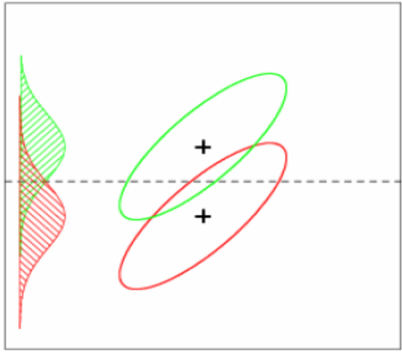
Chap4-2.

2조 HIM
TAEKOAN YOO 유태관

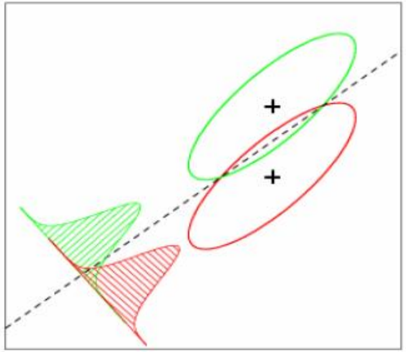
0. Background

- **Curse of dimensionality**
  - Dimension ↑ : hard to analyze
  - Dimension Reduction
    - Feature Extraction : M feature → N feature (mapping)
      - LSA : ?, SVD-based, TF-IDF/BoW
      - PCA : for accuracy, SVD-based, image
      - LDA : class-discriminatory, SVD-X, TF-IDF
    - Feature Selection : M feature → N feature (select)
      - SFS (Sequential Forward Selection)
      - SBS (Sequential Backward Selection)
      - LRS (Plus L minus R Selection)

PCA



LDA



Outline

- **Background**
- **1. PCA**
- **2. LSA vs PCA**
- **3. SVD Improvement**
- **4. Similarity Measure**
- **5. LDA**
- **Conclusion**

1. PCA

- **PCA**
  - 각 축에 대한 min variance가 되는 vector를 찾아 고차원 자료의 '본질 ' 을 포착
  - SVD-based, image

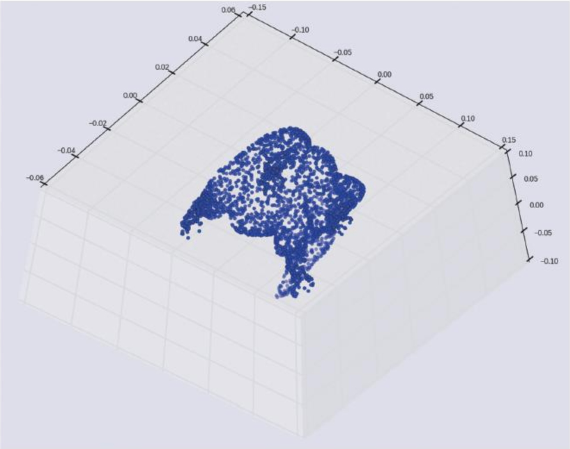


그림 4.4 어떤 물체의 점 구름을 '아랫배(?)' 쪽에서 위로 올려다본 모습.

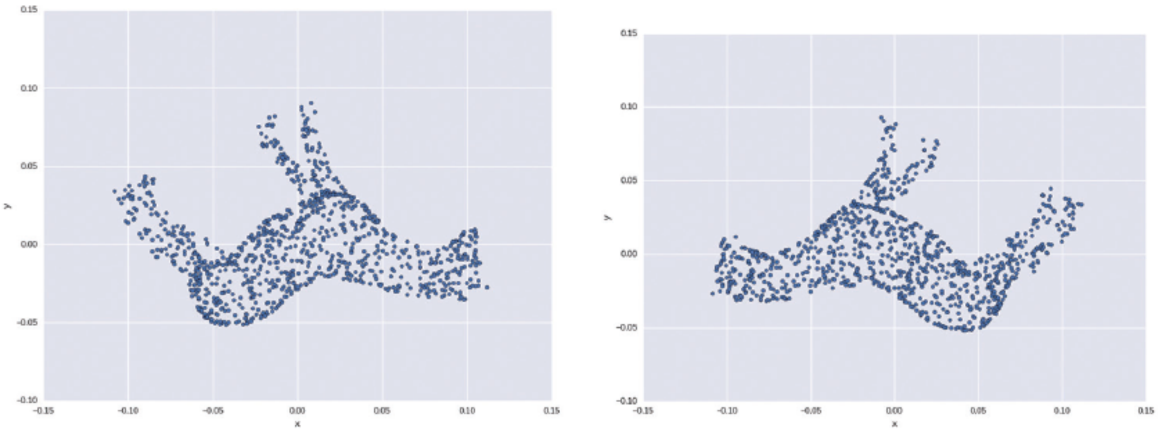


그림 4.5 뒤집힌 말의 점 구름 투영들.

2. LSA vs PCA

- **Problem : Spam vs Ham**
  - 4837 Corpus [9232 token] = 628 spam + 4209 ham
  - Unbalanced Training Set → Overfitting! → Dimensionality Reduction
  - (SVM =) PSA = LSA
    - Input = Tfidf (casual tokenize)
    - Remove Egien Value : 행렬의 회전 성분만 취급하기 위해
      - L2-Norm Normalization
      - Minus average (Centering)
- **PCA**
  - *Sklearn.decomposition.PCA : for dense matrix*
- **LSA**
  - *Sklearn.decomposition.TruncatedSVD : for sparse matrix, fast*

3. SVD Improvement

- **개선안**
  - QDA(Quadratic Discriminant Analysis) :
    - LDA의 대안
    - 1차 Transformation이 아닌, Non-linear Transformation
    - 2차 다항식
  - Random Projection :
    - SVD와 비슷하지만, 확률적으로 변환
    - Probabilistic Transformation
  - NMF(Non-negative Matrix Factorization)
  - LDiA :
    - LSA보다 2배 정확
    - 단어 빈도가 Dirichlet distribution을 따른다고 가정

3. SVD Improvement

- **LDiA**
  - LSA + 단어 빈도들이 Dirichlet distribution을 따른다고 가정
    - LSA : 원래 far => far!!
    - LDiA : 원래 close => close!!
  - Random Seed (주사위)
    - # of corpus (Poisson Distribution)
    - # of topic (Dirichlet distribution)
  - 용어-주제 행렬
    - Topic에 적합한 corpus를 고르는 과정!을 위해 필요
  - *Sklearn.decomposition.LatentDirichletAllocation*

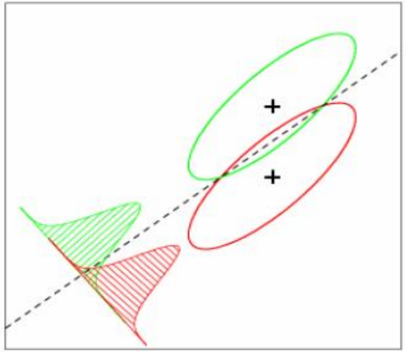
4. Similarity Measure

- **Distance Measure Methods (for similarity)**
  - RMSE (Root Mean Sqaure Error,Euclidean Distance) =$\sqrt{mean({v_i}^2)}$
    - 2-Norm (L_2)
  - SSD (Sum of Sqaures Distance) = $\mathrm{sum}({v_i}^2)$
  - Consine Distance = $\frac{A \bullet B}{|A||B|}$
  - Minkowski Distance
    - p-Norm (L_p)
  - Fractional Distance
    - p-Norm (L_p) , 0<p<1
  - SAD (Sum of Absolute Distance) =
    - 1-norm (L_1)
  - Jaccard Distance (Inverse Set Similarity)
  - Mahalanobis Distance
  - Edit Distance

5. LDA

- **Problem :**
  - No consideration similarity between Documents
  - = No feedback
- **LDA**
  - Comparison
    - LSA : far~~ vectors
    - LDA : far~~ center of gravity
  - 우리가 모형화 하려는 ‘주제’를 알려주는 것이 목표
  - Semantic Vector 중 가장 정확도가 높다.
  - *sklearn.discriminant_analysis.LinearDiscriminantAnalysis*

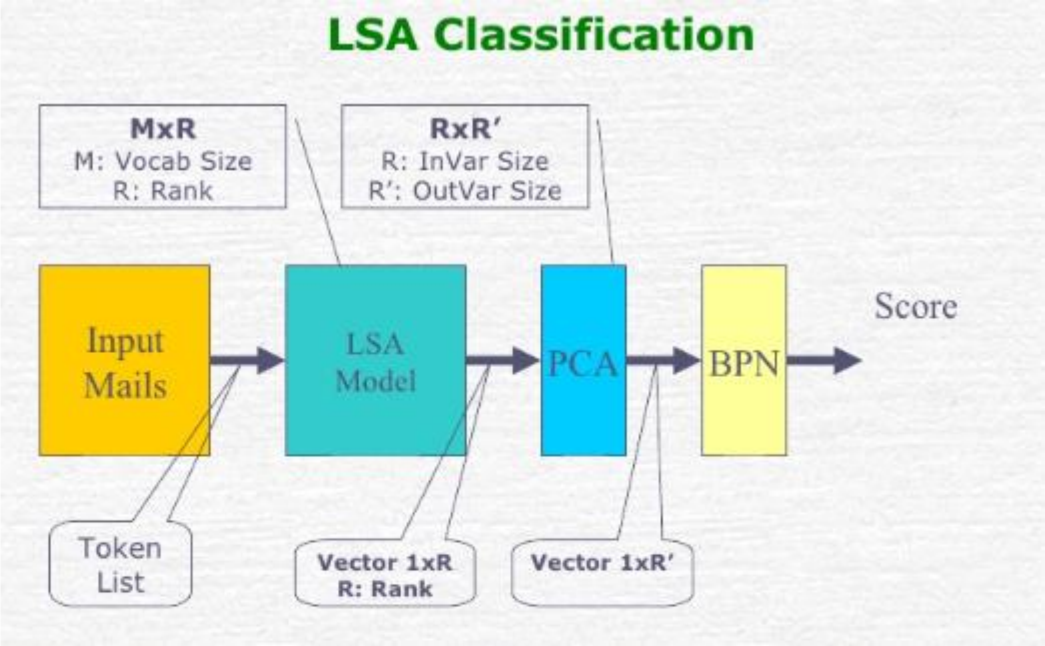
LDA



Conclusion

- **Semantic Vectors 주제 벡터**
  - Geoffry Hinton “14차원 공간의 초평면들을 다루는 방법은 그냥 3차원 공간을 시각화하고 그것이 14차원이라고 크게 외치는 것이다“
  - LSA 잠재의미 분석(잠재의미 인덱싱) : LSA, LDiA --- 주제 벡터 생성
- **Semantic Search 의미기반 검색 : 주제벡터를 가지고 검색**
  - 기존의 잠재의미 인덱싱 방법인 LSH(국소성 민감 해시)는 차원이 올라갈수록 검색의 정확도가 떨어진다
  - LSA, LDiA로 주제 벡터를 만든 후에 완벽한 색인보다는 “충분히 좋은” 색인을 추구

Appendix A.


LSA Classification
MxR
M: Vocab Size
R: Rank
RxR'
R: InVar Size
R': OutVar Size
Input Mails
LSA Model
PCA
BPN
Score
Token List
Vector 1xR
R: Rank
Vector 1xR'

Appendix B.

- **STEP1**

```
!pip install nlpia

import pandas as pd
pd.options.display.width=120
from nlpia.data.loaders import get_data
sms = get_data('sms-spam')
index = ['sms{}{}'.format(i, '!'*j) for (i,j)in zip(range(len(sms)), sms.spam)]
sms.index = index
sms.head(6)
```

	spam	text
sms0	0	Go until jurong point, crazy.. Available only ...
sms1	0	Ok lar... Joking wif u oni...
sms2!	1	Free entry in 2 a wkly comp to win FA Cup fina...
sms3	0	U dun say so early hor... U c already then say...
sms4	0	Nah I don't think he goes to usf, he lives aro...
sms5!	1	FreeMsg Hey there darling it's been 3 week's n...

Appendix B.

- **STEP2. tfidf, BoW**

```
# tfidf
from sklearn.feature_extraction.text import TfidfVectorizer
from nltk.tokenize.casual import casual_tokenize

tfidf = TfidfVectorizer(tokenizer=casual_tokenize)
tfidf_docs = tfidf.fit_transform(raw_documents=sms.text).toarray()
tfidf_docs = pd.DataFrame(tfidf_docs)
tfidf_docs = tfidf_docs - tfidf_docs.mean()

#BoW
from sklearn.feature_extraction.text import CountVectorizer


counter = CountVectorizer(tokenizer=casual_tokenize)
bow_docs = pd.DataFrame(counter.fit_transform(raw_documents=sms.text).toarray(), index=index)
column_nums, terms = zip(*sorted(zip(counter.vocabulary_.values(), counter.vocabulary_.keys())))
bow_docs.column = terms
```

Appendix B.

- **STEP3. PCA**

```
#PCA
from sklearn.decomposition import PCA

pca = PCA(n_components=16)
pca = pca.fit(tfidf_docs)
pca_topic_vectors = pca.transform(tfidf_docs)
columns = ['topic{}'.format(i) for i in range(pca.n_components)]
pca_topic_vectors = pd.DataFrame(pca_topic_vectors, columns=columns, index=index)
pca_topic_vectors.round(3).head(6)
```

	topic0	topic1	topic2	...	topic13	topic14	topic15
sms0	0.201	0.003	0.037	...	-0.031	-0.006	-0.030
sms1	0.404	-0.094	-0.078	...	-0.020	0.036	0.050
sms2!	-0.030	-0.048	0.090	...	-0.021	-0.049	-0.034
sms3	0.329	-0.033	-0.035	...	-0.037	-0.001	0.043
sms4	0.002	0.031	0.038	...	0.043	-0.077	0.019
sms5!	-0.016	0.059	0.014	...	0.063	0.012	-0.049

6 rows × 16 columns

Appendix B.

- **STEP3. LSA**

```
#LSA
from sklearn.decomposition import TruncatedSVD

svd = TruncatedSVD(n_components=16, n_iter=100)
svd_topic_vectors = svd.fit_transform(tfidf_docs.values)
svd_topic_vectors = pd.DataFrame(svd_topic_vectors, columns=columns, index=index)
svd_topic_vectors.round(3).head(6)
```

	topic0	topic1	topic2	...	topic13	topic14	topic15
sms0	0.201	0.003	0.037	...	-0.036	-0.014	0.037
sms1	0.404	-0.094	-0.078	...	-0.021	0.051	-0.042
sms2!	-0.030	-0.048	0.090	...	-0.020	-0.042	0.052
sms3	0.329	-0.033	-0.035	...	-0.046	0.022	-0.070
sms4	0.002	0.031	0.038	...	0.034	-0.083	-0.021
sms5!	-0.016	0.059	0.014	...	0.075	-0.001	0.020

6 rows × 16 columns

Appendix B.

- **STEP3. LDiA**

```
#LDiA
from sklearn.decomposition import LatentDirichletAllocation as LDiA

ldia=LDiA(n_components=16, learning_method='batch')
ldia = ldia.fit(bow_docs)
ldia16_topic_vectors = ldia.transform(bow_docs)
ldia16_topic_vectors = pd.DataFrame(ldia16_topic_vectors, index=index, columns=columns)
ldia16_topic_vectors.round(2).head()
```

	topic0	topic1	topic2	...	topic13	topic14	topic15
sms0	0.00	0.00	0.00	...	0.00	0.00	0.00
sms1	0.01	0.01	0.01	...	0.01	0.01	0.01
sms2!	0.42	0.00	0.00	...	0.00	0.00	0.00
sms3	0.00	0.00	0.00	...	0.00	0.00	0.00
sms4	0.00	0.39	0.00	...	0.00	0.55	0.00

5 rows × 16 columns

Appendix B.

- **STEP3. LDA**

```
#LDA
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis as LDA
from sklearn.model_selection import train_test_split
from sklearn.model_selection import cross_val_score

x_train, x_test, y_train, y_test = train_test_split(pca_topic_vectors.values, sms.spam, test_size=0.3, random_state=271828)
lda = LDA(n_components=1)
lda = lda.fit(x_train, y_train)
lda.score(x_test, y_test).round(3)
```

```
0.965
```