Machine Learning

Assignment 2:-

(1) Vocab size; ⇒ 292 after preprocessing.

Assume each doc multinomial distribution.

$p(x/y) \rightarrow$ multinomial

estimate $P_c \Rightarrow$ parameter for multinomial distribution

& obtain w_c and β_c from $p(y)$ and $\ell p(x/y)$

→ P_c is a vector, ~~for a document~~ for each class, with each entry as occurance of word "i" in all documents of class "j" and normalized with total words in class "j"

$$\therefore \underset{\text{for class "j"}}{P_c} = \left[\frac{\text{word 1 count}}{\text{total words in class "j"}}, \ldots, \frac{\text{word n count}}{\text{total words in class j}} \right]$$

$$\therefore P_c = \frac{1}{\sum_{i \in class\, j} 1^T x_i} \sum_{i \in class\, j} x_i$$

given, x = frequency count, $= [x_1, x_2, \ldots x_m]$

Probability of new document "x" in class c is proportional to,

$$p(x) \propto \pi_c \prod_{j=1}^{m} P_{cj}^{x_j}$$

preventing underflow by taking log,

$$\propto \log\left(\pi_c \prod_{j=1}^{m} P_{cj}^{x_j}\right)$$

$$\propto \log \pi_c + \sum_{j=1}^{m} x_j \log P_{cj}$$

∴ Chosing max of all, we get classifier,

$$\hat{y} = \underset{c}{\arg\max}\left(\log \pi_c + \sum_{j=1}^{m} x_j \log p_{cj}\right)$$

this is of form, $\hat{y} = \underset{c}{\arg\max}\,(w_c^T x + \beta_c)$.

$$\boxed{\therefore w_c(j) = \log p_{cj} \qquad \beta_c = \log \pi_c}$$

(2) $p(x|y) \Rightarrow$ multivariate normal distribution,
covariance matrix same $\Rightarrow \Sigma_1 = \dots = \Sigma_m = \Sigma$

∴ classifier similar to linear Discriminant Analysis (LDA)

parameters $\Rightarrow \mu_c$ and Σ.

⟶

Since bag-of-words doesn't always follow a normal distribution, we convert our dataset as a ~~tfid tf~~ TF-IDF matrix,

$$TF \text{ (term frequency)} = \frac{\#\text{ Term } t \text{ in document } d}{\text{Total terms in document } d}$$

(Importance to more frequent words).

$$IDF \text{ (Inverse document freq)} = \log\left(\frac{\text{Total documents in corpus } D}{\#\text{ documents with term } t}\right)$$

(Reduce impact of high common words like "the", and adds importances to rare words).

The same is implemented in code.

Linear Discriminant Analysis Model (LDA)

parameters $\Rightarrow$ μ_c and Σ

$$\boxed{\mu_c = \frac{1}{n_c} \sum_{i \in class} x_i}$$

$$\Sigma = \frac{1}{n} \sum_{i=1}^{n} (x_i - \mu_c)(x_i - \mu_c)^T$$

$$= \frac{1}{n} \sum x_i x_i^T - x_i \mu_c^T - \mu_c x_i^T + \mu_c \mu_c^T$$

$$= \frac{1}{n} \left[\sum_{i=1}^{n} x_i x_i^T - \underbrace{\sum_{i=1}^{n} x_i}_{n_c \mu_c} \mu_c^T - \sum_{i=1}^{n} \mu_c x_i^T + \underbrace{\sum \mu_c \mu_c^T}_{\text{common}} \right]$$

$$= \frac{1}{n} \left(\sum_{i=1}^{n} x_i x_i^T - \sum n_c \mu_c \mu_c^T - \cancel{\sum \mu_c \mu_c^T} + \cancel{\sum \mu_c \mu_c^T} \right)$$

$$\boxed{\Sigma = \frac{1}{n} \sum_{i=1}^{n} x_i x_i^T - \frac{1}{n} \sum_{c=1}^{k} n_c \mu_c \mu_c^T}$$

The probability that a doc belongs to class,

$$\pi_c \det(2\pi\Sigma)^{-1/2} \exp\left(-\frac{1}{2}(x-\mu_c)^T \Sigma^{-1} (x-\mu_c)\right)$$

Taking log,

$$\log\left(\pi_c \underbrace{\det(2\pi\Sigma)^{-1/2}}_{\substack{\text{Constant} \\ \text{ignore}}} \exp\left(-1/2\, (x-\mu_c)^T \Sigma^{-1} (x-\mu_c)\right)\right)$$

$$\therefore = \log \pi_c + \left(-\frac{1}{2}(x-\mu_c)^T \Sigma^{-1} (x-\mu_c)\right)$$

$$= \log \pi_c + \left(-\frac{1}{2}\left(\underbrace{x^T \Sigma^{-1} x}_{\substack{\text{constant} \\ \text{term}}} - x^T \Sigma^{-1} \mu_c - \mu_c^T \Sigma^{-1} x + \mu_c^T \Sigma^{-1} \mu_c\right)\right)$$

Chosing maximum of all classes gives us the classifier,

$$\boxed{\hat{y} = \underset{c}{\operatorname{argmax}} \left(\log \pi_c - \frac{1}{2} \mu_c^T \Sigma^{-1} \mu_c + \mu_c^T \Sigma^{-1} x \right)}$$

homework_2

February 20, 2025

[106]:
```
import pandas as pd
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis as LDA
from sklearn.naive_bayes import MultinomialNB as MNB
from sklearn.metrics import accuracy_score
import math
import numpy as np
from numpy.linalg import qr, norm
import matplotlib.pyplot as plt
from sklearn.mixture import GaussianMixture
import matplotlib.cm as cm
import urllib
```

[3]:
```
X_train_input = pd.read_csv("20news-bydate/matlab/train.data", delimiter="␣
↪",names = ["docIdx", "wordIdx", "freq"],)
y_train_input = pd.read_csv("20news-bydate/matlab/train.label",␣
↪names=["labels"])
y_train_input['docIdx'] = y_train_input.index + 1
X_test_input = pd.read_csv("20news-bydate/matlab/test.data", delimiter=" ",␣
↪names = ["docIdx", "wordIdx", "freq"],)
y_test_input = pd.read_csv("20news-bydate/matlab/test.label", names=["labels"])
y_test_input['docIdx'] = y_test_input.index + 1
```

[4]:
```
word_cnt = X_train_input[["wordIdx", "freq"]].groupby(["wordIdx"],␣
↪as_index=False).sum().sort_values(by='freq', ascending=False)
word_cnt_filtered = word_cnt[word_cnt['freq']>1000].reset_index()

def preprocessing(X_df, y_df):
    X_df_filtered = X_df.loc[X_df["wordIdx"].isin(word_cnt_filtered.wordIdx)].
↪reset_index(drop=True)

    # combine X_* and y_*
    combined_data = X_df_filtered.merge(y_df, on="docIdx", how="inner")
    X = combined_data
    y = combined_data['labels']
    return X,y
```

[5]:
```
X_train, y_train = preprocessing(X_train_input, y_train_input)
X_test, y_test = preprocessing(X_test_input, y_test_input)
```

[7]:
```
def tfidf_matrix(data, words):
    total_doc = len(data)#.groupby(["wordIdx"]))
    idf = []
    for index, group in data.groupby(["wordIdx"], as_index = False).agg(list).
↪iterrows():
        idf.append( math.log ( total_doc / len(group.docIdx) ) )
    X = []
    y = []
    tfidf = []
    for index, group in data.groupby(["docIdx"], as_index = False).agg(list).
↪iterrows():
        x = [0 for i in range(len(words))]
        y.append(group.labels[0])
        for ind in range(len(words)):
            # print(words[ind])
            if words.wordIdx[ind] in group.wordIdx:
                x[ind] = group.freq[group.wordIdx.index(words.wordIdx[ind])]
            # if word[1].wordIdx in group.wordIdx:
            #     x[words.index(word)] = group.freq[group.wordIdx.index(word[1].
↪wordIdx)] / sum(group.freq)
        X.append(x)

    for i in range(len(X)):
        x = []
        total_words_doc = sum(X[i])
        for j in range(len(X[0])):
            x.append(X[i][j] / total_words_doc * idf[j])
        tfidf.append(x)

    # print(y[0])
    return np.array(tfidf), np.array(y)
```

[8]:
```
X_train_tfidf, y_train_tfidf = tfidf_matrix(X_train, word_cnt_filtered)
X_test_tfidf, y_test_tfidf = tfidf_matrix(X_test, word_cnt_filtered)
```

0.0.1 1

[119]:
```
mnb = MNB()
mnb.fit(X_train_tfidf, y_train_tfidf)
y_pred = mnb.predict(X_test_tfidf)
accuracy_mnb = accuracy_score(y_pred, y_test_tfidf)
print("Accuracy on test data", accuracy_mnb*100)
```

```
Accuracy on test data 40.60250599840043
```

0.0.2 2

[10]:
```
lda = LDA()
lda.fit(X_train_tfidf, y_train_tfidf)
y_pred = lda.predict(X_test_tfidf)
```

[11]:
```
accuracy = accuracy_score(y_pred, y_test_tfidf)
```

[120]:
```
print("Accuracy on test data",lda.score(X_test_tfidf, y_test_tfidf) * 100)
```

```
Accuracy on test data 39.53612370034658
```

0.0.3 3

[46]:
```
phi = X_train_tfidf
phi.shape
```

[46]:
```
(11260, 292)
```

[47]:
```
def orthogonal_iteration(phi, k, terminate=1e-5):
    m,n = phi.shape
    Q = np.random.randn(n, k)
    Q, R = qr(Q)

    while True:
        temp = phi @ Q
        Q_new, R = qr(phi.T @ temp)

        if norm(Q_new - Q) < terminate:
            break

        Q = Q_new

    return Q
```

[48]:
```
theta = orthogonal_iteration(phi, 2)
theta.shape
```

[48]:
```
(292, 2)
```

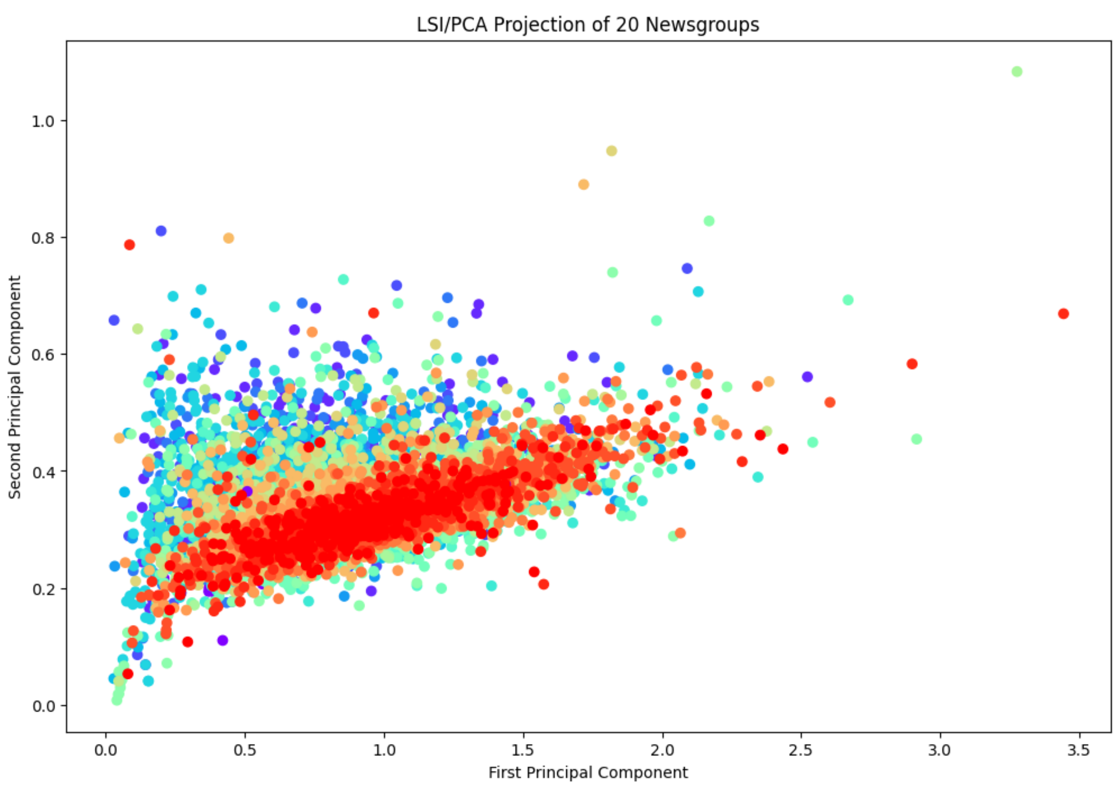
[54]:
```
Y = phi @ theta
```

[55]:
```
# plot Y
Y.shape
```

[55]:
```
(11260, 2)
```

[90]:
```
colors = cm.rainbow(np.linspace(0, 1, 20))
plt.figure(figsize=(12, 8))
```

```python
# for i in range(len(Y)):
#     plt.scatter(Y[i][0], Y[i][1],c=colors[i for j in X_train.
↪groupby(['labels']).agg(set) if i in j])
arr = []
doc_groups = X_train.groupby(['labels']).agg(set)
for i in range(len(Y)):
    flag = 1
    for j,group in doc_groups.iterrows():
        if i+1 in group.docIdx:
            arr.append(j-1)
            flag = 0
            break
    if flag:
        arr.append(0)
plt.scatter(Y[:, 0], Y[:, 1], c=colors[arr])
plt.xlabel('First Principal Component')
plt.ylabel('Second Principal Component')
plt.title('LSI/PCA Projection of 20 Newsgroups')
plt.show()
```



0.0.4 4

[94]:
```
phi = orthogonal_iteration(X_train_tfidf, 100)
```

[95]:
```
gmm = GaussianMixture(n_components=20, covariance_type='full')
```

[96]:
```
gmm.fit(phi)
```

[96]:
```
GaussianMixture(n_components=20)
```

[97]:
```
means = gmm.means_
```

[ ]:
```

```

[98]:
```
Y = phi @ means.T
```

[99]:
```
means.shape
```

[99]:
```
(20, 100)
```

[100]:
```
phi.shape
```

[100]:
```
(292, 100)
```

[101]:
```
Y.shape
```

[101]:
```
(292, 20)
```

[115]:
```
url = "http://qwone.com/~jason/20Newsgroups/vocabulary.txt"
file = urllib.request.urlopen(url)

word_index_map = {}
i = 0
for line in file:
    word_index_map[i] = line.decode("utf-8").strip()
    i = i + 1
```

[117]:
```
for i in range(20):
    temp = Y[:, i]
    ind1 = temp.argsort()[-10:][::-1]
    print("Top 10 words in cluster", i+1, ":")
    for i in ind1:
        print(word_index_map[i], end=" ")
    print()
```

```
Top 10 words in cluster 1 :
laurel organizations atrocities san feet francisco box can netcom foundation
Top 10 words in cluster 2 :
and blueprints hannover area secular south bay berlin this internationaler
```

```
Top 10 words in cluster 3 :
name immoralities letters alternate book critiques claus describe well hrsg
Top 10 words in cluster 4 :
write newsletter horrors jr islington santa dead atrocities contradictions that
Top 10 words in cluster 5 :
archive austin books ny contradictions foote ink critiques miller blueprints
Top 10 words in cluster 6 :
for americans wrote promoting claus leaving und post der that
Top 10 words in cluster 7 :
december similarity well east per books area atrocities nl this
Top 10 words in cluster 8 :
last feet this jr american book well moulded so based
Top 10 words in cluster 9 :
hollywood book pp lines ethical james handbook moulded davy conduit
Top 10 words in cluster 10 :
religion horrors british thought they short living word be edgar
Top 10 words in cluster 11 :
which copying letters com wrote exists informationen many society immoralities
Top 10 words in cluster 12 :
plastic this area king containing immoralities pp square monks characters
Top 10 words in cluster 13 :
bumper who aah monks claus that horrors short north fax
Top 10 words in cluster 14 :
in provoking freethinker one vertrieb absurdities prometheus this books monks
Top 10 words in cluster 15 :
addresses humanism berlin kingdom hannover horrors pp word new figmo
Top 10 words in cluster 16 :
paraphernalia magazine that exists canyon holloway press lists king ethical
Top 10 words in cluster 17 :
dick rationalist that netcom magazine informationen claus islington set national
Top 10 words in cluster 18 :
atheism square is zeit miz so norm netcom informationen provoking
Top 10 words in cluster 19 :
alt cameron bible passage lion is wrote publish pp one
Top 10 words in cluster 20 :
it square provoking various thomas one be pp any ultimate
```

[121]:
```python
ans = """
Most of the words in the top 10 are common words like can, and, it, is
but some words like hollywood in cluster 10 or organisations in cluster 1
show that the cluster might have new articles about hollywood or organisations
respectively.
"""
print(ans)
```

```
Most of the words in the top 10 are common words like can, and, it, is
but some words like hollywood in cluster 10 or organisations in cluster 1
```

show that the cluster might have new articles about hollywood or organisations respectively.

[ ]:

```

```