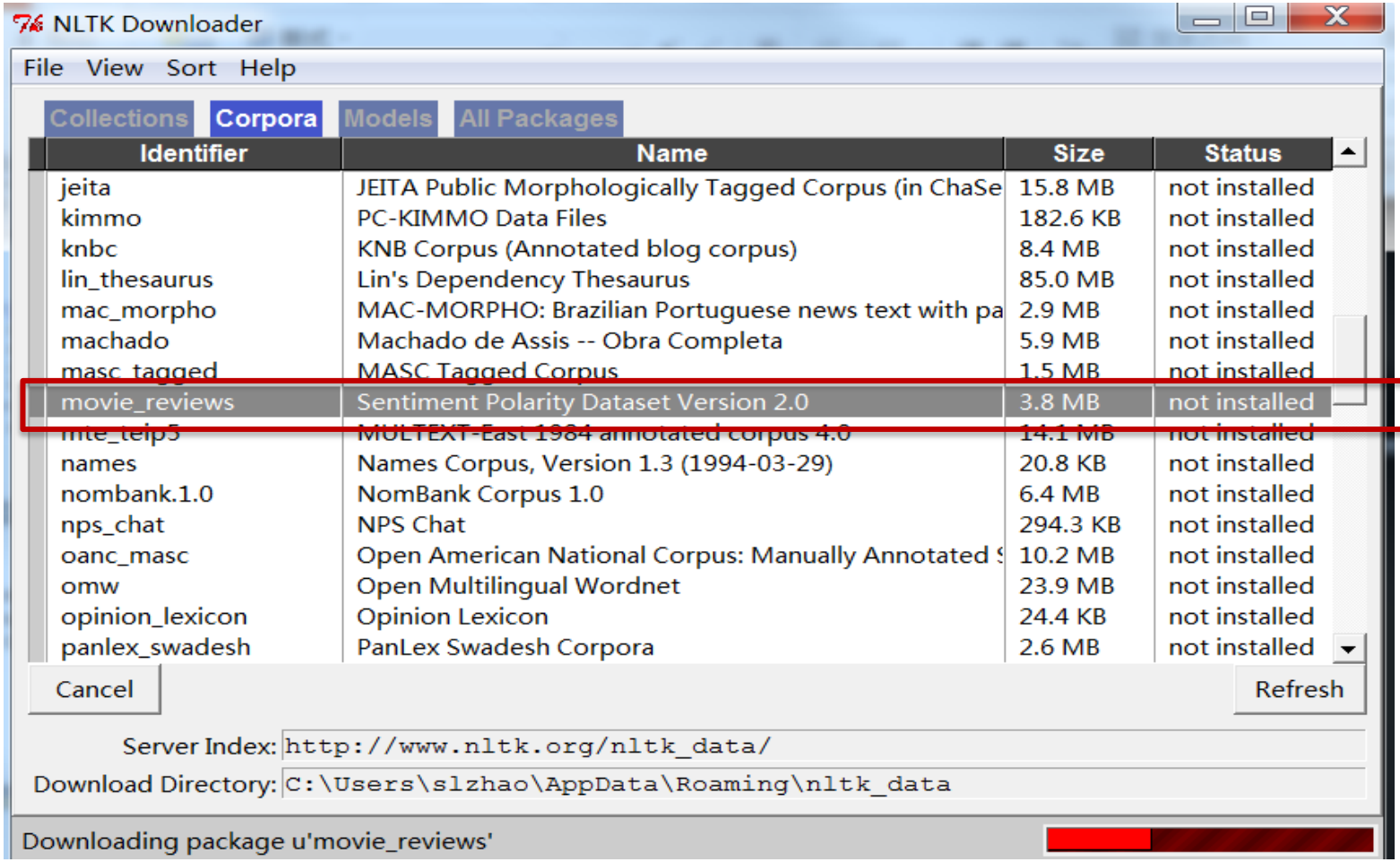
- Sentiment analysis is becoming a popular area of research and social media analysis, especially around user reviews and tweets. It is a special case of text mining generally focused on identifying opinion polarity.
- For simplicity we focus on 2 possible sentiment classifications: *positive* and *negative*.

Simple Sentiment Analysis

- Prepare data
- Feature extractor
- Classifier (NBC)
- Evaluation

Prepare Data

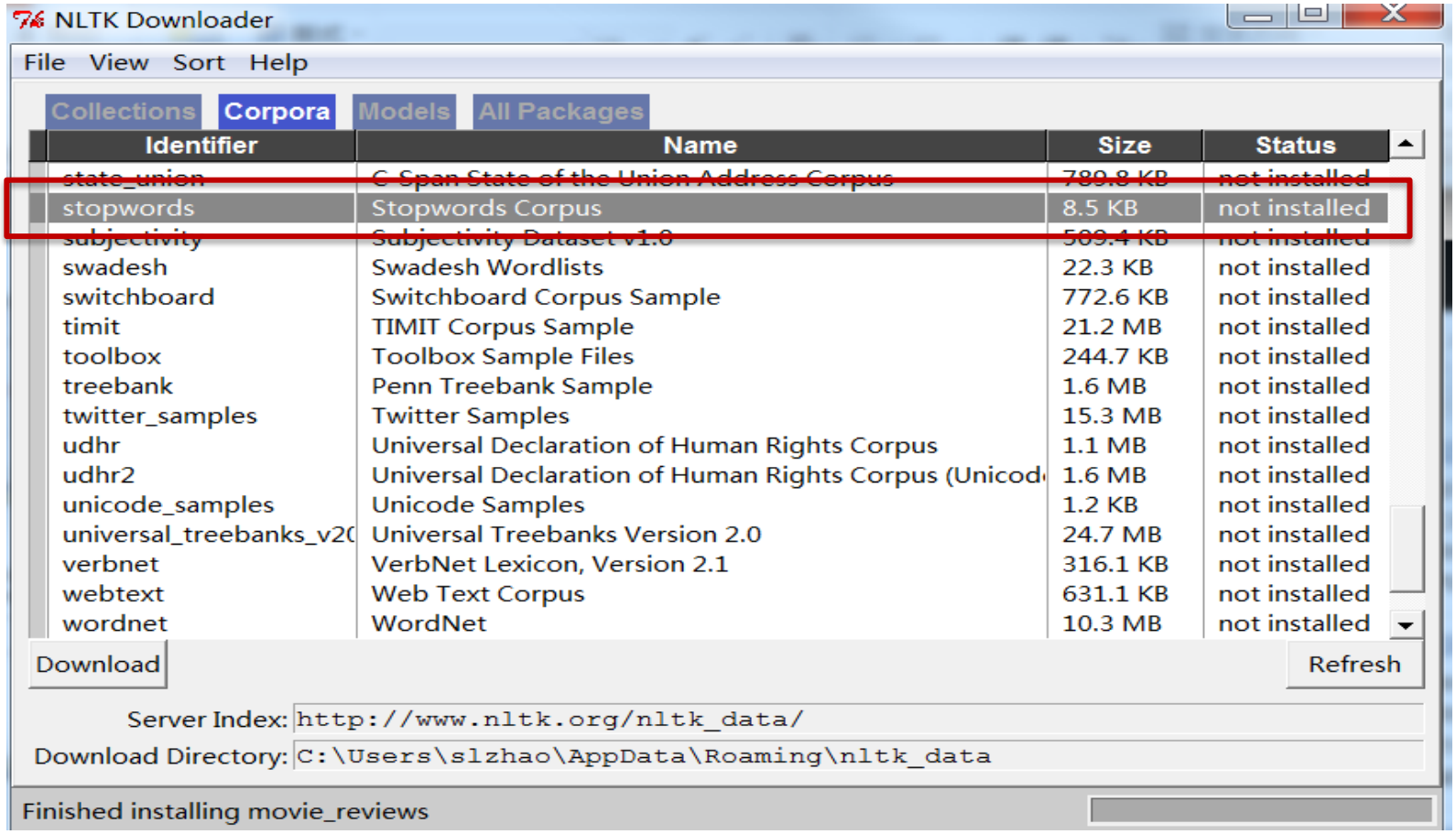
- A movie reviews corpus from NLTK



Prepare Data

- Split to get train and test data
- Filter stop words (optional)
  - **stop words** are words which are filtered out before or after processing of natural language data
  - example: the, a, who, etc.

Stop Words in NLTK



Stop Words in NLTK

```
In [10]: from nltk.corpus import stopwords

In [11]: swlist = stopwords.words('english')

In [12]: swlist[:10]
Out[12]:
[u'i',
 u'me',
 u'my',
 u'myself',
 u'we',
 u'our',
 u'ours',
 u'ourselves',
 u'you',
 u'your']

In [13]:
```

Feature Extractor

- Bag of words representation

I love this movie! It's sweet, but with satirical humor. The dialogue is great and the adventure scenes are fun… It manages to be whimsical and romantic while laughing at the conventions of the fairy tale genre. I would recommend it to just about anyone. I've seen it several times, and I'm always happy to see it again whenever I have a friend who hasn't seen it yet.

great	2
love	2
recommend	1
laugh	1
happy	1
...	...

Feature Extractor

- Bag of Words Feature Extraction
  - Every word is a feature name with a value of True.
  - Feature extraction method:

```
def word_feats(words):
        return dict([(word, True) for word in words])
```

Supervised Text Classification

- Given:
  - A document *d*
  - A fixed set of classes:

    $C = \{c_1, c_2, \ldots, c_J\}$
  - A training set *D* of documents each with a label in *C*
- Determine:
  - A learning method or algorithm which will enable us to learn a classifier γ
  - For a test document *d*, we assign it the class

    $\gamma(d) \in C$

Text Classifier in NLTK

Search

From here you can search these documents. Enter your search words into the box below and click "search". Note that the search function will automatically search for all of the words. Pages containing fewer words won't appear in the result list.

nltk.classify search

Search Results

Search finished, found 178 page(s) matching the search query.

- **nltk.classify** (Python module, in nltk.classify package)
- **nltk.classify.api** (Python module, in nltk.classify package)
- **nltk.classify.decisiontree** (Python module, in nltk.classify package)
- **nltk.classify.maxent** (Python module, in nltk.classify package)
- **nltk.classify.megam** (Python module, in nltk.classify package)
- **nltk.classify.naivebayes** (Python module, in nltk.classify package)
- **nltk.classify.positivenaivebayes** (Python module, in nltk.classify package)

Naïve Bayesian Classifier

nltk.classify.naivebayes module

A classifier based on the Naive Bayes algorithm. In order to find the probability for a label, this algorithm first uses the Bayes rule to express P(label|features) in terms of P(label) and P(features|label):

$$P(\text{label}|\text{features}) = \frac{P(\text{label}) * P(\text{features}|\text{label})}{P(\text{features})}$$

The algorithm then makes the 'naive' assumption that all features are independent, given the label:

$$P(\text{label}|\text{features}) = \frac{P(\text{label}) * P(f1|\text{label}) * \ldots * P(fn|\text{label})}{P(\text{features})}$$

Rather than computing P(featues) explicitly, the algorithm just calculates the denominator for each label, and normalizes them so they sum to one:

$$P(\text{label}|\text{features}) = \frac{P(\text{label}) * P(f1|\text{label}) * \ldots * P(fn|\text{label})}{SUM[l](P(l) * P(f1|l) * \ldots * P(fn|l))}$$

Evaluation

- Classification accuracy: *r/n*
  - where n is the total number of test docs and *r* is the number of test docs correctly classified

- More measures: precision, recall, and F-score

Demo

Scikit-learn in NLTK

- Scikit-learn is a machine learning package in Python, http://scikit-learn.org/stable/index.html
  - Simple and efficient tools for data mining and data analysis
  - Accessible to everybody, and reusable in various contexts
  - Built on NumPy, SciPy, and matplotlib
  - Open source, commercially usable - BSD license

Scikit-learn for ML

Classification

Identifying to which category an object belongs to.

Applications: Spam detection, Image recognition.
Algorithms: *SVM*, *nearest neighbors*, *random forest*, ...

— *Examples*

Regression

Predicting a continuous-valued attribute associated with an object.

Applications: Drug response, Stock prices.
Algorithms: *SVR*, *ridge regression*, *Lasso*, ...

— *Examples*

Clustering

Automatic grouping of similar objects into sets.

Applications: Customer segmentation, Grouping experiment outcomes
Algorithms: *k-Means*, *spectral clustering*, *mean-shift*, ...

— *Examples*

Dimensionality reduction

Reducing the number of random variables to consider.

Applications: Visualization, Increased efficiency
Algorithms: *PCA*, *feature selection*, *non-negative matrix factorization*.

— *Examples*

Model selection

Comparing, validating and choosing parameters and models.

Goal: Improved accuracy via parameter tuning
Modules: *grid search*, *cross validation*, *metrics*.

— *Examples*

Preprocessing

Feature extraction and normalization.

Application: Transforming input data such as text for use with machine learning algorithms.
Modules: *preprocessing*, *feature extraction*.

— *Examples*

Scikit-learn in NLTK

- scikit-learn supports many classification algorithms, including SVMs, Naive Bayes, logistic regression (MaxEnt) and decision trees.

nltk.classify.scikitlearn module ¶

scikit-learn (http://scikit-learn.org) is a machine learning library for Python. It supports many classification algorithms, including SVMs, Naive Bayes, logistic regression (MaxEnt) and decision trees.

This package implement a wrapper around scikit-learn classifiers. To use this wrapper, construct a scikit-learn estimator object, then use that to construct a SklearnClassifier. E.g., to wrap a linear SVM with default settings:

Example

- Define SVM classifier

```
>>> from sklearn.svm import LinearSVC
>>> from nltk.classify.scikitlearn import SklearnClassifier
>>> classif = SklearnClassifier(LinearSVC())
```

Example

- Feature extractor
  - employ tf-idf weighting and chi-square feature selection to get the best 1000 features

```
>>> from sklearn.feature_extraction.text import TfidfTransformer
>>> from sklearn.feature_selection import SelectKBest, chi2
>>> from sklearn.naive_bayes import MultinomialNB
>>> from sklearn.pipeline import Pipeline
>>> pipeline = Pipeline([('tfidf', TfidfTransformer()),
...                      ('chi2', SelectKBest(chi2, k=1000)),
...                      ('nb', MultinomialNB())])
>>> classif = SklearnClassifier(pipeline)
```

References

- https://dato.com/learn/gallery/notebooks/five_line_recommender.html
- https://dato.com/
- https://dato.com/learn/userguide/recommender/making-recommendations.html
- https://dato.com/learn/userguide/recommender/introduction.html
- http://streamhacker.com/2010/05/10/text-classification-sentiment-analysis-naive-bayes-classifier/
- https://en.wikipedia.org/wiki/Stop_words
- http://www.nltk.org/api/nltk.classify.html?highlight=nltk.classify#module-nltk.classify.scikitlearn
- http://www.nltk.org/api/nltk.classify.html?highlight=nltk.classify#module-nltk.classify.naivebayes
- http://scikit-learn.org/stable/index.html