Problem Set 3

Jiarong Ye

October 17, 2018

Q1

1.(25 pts.) Consider a version of the perceptron learning algorithm in which the learning rate η(t) can vary with each weight change step t (e.g., it might be different at different times of the day or it may be a function of the weather, the learner's mood, the amount of coffee consumed, etc.). Prove from first principles that the resulting variant of the perceptron learning algorithm is guaranteed to terminate with a weight vector that defines a separating hyperplane whenever the training data are linearly separable, so long as $0 < A \leq \eta(t) \leq B$ where A and B are fixed lower and upper bounds respectively.

Proof.

The goal of this proof is to get a upper bound of the number of weight updates (denote as t)

So first define the training phase of the perceptron learning algorithm:
With initial weight as $W_o = [0, 0, ..., 0]^T$

$$\begin{aligned} y_k &= sign(W \cdot X_k) \\ W &\leftarrow W + \eta(d_k - y_k)X_k \end{aligned} \tag{1}$$

Assume $E = (X_k, d_k)$, where $X \in \mathbb{R}$, and $d_k \in \{-1, 1\}$,
denote

$$S^+ = \{X^k | (X^k, d^k) \in E \cap d_k = 1\}$$

$$S^- = \{X^k | (X^k, d^k) \in E \cap d_k = -1\}$$

To prove that the perceptron algorithm is guaranteed to terminate with a weight vector that defines a separating hyperplane with two prerequisites:

1. the training data are linearly separable
2. $0 < A \leq \eta(t) \leq B$ where A and B are fixed lower and upper bounds respectively

Denote the weight vector as W^*
If the training data are linearly separable, then from which we can get that:

$$\forall X_k \in S^+, W^* \cdot X_k \geq \delta$$

$$\forall X_k \in S^-, W^* \cdot X_k \leq -\delta$$

In general, we could define

$$\forall X_k \in S^+, Z_k = X_k$$

$$\forall X_k \in S^-, Z_k = -X_k$$

$$Z = Z_k$$

so we could organize

$$\forall X_k \in S^+, W^* \cdot X_k \geq \delta$$

$$\forall X_k \in S^-, W^* \cdot X_k \leq -\delta$$

as

$$\forall Z_k \in Z, W^* \cdot Z_k \geq \delta$$

also update E as $E = \{Z_k, 1\}$
weight as $W_{t+1} = W_t + \eta(d_k - y_k)Z_k = W_t + 2\eta Z_k$
then we divide the proof process into several steps:

step 1

multiply W^* on both side of the weights updating equation above we have:

$$W^* \cdot W_{t+1} = W^* \cdot (W_t + 2\eta Z_k) = (W^* \cdot W_t) + 2\eta(W^* \cdot Z_k)$$

Since we have already establish that the training data is linearly separable, so $W^* \cdot Z_k \geq \delta$, plug back in we get the following inequality:

$$W^* \cdot W_{t+1} \geq W^* \cdot W_t + 2\eta\delta$$

$$\therefore W^* \cdot W_t \geq 2t\eta\delta$$

since we know:

$$W^* \cdot W_t = \|W^*\| \, \|W_t\| \cos\theta$$

and $\cos\theta \in [-1, 1]$

$$\therefore W^* \cdot W_t \leq \|W^*\| \, \|W_t\|$$

$$\therefore \|W^*\| \, \|W_t\| \geq W^* \cdot W_t \geq 2t\eta\delta$$

step 2

square the weight vector at $t+1$ state:

$$\|W_{t+1}\|^2 = (W_t + 2\eta Z_k) \cdot (W_t + 2\eta Z_k)$$

expand it, we have:

$$\|W_{t+1}\|^2 = (W_t \cdot W_t) + 4\eta(W_t \cdot Z_k) + 4\eta^2(Z_k \cdot Z_k)$$

since the training phase of the algorithm is based on Gradient Descent, the update direction goes in the opposite way as the gradient, thus we should be able to assume that with $W_{t+1} = W_t + 2\eta Z_k$, $W_t \cdot Z_k \leq 0$

so

$$\|W_{t+1}\|^2 = (W_t \cdot W_t) + 4\eta(W_t \cdot Z_k) + 4\eta^2(Z_k \cdot Z_k) \leq \|W_t\|^2 + 4\eta^2 \|Z_k\|^2$$

here introduce the upper bound of Z_k as $L = \max\{Z_k\}$

thus we could bound the norm of the weight at $t+1$ state above as:

$$\|W_{t+1}\|^2 \leq \|W_t\|^2 + 4\eta^2 L^2$$

so

$$\|W_t\|^2 \leq 4t\eta^2 L^2$$

$$\therefore \forall t, \|W_t\| \leq 2\eta L\sqrt{t}$$

step 3

recap the result from step 1 and step 2:

$$\begin{aligned} \|W^*\| \|W_t\| &\geq 2t\eta\delta \\ \|W_t\| &\leq 2\eta L\sqrt{t} \end{aligned} \tag{2}$$

replace $\|W_t\|$ with $2\eta L\sqrt{t}$ in the first inequality (would not change the inequality because $2\eta L\sqrt{t}$ is the upper bound of $\|W_t\|$)

thus we have:

$$\|W^*\| 2\eta L\sqrt{t} \geq 2t\eta\delta$$

here from the 2nd prerequisite we define before the proof process:

$0 < A \leq \eta(t) \leq B$ where A and B are fixed lower and upper bounds respectively

we would be able to cancel out $2\eta\sqrt{t}$ on both side:

$$\|W^*\| L \geq \delta\sqrt{t}, \text{ the inequality is independent from } \eta$$

$$\therefore t \leq \left(\frac{\|W^*\| L}{\delta}\right)^2$$

Thus no matter the learning rate $\eta(t)$ varies with each weight change step t, t would always be upper bounded, which satisfies the goal established at the beginning, hence proving that when

1. the training data are linearly separable
2. $0 < A \leq \eta(t) \leq B$ where A and B are fixed lower and upper bounds respectively

the resulting variant of the perceptron learning algorithm is guaranteed to terminate with a weight vector that defines a separating hyperplane. **QED**

□

Q2

2.(25 pts.) Use the decision tree learning algorithm to construct a decision tree classifier from the following training set and show all your calculations.

Outlook	Temperature	Humidity	Wind	PlayTennis
sunny	hot	high	weak	No
sunny	hot	high	strong	No
overcast	hot	high	weak	Yes
rain	mild	high	weak	Yes
rain	cool	normal	weak	Yes
rain	cool	normal	strong	No
overcast	cool	normal	strong	Yes
sunny	mild	high	weak	No
sunny	cool	normal	weak	Yes
rain	mild	normal	weak	Yes
sunny	mild	normal	strong	Yes
overcast	mild	high	strong	Yes
overcast	hot	normal	weak	Yes
rain	mild	high	strong	No

Root Node

First construct a few tables for each attribute to **entropy** and **info gain** calculation:

Outlook

	sunny	overcast	rain
play: yes	2	4	3
play: no	3	0	2

Temperature

	hot	mild	cool
play: yes	2	4	3
play: no	2	2	1

Humidity

	high	normal
play: yes	3	6
play: no	4	1

Wind

	weak	strong
play: yes	6	3
play: no	2	3

Entropy before split:

$$E = -\frac{9}{14}\log(\frac{9}{14}) - \frac{9}{14}\log(\frac{9}{14}) = 0.940$$

- Entropy after splitting on *Outlook*:

$$E(Outlook_{sunny}) = -\frac{2}{5}\log\frac{2}{5} - \frac{3}{5}\log\frac{2}{5} = 0.971$$

$$E(Outlook_{overcast}) = -\frac{4}{4}\log\frac{4}{4} = 0$$

$$E(Outlook_{rain}) = -\frac{2}{5}\log\frac{2}{5} - \frac{3}{5}\log\frac{2}{5} = 0.971$$

$$\text{Info gain } (Outlook) = E - \frac{5}{14}E(Outlook_{sunny}) - \frac{4}{14}E(Outlook_{overcast}) - -\frac{5}{14}E(Outlook_{rain}) = 0.940 - \frac{5}{14}\cdot 0.971 \cdot 2 = 0.246$$

- Entropy after splitting on *Temperature*:

$$E(Temperature_{hot}) = -\frac{2}{4}\log\frac{2}{4}\cdot 2 = 1$$

$$E(Temperature_{mild}) = -\frac{4}{6}\log\frac{4}{6} - \frac{2}{6}\log\frac{2}{6} = 0.918$$

$$E(Temperature_{cool}) = -\frac{3}{4}\log\frac{3}{4} - \frac{1}{4}\log\frac{1}{4} = 0.811$$

$$\text{Info gain}(Temperature) = E - \frac{4}{14}E(Temperature_{hot}) - \frac{6}{14}E(Temperature_{mild}) - \frac{4}{14}E(Temperature_{cool})$$

$$= 0.940 - \frac{4}{14}\cdot 1 - \frac{6}{14}\cdot 0.918 - \frac{4}{14}\cdot 0.811 = 0.029$$

- Entropy after splitting on *Humidity*:

$$E(Humidity_{high}) = -\frac{3}{7}\log\frac{3}{7} - \frac{4}{7}\log\frac{4}{7} = 0.985$$

$$E(Humidity_{normal}) = -\frac{6}{7}\log\frac{6}{7} - \frac{1}{7}\log\frac{1}{7} = 0.592$$

$$\text{Info gain } (Humidity) = E - \frac{7}{14}E(Humidity_{high}) - \frac{7}{14}E(Humidity_{normal}) = 0.940 - \frac{7}{14}\cdot 0.985 - \frac{7}{14}\cdot 0.592 = 0.152$$

- Entropy after splitting on *Wind*:

$$E(Wind_{weak}) = -\frac{6}{8}\log\frac{6}{8} - \frac{2}{8}\log\frac{2}{8} = 0.811$$

$$E(Wind_{strong}) = -\frac{3}{6}\log\frac{3}{6} \cdot 2 = 1$$

$$\text{Info gain } (Wind) = E - \frac{8}{14}E(Wind_{weak}) - \frac{6}{14}E(Wind_{strong}) = 0.940 - \frac{8}{14} \cdot 0.811 - \frac{6}{14} \cdot 1 = 0.048$$

Since $info\ gain(Outlook) = 0.246 > info\ gain(Humidity) = 0.152 > info\ gain(Wind) = 0.048 > info\ gain(Temperature) = 0.029$, so if split on *Outlook*, the reduction of entropy, i.e. the reduction of uncertainty is larger than splitting on others. Thus **Outlook** should be selected for the root node for the Decision Tree.

First Child

Since $Outlook_{overcast}$ is pure, thus no further split is required. Hence next split should be on $Outlook_{sunny}$ and $Outlook_{rain}$

- Outlook:sunny

Temperature (Outlook:sunny)

	hot	mild	cool
play: yes	0	1	1
play: no	2	1	0

Humidity (Outlook:sunny)

	high	normal
play: yes	0	2
play: no	3	0

Wind (Outlook:sunny)

	weak	strong
play: yes	1	1
play: no	2	1

Entropy before split:

$$E = -\frac{2}{5}\log(\frac{2}{5}) - \frac{3}{5}\log(\frac{3}{5}) = 0.971$$

- Entropy after splitting on *Temperature*:

$$E(Temperature_{hot}) = 0$$

$$E(Temperature_{mild}) = 1 = 0.918$$

$$E(Temperature_{cool}) = 0$$

$$\text{Info gain}(Temperature) = E - \frac{2}{5}E(Temperature_{hot}) - \frac{2}{5}E(Temperature_{mild}) - \frac{1}{5}E(Temperature_{cool}) = 0.971 - 0.4 = 0.571$$

- Entropy after splitting on *Humidity*:

$$E(Humidity_{high}) = 0$$

$$E(Humidity_{normal}) = 0$$

$$\text{Info gain }(Humidity) = E = 0.971$$

- Entropy after splitting on *Wind*:

$$E(Wind_{weak}) = -\frac{1}{3}\log\frac{1}{3} - \frac{2}{3}\log\frac{2}{3} = 0.918$$

$$E(Wind_{strong}) = 1$$

$$\text{Info gain }(Wind) = E - \frac{3}{5}E(Wind_{weak}) - \frac{2}{5}E(Wind_{strong}) = 0.971 - 0.6 \cdot 0.918 - 0.4 \cdot 1 = 0.0202$$

Since $info\,gain(Humidity) = 0.971 > info\,gain(Temperature) = 0.571 > info\,gain(Wind) = 0.0201$, so if split on *Humidity*, the reduction of entropy, i.e. the reduction of uncertainty is larger than splitting on others. Thus **Humidity** should be selected for the next node for the Decision Tree.

- Outlook:rain

Temperature (Outlook:rain)

	mild	cool
play: yes	2	1
play: no	1	1

Humidity (Outlook:rain)

	high	normal
play: yes	1	2
play: no	1	1

Wind (Outlook:rain)

	weak	strong
play: yes	3	2
play: no	0	0

Entropy before split:

$$E = -\frac{2}{5}\log(\frac{2}{5}) - \frac{3}{5}\log(\frac{3}{5}) = 0.971$$

- Entropy after splitting on *Temperature*:

$$E(Temperature_{mild}) = -\frac{1}{3}\log\frac{1}{3} - \frac{2}{3}\log\frac{2}{3} = 0.918$$

$$E(Temperature_{cool}) = 1$$

$$\text{Info gain}(Temperature) = E - \frac{3}{5}E(Temperature_{mild}) - \frac{2}{5}E(Temperature_{cool}) = 0.971 - 0.6 \cdot 0.918 - 0.4 \cdot 1 = 0.0202$$

- Entropy after splitting on *Humidity*:

$$E(Humidity_{high}) = 1$$

$$E(Humidity_{normal}) = -\frac{1}{3}\log\frac{1}{3} - \frac{2}{3}\log\frac{2}{3} = 0.918$$

$$\text{Info gain }(Humidity) = E - \frac{2}{5}E(Humidity_{high})) - \frac{3}{5}E(Humidity_{normal}) = 0.971 - 0.6 \cdot 0.918 - 0.4 \cdot 1 = 0.0202$$

- Entropy after splitting on *Wind*:

$$E(Wind_{weak}) = 0$$

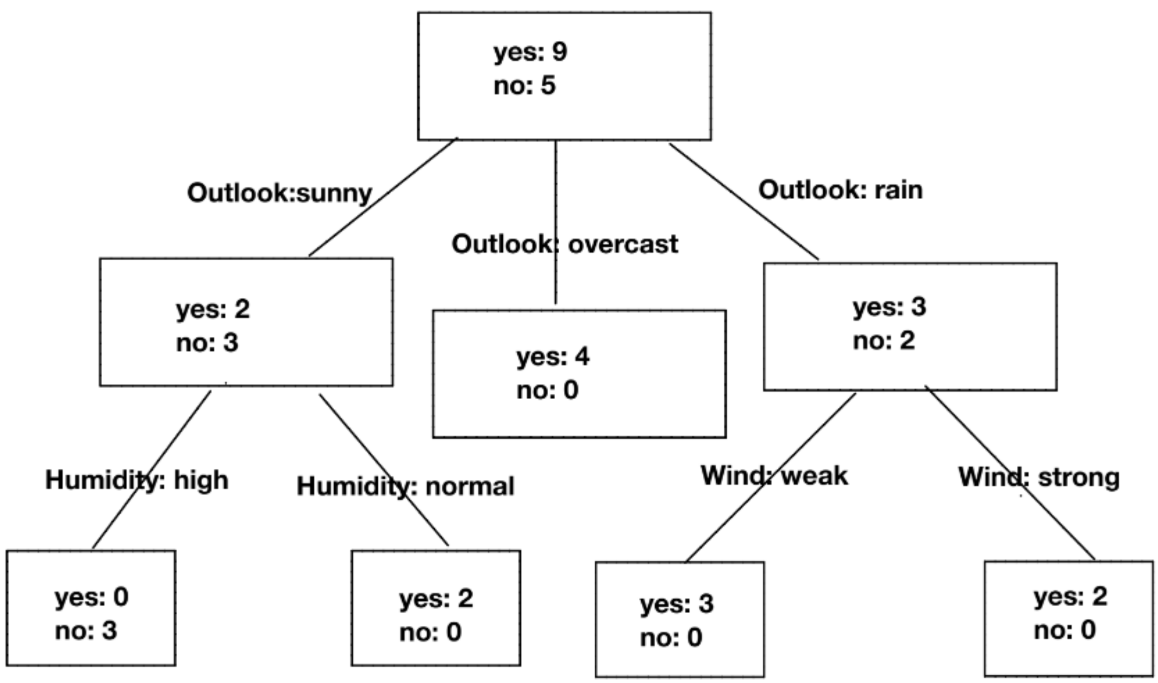
$$E(Wind_{strong}) = 0$$

$$\text{Info gain } (Wind) = E = 0.971$$

Since $info\,gain(Wind) = 0.971 > info\,gain(Temperature) = 0.0202 = info\,gain(Humidity) = 0.0202$, so if split on *Wind*, the reduction of entropy, i.e. the reduction of uncertainty is larger than splitting on others. Thus **Wind** should be selected for the next node for the Decision Tree.

Till this point, there's no impure attribute that needed to be splitted. Therefore, the Decision Tree construction is completed.

Final Tree:



Q3

3.(25 pts.) Recall the use of bag of words representation for text classification. Note that in a bag of words representation each document is encoded using a bag of words. Consider a more general scenario in which it might be beneficial to encode a data sample to be classified using multiple bags of features that correspond to perhaps different modality (e.g., textual features or words, visual features, etc. in the case of a document that contains both words and pictures). We can think of each modality as being associated with its own vocabulary (e.g., a set of words, a set of visual features, etc.). Precisely formulate the problem of learning classifiers in a setting wherein each data sample is represented using multiple (say M) bags of features, one for each modality. Outline a learning algorithm (at sufficient detail needed for implementation). Hint: You may consider variations or adaptations of the learning algorithms that you have studied so far

Answer:

This should probably considered as a MultiModality Learning problem, in which different format of data sources, like images, texts, videos, audios are used together to extract features from each and concatenate with data fusion techniques for a machine learning task, examples I know are mostly in the field of computer vision, such as image captioning, context encoder for task like image inpainting, semantic image segmentation, etc. In this case, the task is text classification, and major data formats include text and images. Thus a possible algorithm pipeline could be constructed as follows:

- Data Preprocessing
  - Feature Engineering
    - text
      - BOW
      - TF-IDF
      - Word2Vec
    - image
      - BOVW (Bag of Visual Words, inspired by BOW)
        eg: SIFT/ORB/SURF
  - Feature Fusion
- Classifier
  - Naive Bayes
  - Decision Tree
  - Logistic Regression
  - ...

Example (because I could not find a dataset for document classification task that includes both text and images ready to train, so the code is a just general idea of what I think how the algorithm probably works):

```
In [11]: from sklearn.datasets import fetch_20newsgroups # just a random example
         from sklearn.feature_extraction.text import CountVectorizer, TfidfVectorizer
         from sklearn.naive_bayes import GaussianNB
         from sklearn.tree import DecisionTreeClassifier
         from sklearn.ensemble import RandomForestClassifier,
         from sklearn.linear_model import LogisticRegression
         from sklearn.neural_network import MLPClassifier
         from sklearn.preprocessing import StandardScaler
         from scipy.cluster import vq
         from sklearn.cluster import KMeans
         import matplotlib.pyplot as plt
         import cv2
         import glob
```

```
import pandas as pd
import numpy as np
import nltk
import gensim


# just a random example (the image dataset and text dataset have no relations)
IMAGENET_CAR = '/home/karen/Downloads/data/ImageNet_Utils/imageNet_dataset/train/car'


class MultuModalityPipeline:

    def __init__(self):
        self.text_dataset = fetch_20newsgroups(subset='train', shuffle=True)
        # pick the top 20 as an example to see what the result will be like
        self.text_X = self.text_dataset.data[:20]
        self.text_y = self.text_dataset.target[:20]
        self.image_dataset = glob.glob("%s/*" %IMAGENET_CAR)[:20]

    def create_word2vec_matrix(self, text, word2vec):
        word2vec_matrix=[]
        for idx, line in enumerate(text):
            regex = nltk.tokenize.RegexpTokenizer(r'[a-zA-Z]+')
            word_lst = ' '.join(regex.tokenize(line))
            current_word2vec=[]
            for word in word_lst:
                if word in word2vec.vocab:
                    vec = word2vec.vectors_norm[word2vec.vocab[word].index]
                    if vec is not None:
                        current_word2vec.append(vec)
            if np.array(current_word2vec).shape[0]!=0:
                sentence_word2vec = list(np.array(current_word2vec).mean(axis=0))
                word2vec_matrix.append(sentence_word2vec)
            current_word2vec=[]
        return word2vec_matrix


    def word_feature_vectorizer(self, option = 'BOW'):
        if option != 'Word2Vec':
            if option == 'BOW':
                # n-gram = 1 ~ 2
                vect = CountVectorizer(token_pattern='[a-zA-Z]+', ngram_range=([1,2]),
                    preprocessor=None, stop_words='english', max_features=3000)
            elif option == 'TFIDF':
                vect = TfidfVectorizer(token_pattern='[a-zA-Z]+', analyzer = 'word',
                sublinear_tf=True, min_df=10, norm='l1', encoding='latin-1',
                                            ngram_range=(1,2), stop_words='english')
```

```python
            X_vect = vect.fit_transform(self.text_X)
            return X_vect, vect.get_feature_names()
        else:
            wv = gensim.models.KeyedVectors.load_word2vec_format(fname=
            '/home/karen/Downloads/data/GoogleNews-vectors-negative300.bin.gz',
                                                    binary=True)
            wv.init_sims(replace=True)
            wv_matrix = self.create_word2vec_matrix(text=self.text_X, word2vec=wv)
            return wv_matrix


    def img_feature_vectorizer(self, num_features, option = 'ORB'):
        dataset_size = len(self.image_dataset)
        resp = np.zeros((dataset_size, 1))
        des_list = []
        for idx, f in enumerate(self.image_dataset):
            img = cv2.imread(f)
            img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
            if option == 'ORB':
                descriptor = cv2.ORB_create()
            elif option == 'SIFT':
                descriptor = cv2.xfeatures2d.SIFT_create()
            elif option == 'SURF':
                descriptor = cv2.xfeatures2d.SURF_create()

            (_, features) = descriptor.detectAndCompute(img, None)
            des = np.float32(features)
            des_list.append((f, des))

        descriptors = des_list[0][1]
        for image_path, descriptor in des_list[1:]:
            descriptors = np.vstack((descriptors, descriptor))

        criteria = (cv2.TERM_CRITERIA_EPS + cv2.TERM_CRITERIA_MAX_ITER, 10, 1.0)
        flags = cv2.KMEANS_RANDOM_CENTERS
        print(descriptors.shape)
        _, _, centers = cv2.kmeans(descriptors, num_features, None, criteria, 10, flags)
        im_features = np.zeros((dataset_size, num_features), "float32")
        for i in range(dataset_size):
            words, distance = vq.vq(des_list[i][1], centers)
            for w in words:
                im_features[i][w] += 1

        # Scaling the values of features
        im_features = StandardScaler().fit(im_features).transform(im_features)

        return im_features
```

```
    def classifier(self, option='NB'):
        if option == 'NB':
            clf = GaussianNB()
        elif option == 'DT':
            clf = DecisionTreeClassifier(criterion = 'entropy',
                                         random_state = 100,
                                         max_depth = 5,
                                         min_samples_leaf = 2)
        elif option == 'RF':
            clf = RandomForestClassifier(random_state=100,
                                       n_estimators=20,
                                       criterion='entropy',
                                       n_jobs=4)
        elif option == 'LR':
            clf = LogisticRegression()

        return clf

    def evaluate(self):
        return NotImplemented
```

```
In [12]: mp = MultuModalityPipeline()
```

Modality 1: Text; Feature 1: BOW

```
In [14]: X_vect, text_feature_dict = mp.word_feature_vectorizer()
         bow_feature_df = pd.DataFrame(X_vect.todense(), columns=text_feature_dict)
         bow_feature_df
```

Out[14]:

	aa	aa insane	aaron	aaron family	ability	ability handle	ablility	\
0	0	0	1	1	0	0	0	
1	0	0	0	0	0	0	0	
2	0	0	0	0	0	0	0	
3	0	0	0	0	0	0	0	
4	0	0	0	0	0	0	0	
5	0	0	0	0	0	0	0	
6	0	0	0	0	0	0	0	
7	1	1	0	0	0	0	0	
8	0	0	0	0	0	0	0	
9	0	0	0	0	0	0	0	
10	0	0	0	0	0	0	0	
11	0	0	0	0	0	0	0	
12	0	0	0	0	0	0	0	
13	0	0	0	0	0	0	0	
14	0	0	0	0	0	0	0	
15	0	0	0	0	1	1	1	
16	0	0	0	0	0	0	0	

```
17   0          0      0             0        0               0         0
18   0          0      0             0        0               0         0
19   0          0      0             0        0               0         0

    ablility handle  ac  ac adaptor      ...        z fidonet  z speed  \
0                 0   0           0      ...                0        0
1                 0   0           0      ...                0        0
2                 0   0           0      ...                0        0
3                 0   0           0      ...                0        0
4                 0   0           0      ...                0        0
5                 0   0           0      ...                1        0
6                 0   0           0      ...                0        0
7                 0   0           0      ...                0        0
8                 0   0           0      ...                0        0
9                 0   0           0      ...                0        0
10                0   1           1      ...                0        0
11                0   0           0      ...                0        0
12                0   0           0      ...                0        0
13                0   1           0      ...                0        0
14                0   0           0      ...                0        0
15                1   0           0      ...                0        1
16                0   0           0      ...                0        0
17                0   0           0      ...                0        0
18                0   0           0      ...                0        0
19                0   0           0      ...                0        0

    zabriskie  zabriskie berkeley  zeppelin  zeppelin convex  zhenghao  \
0           0                   0         0                0         0
1           0                   0         0                0         0
2           0                   0         0                0         0
3           1                   1         0                0         0
4           0                   0         0                0         0
5           0                   0         0                0         0
6           0                   0         0                0         1
7           0                   0         0                0         0
8           0                   0         0                0         0
9           0                   0         0                0         0
10          0                   0         0                0         0
11          0                   0         0                0         0
12          0                   0         0                0         0
13          0                   0         0                0         0
14          0                   0         0                0         0
15          0                   0         0                0         0
16          0                   0         0                0         0
17          0                   0         1                1         0
18          0                   0         0                0         0
19          0                   0         0                0         0
```

```
    zhenghao yeh  zion  zion berkeley
0              0     0              0
1              0     0              0
2              0     0              0
3              0     1              1
4              0     0              0
5              0     0              0
6              1     0              0
7              0     0              0
8              0     0              0
9              0     0              0
10             0     0              0
11             0     0              0
12             0     0              0
13             0     0              0
14             0     0              0
15             0     0              0
16             0     0              0
17             0     0              0
18             0     0              0
19             0     0              0

[20 rows x 3000 columns]
```

Modality 1: Text; Feature 2: TF-IDF

In [26]:
```python
X_vect, text_feature_dict = mp.word_feature_vectorizer(option='TFIDF')
tfidf_feature_df = pd.DataFrame(X_vect.todense(), columns=text_feature_dict)
tfidf_feature_df
```

Out[26]:
```
         edu      host     lines      nntp  nntp posting  organization  \
0   0.264696  0.000000  0.142063  0.000000      0.000000      0.135454
1   0.204917  0.125176  0.000000  0.125176      0.125176      0.084603
2   0.148184  0.000000  0.056429  0.000000      0.000000      0.053804
3   0.127174  0.096289  0.068254  0.096289      0.096289      0.065079
4   0.157547  0.096239  0.068219  0.096239      0.096239      0.065045
5   0.230260  0.000000  0.087684  0.000000      0.000000      0.083605
6   0.114952  0.087036  0.061695  0.087036      0.087036      0.058825
7   0.124723  0.076188  0.054006  0.076188      0.076188      0.051493
8   0.139254  0.085065  0.060298  0.085065      0.085065      0.057493
9   0.000000  0.000000  0.147823  0.000000      0.000000      0.140946
10  0.144351  0.088178  0.062505  0.088178      0.088178      0.059597
11  0.149989  0.000000  0.080499  0.000000      0.000000      0.076754
12  0.123566  0.060705  0.043031  0.060705      0.060705      0.041029
13  0.348482  0.000000  0.150895  0.000000      0.000000      0.143875
14  0.121685  0.092133  0.065308  0.092133      0.092133      0.062270
15  0.152846  0.000000  0.082032  0.000000      0.000000      0.078216
16  0.142570  0.087090  0.061733  0.087090      0.087090      0.058862
```

17	0.108524	0.082169	0.058245	0.082169	0.082169	0.055535
18	0.000000	0.000000	0.066704	0.000000	0.000000	0.063601
19	0.000000	0.124060	0.087940	0.124060	0.124060	0.083849

	posting	posting host	s	subject	t	university	writes
0	0.000000	0.000000	0.000000	0.135454	0.000000	0.000000	0.322334
1	0.125176	0.125176	0.000000	0.084603	0.000000	0.000000	0.000000
2	0.000000	0.000000	0.258407	0.053804	0.218967	0.134786	0.075620
3	0.096289	0.096289	0.000000	0.065079	0.101499	0.000000	0.091467
4	0.162947	0.096239	0.000000	0.065045	0.000000	0.096239	0.000000
5	0.000000	0.000000	0.273642	0.083605	0.000000	0.123700	0.117504
6	0.087036	0.087036	0.091744	0.058825	0.091744	0.087036	0.000000
7	0.076188	0.076188	0.135976	0.051493	0.000000	0.128997	0.072372
8	0.085065	0.085065	0.089667	0.057493	0.089667	0.000000	0.080804
9	0.000000	0.000000	0.000000	0.140946	0.372190	0.000000	0.198095
10	0.088178	0.088178	0.000000	0.059597	0.000000	0.149299	0.083761
11	0.000000	0.000000	0.251219	0.076754	0.251219	0.113564	0.000000
12	0.060705	0.060705	0.108343	0.069468	0.152697	0.060705	0.097635
13	0.000000	0.000000	0.000000	0.143875	0.000000	0.212874	0.000000
14	0.092133	0.092133	0.097118	0.105433	0.000000	0.000000	0.087519
15	0.000000	0.000000	0.256003	0.078216	0.121987	0.000000	0.230700
16	0.087090	0.087090	0.000000	0.058862	0.155433	0.087090	0.000000
17	0.082169	0.082169	0.086614	0.055535	0.146650	0.000000	0.078053
18	0.000000	0.000000	0.317143	0.063601	0.305460	0.094102	0.089389
19	0.124060	0.124060	0.000000	0.083849	0.000000	0.124060	0.000000

Modality 1: Text; Feature 3: Word2Vec

In [16]:
```
X_vect = mp.word_feature_vectorizer(option='Word2Vec')
wv_feature_df = pd.DataFrame(X_vect)
wv_feature_df
```

Out[16]:

	0	1	2	3	4	5	6 \
0	-0.067881	0.045429	-0.001874	0.050565	-0.022454	0.007563	-0.035084
1	-0.063458	0.041883	-0.002374	0.050936	-0.020866	0.005868	-0.041116
2	-0.070741	0.044540	-0.003403	0.051431	-0.022400	0.006889	-0.034586
3	-0.068900	0.042798	-0.001216	0.050963	-0.023288	0.006770	-0.035063
4	-0.068026	0.044948	0.000409	0.047730	-0.022130	0.008037	-0.038194
5	-0.064843	0.040551	0.000955	0.051553	-0.018312	0.005935	-0.041921
6	-0.066959	0.047140	-0.002836	0.049983	-0.018009	0.007732	-0.034249
7	-0.066149	0.040620	0.002770	0.050021	-0.020405	0.003795	-0.038243
8	-0.067165	0.042753	-0.004081	0.052972	-0.021730	0.006952	-0.035407
9	-0.068744	0.042236	0.002584	0.048645	-0.022703	0.002229	-0.040206
10	-0.066204	0.045692	0.000309	0.045926	-0.025670	0.006050	-0.038649
11	-0.066014	0.041827	0.003820	0.048460	-0.023610	0.002467	-0.040933
12	-0.064196	0.044367	-0.001830	0.048302	-0.024106	0.005512	-0.035001
13	-0.068239	0.045414	-0.000189	0.048976	-0.024862	0.004375	-0.041593
14	-0.064602	0.038192	0.004091	0.050010	-0.027116	0.002943	-0.037517

15	-0.066727	0.043360	-0.000864	0.049920	-0.019202	0.008397	-0.037460
16	-0.065714	0.042904	0.003364	0.051934	-0.018441	0.005104	-0.038052
17	-0.063816	0.040570	0.001282	0.047624	-0.024845	0.002346	-0.039064
18	-0.065941	0.043535	-0.001149	0.049354	-0.022199	0.008688	-0.036506
19	-0.062873	0.038084	0.006111	0.047440	-0.023180	-0.000576	-0.031899

	7	8	9	...	290	291	292 \
0	-0.021060	-0.014811	0.008261	...	0.026471	-0.001058	-0.037982
1	-0.021798	-0.016755	0.010505	...	0.024618	0.002757	-0.038037
2	-0.020604	-0.016119	0.009814	...	0.027248	0.004349	-0.034061
3	-0.022992	-0.012264	0.007841	...	0.025188	0.004164	-0.036158
4	-0.024864	-0.020690	0.009575	...	0.026218	0.001301	-0.038024
5	-0.025152	-0.015071	0.009894	...	0.025125	0.005245	-0.033732
6	-0.016489	-0.018881	0.010751	...	0.028442	0.001760	-0.040036
7	-0.021766	-0.018202	0.014641	...	0.029046	0.004711	-0.032712
8	-0.017782	-0.017423	0.010191	...	0.026903	0.001145	-0.034262
9	-0.027820	-0.015648	0.009217	...	0.025240	0.010071	-0.036973
10	-0.024547	-0.021882	0.010576	...	0.026863	0.002797	-0.037480
11	-0.026104	-0.016326	0.010259	...	0.025233	0.002316	-0.034669
12	-0.022446	-0.015185	0.009929	...	0.023964	0.001423	-0.035920
13	-0.026228	-0.017892	0.010589	...	0.025282	0.003004	-0.036937
14	-0.028468	-0.019412	0.011972	...	0.025719	0.006645	-0.034013
15	-0.022444	-0.017726	0.011060	...	0.027127	0.004633	-0.036959
16	-0.021622	-0.013823	0.010142	...	0.027467	0.002993	-0.035956
17	-0.022667	-0.016617	0.009747	...	0.026705	0.000457	-0.033467
18	-0.020673	-0.020272	0.011709	...	0.028083	0.004155	-0.033117
19	-0.027503	-0.021039	0.007145	...	0.025136	0.002955	-0.030935

	293	294	295	296	297	298	299
0	0.033718	-0.010165	-0.062011	-0.041459	-0.007137	-0.040725	0.056349
1	0.032780	-0.010477	-0.061677	-0.045496	-0.005746	-0.040618	0.050063
2	0.034251	-0.012408	-0.062688	-0.042718	-0.010747	-0.041362	0.057279
3	0.030534	-0.011656	-0.061273	-0.038613	-0.006891	-0.038747	0.056370
4	0.032200	-0.012778	-0.061774	-0.042119	-0.007339	-0.041163	0.060847
5	0.030699	-0.013031	-0.059054	-0.043476	-0.002783	-0.042552	0.053156
6	0.032523	-0.013378	-0.064923	-0.041276	-0.008187	-0.041436	0.057905
7	0.031138	-0.014588	-0.060625	-0.044377	-0.004594	-0.043658	0.052576
8	0.034708	-0.011263	-0.061471	-0.042379	-0.007524	-0.041796	0.054347
9	0.031803	-0.011111	-0.061931	-0.046313	-0.003641	-0.039562	0.055385
10	0.034381	-0.013811	-0.063596	-0.045905	-0.007685	-0.041709	0.056482
11	0.033251	-0.013138	-0.062028	-0.042660	-0.004591	-0.036896	0.054570
12	0.032694	-0.012121	-0.062201	-0.041078	-0.005902	-0.038998	0.054119
13	0.034426	-0.012254	-0.062353	-0.045223	-0.006469	-0.038880	0.055237
14	0.033189	-0.016299	-0.062951	-0.044412	-0.006223	-0.039079	0.054843
15	0.032797	-0.013859	-0.062631	-0.042387	-0.007902	-0.043068	0.054968
16	0.033559	-0.014305	-0.062611	-0.042249	-0.004880	-0.041806	0.054143
17	0.034944	-0.013156	-0.059124	-0.044696	-0.003734	-0.036333	0.055879
18	0.034657	-0.013079	-0.062221	-0.043816	-0.008999	-0.045358	0.055255

```
19  0.033070 -0.018818 -0.062221 -0.041482 -0.003093 -0.036871  0.061221

[20 rows x 300 columns]
```

Modality 2: Image; Feature 1: ORB

In [18]:
```python
im_features = mp.img_feature_vectorizer(100)
orb_features_df = pd.DataFrame(im_features)
orb_features_df
```

```
(9971, 32)
```

Out[18]:
```
           0         1         2         3         4         5         6  \
0   0.276563 -1.108874 -1.319950  0.770594 -0.813383  0.662637  0.506502
1   0.276563  0.431229  0.913812 -0.177830  1.272215 -0.261973 -0.578860
2  -1.259899 -0.492833 -0.913812 -1.126253 -1.647622 -1.186583 -1.302434
3  -0.030729 -0.492833  0.913812 -0.889147  0.437975 -1.186583  3.039013
4   0.583856  2.279352 -1.116881  2.193230  0.020856 -0.261973 -0.940647
5  -0.645314 -1.108874  0.507673 -0.652041 -1.647622 -0.878380 -0.217072
6  -1.567192 -0.800853  1.116881 -1.126253  0.855095  1.895451 -0.578860
7  -0.645314  0.123208 -0.101535  1.007700 -0.396264 -0.878380 -1.302434
8  -0.645314  1.663311 -0.304604  1.007700  1.272215 -0.261973  0.144715
9  -0.030729 -0.492833 -0.304604 -1.126253  1.272215  1.279044  1.591864
10 -0.952607 -1.416895  0.913812 -0.889147  0.437975 -0.261973  0.868290
11 -0.952607 -0.492833 -0.507673  1.481912  1.272215 -0.261973 -0.217072
12  0.276563  1.355290  0.101535  0.533489  0.855095 -0.878380 -0.578860
13  0.891148  0.739249 -1.319950 -0.889147 -0.396264  0.046231  0.144715
14  1.198441  1.355290 -1.319950  0.533489 -0.396264  1.895451 -0.940647
15  0.583856  0.123208  0.304604  0.059276 -0.813383  1.279044 -0.578860
16 -0.030729 -1.108874  0.507673 -1.363359 -0.813383  0.970841  0.144715
17  1.505733  0.123208 -0.304604 -0.652041  1.272215 -0.570176  0.506502
18 -1.259899 -0.492833  2.741435  0.533489 -0.813383 -1.494786 -0.217072
19  2.427611 -0.184812 -0.507673  0.770594 -1.230503  0.354434  0.506502

           7         8         9    ...          90        91        92  \
0  -0.066140 -0.332694 -0.916949    ...    1.644815  0.588054 -1.145197
1  -0.595257  0.450116 -1.269622    ...    1.130810 -0.550115  1.431496
2  -1.124374 -0.332694 -0.564277    ...    0.359803  2.864393  0.000000
3   2.314888 -1.115505 -1.269622    ...   -1.182211 -1.688285 -1.145197
4  -0.595257 -1.506910 -0.211604    ...    2.415822  0.208664  2.290393
5   2.314888 -0.332694 -0.211604    ...    0.359803  0.208664 -1.145197
6   0.727536  0.058711  0.493742    ...   -0.668206 -1.308895  0.572598
7  -0.330698  0.841521  1.199088    ...   -0.154201  0.208664 -1.145197
8  -0.066140  2.407142  0.493742    ...    0.102801 -0.929505  0.572598
9  -0.595257 -0.332694 -0.916949    ...   -0.925208 -1.308895  0.286299
10  0.198419 -0.332694 -0.564277    ...   -1.439213 -0.170725  0.000000
11 -0.859815 -1.506910  2.609779    ...   -0.668206 -0.170725  0.286299
```

```
12 -0.859815 -0.332694 -0.564277    ...    -0.668206  0.967444 -0.858898
13 -1.388933 -0.724100  0.493742    ...    -0.411204  0.588054 -0.286299
14 -0.595257  0.841521 -0.916949    ...    -0.411204 -0.929505  0.572598
15 -0.859815  0.450116  0.141069    ...     0.359803  0.588054  0.286299
16  0.727536 -0.332694 -0.564277    ...     1.130810  0.208664 -1.431496
17  0.198419 -0.724100  1.551760    ...     0.873808  0.967444 -0.286299
18  0.992095  2.015737 -0.211604    ...    -0.925208 -0.550115 -0.572598
19  0.462978  0.841521  1.199088    ...    -0.925208  0.208664  1.717795

          93        94        95        96        97        98        99
0   0.918308 -0.040193 -0.690528 -0.943180 -1.427659 -0.197598  0.231593
1   1.611371 -0.040193  0.091202 -0.173237 -1.075151  1.239476  2.084336
2  -0.121286 -0.040193 -1.211681 -0.173237 -0.017625 -0.916135 -0.077198
3   2.304434 -1.245995 -0.169375  1.751620  1.039900 -0.916135 -0.694779
4   0.571777 -0.844061 -0.429951  0.981677 -0.722642  0.880208 -0.385988
5  -0.121286  0.361741 -0.429951  0.981677  1.039900 -1.275403 -0.694779
6  -0.814349  0.763674  0.612355 -0.558209  1.744917 -0.197598  2.084336
7   0.571777 -1.245995  0.872932 -0.558209 -0.722642 -0.197598  2.084336
8  -0.814349  1.165609  0.872932  2.136592 -0.017625  1.239476 -0.694779
9  -0.121286 -1.245995 -0.169375 -1.328152  0.687391 -0.556866  0.540383
10 -1.160880 -0.040193 -0.690528 -0.173237 -0.722642 -0.556866 -0.077198
11 -0.814349  1.969476  0.351778 -0.943180 -0.722642  0.161671 -0.385988
12 -1.507412 -1.245995  0.872932 -0.943180 -0.370134 -0.916135 -1.003569
13 -1.160880  0.361741 -1.211681 -1.328152 -0.370134 -0.556866 -1.003569
14 -1.160880 -0.442127 -1.211681  0.596706 -0.722642  1.958014 -1.003569
15  0.571777  0.361741 -0.429951  0.211734 -1.427659 -0.916135 -1.003569
16  0.225245  1.567542 -0.169375  0.981677  1.392408 -0.916135 -0.694779
17  0.225245 -0.442127  0.612355  0.211734  1.039900  1.598745  0.540383
18  1.264840  1.567542  3.218122 -1.328152  1.744917 -0.556866  0.231593
19 -0.467817 -1.245995 -0.690528  0.596706 -0.370134  1.598745 -0.077198

[20 rows x 100 columns]
```

Modality 2: Image; Feature 2: SIFT

In [7]:
```
im_features = mp.img_feature_vectorizer(100, option='SIFT')
sift_features_df = pd.DataFrame(im_features)
sift_features_df
```

```
(33654, 128)
```

Out[7]:
```
          0         1         2         3         4         5         6  \
0  -0.399969 -0.363566 -0.297945 -0.177570 -0.112933 -0.247042 -0.464891
1  -0.190195 -0.468947 -0.200789 -0.288551 -0.243239  0.007640 -0.228506
2  -0.371999 -0.363566 -0.362716 -0.621494 -0.330110 -0.450788 -0.267903
3   0.257323 -0.047422 -0.297945 -0.122079 -0.199804 -0.297978 -0.228506
4  -0.218165 -0.363566 -0.362716 -0.233060 -0.243239 -0.145169 -0.386096
```

5	-0.092301	0.163341	0.090679	-0.288551	-0.069497	-0.145169	-0.110313
6	-0.427939	-0.468947	-0.136019	-0.510513	-0.330110	-0.450788	-0.386096
7	-0.371999	-0.047422	-0.265560	-0.122079	-0.243239	-0.399851	-0.149711
8	-0.302075	-0.363566	-0.265560	-0.177570	-0.373546	-0.043296	-0.386096
9	4.298969	4.273218	4.333162	4.261671	4.317497	4.286303	4.262817
10	-0.260120	-0.468947	-0.233175	-0.455022	-0.330110	-0.094232	-0.386096
11	-0.106285	-0.363566	-0.136019	-0.011098	-0.330110	-0.348915	0.204867
12	0.075519	0.163341	-0.168404	0.266354	0.234552	-0.196105	-0.267903
13	-0.246135	-0.152803	-0.330331	0.044392	-0.199804	0.109513	0.126072
14	-0.316060	-0.363566	-0.330331	-0.344041	-0.243239	-0.348915	-0.425494
15	-0.399969	-0.258185	-0.200789	-0.233060	-0.243239	-0.297978	0.204867
16	-0.218165	-0.047422	-0.038862	-0.011098	-0.112933	-0.399851	-0.425494
17	-0.120270	0.057960	-0.233175	-0.066589	-0.416982	0.160450	-0.110313
18	-0.358014	-0.258185	-0.265560	-0.510513	-0.199804	-0.196105	-0.070916
19	-0.232150	-0.258185	-0.297945	-0.399532	-0.330110	-0.501724	-0.504289

	7	8	9	...	90	91	92	\
0	0.190213	-0.314225	-0.402597	...	-0.432400	-0.217230	-0.256622	
1	-0.388695	-0.314225	-0.315549	...	-0.211286	-0.247401	-0.256622	
2	-0.554098	-0.479942	-0.359073	...	-0.334127	-0.337914	-0.256622	
3	-0.223293	-0.148508	-0.097929	...	-0.186718	-0.277572	-0.256622	
4	-0.388695	-0.403457	-0.402597	...	-0.309559	-0.337914	-0.256622	
5	-0.223293	0.042704	-0.184977	...	-0.260423	-0.217230	-0.159171	
6	-0.554098	-0.416205	-0.315549	...	-0.334127	-0.428427	-0.224138	
7	-0.223293	-0.314225	-0.184977	...	-0.186718	0.024137	-0.224138	
8	-0.057891	-0.250488	-0.010881	...	-0.235854	-0.187060	-0.256622	
9	4.159867	4.223872	4.298000	...	4.309257	4.338574	4.356078	
10	-0.554098	-0.377962	-0.489646	...	-0.358695	-0.307743	-0.191654	
11	-0.305994	-0.161255	-0.141453	...	-0.113014	-0.096547	-0.256622	
12	0.438316	0.221169	0.163215	...	-0.088445	-0.217230	-0.191654	
13	0.438316	-0.352467	0.032643	...	0.255509	-0.217230	-0.256622	
14	-0.554098	-0.326972	-0.402597	...	-0.432400	-0.187060	-0.256622	
15	-0.305994	-0.467195	-0.359073	...	-0.186718	-0.096547	-0.224138	
16	0.024810	-0.365215	-0.359073	...	-0.334127	-0.217230	-0.159171	
17	-0.057891	0.272158	-0.054405	...	-0.137582	-0.187060	-0.159171	
18	-0.305994	-0.441700	-0.228501	...	-0.088445	-0.307743	-0.256622	
19	-0.554098	0.374138	-0.184977	...	-0.334127	-0.277572	-0.256622	

	93	94	95	96	97	98	99
0	-0.591313	-0.237764	-0.307027	-0.355158	-0.319384	-0.368900	-0.219752
1	-0.326546	-0.211198	-0.266892	-0.237263	-0.151287	-0.344224	-0.219752
2	-0.503058	-0.237764	-0.266892	-0.384632	-0.445457	-0.319549	-0.242642
3	-0.370674	-0.237764	-0.266892	0.028000	-0.277360	-0.245522	-0.242642
4	-0.458930	-0.237764	-0.307027	-0.207790	-0.067239	-0.294873	-0.219752
5	0.026477	-0.211198	-0.266892	-0.266737	-0.277360	-0.245522	-0.219752
6	-0.326546	-0.237764	-0.186624	-0.325684	-0.361408	-0.319549	-0.242642
7	0.070605	-0.237764	-0.106356	-0.148842	-0.067239	-0.171495	-0.242642
8	-0.194163	-0.237764	-0.146490	-0.207790	0.058834	-0.196171	-0.242642

```
9   4.218624  4.358122  4.348539  4.331159  4.303281  4.319461  4.358406
10 -0.503058 -0.237764 -0.266892 -0.325684 -0.151287 -0.368900 -0.196861
11 -0.061779 -0.237764 -0.226758 -0.178316 -0.277360 -0.146820 -0.196861
12  0.070605 -0.237764 -0.266892 -0.089895 -0.193311 -0.146820 -0.242642
13  0.379500 -0.237764 -0.226758 -0.060421 -0.109263  0.050585 -0.242642
14 -0.458930 -0.237764 -0.307027 -0.355158 -0.403433 -0.393576 -0.242642
15 -0.017651 -0.237764 -0.226758 -0.296211 -0.193311  0.099936 -0.242642
16 -0.238290 -0.211198 -0.066221 -0.207790 -0.403433 -0.418251 -0.219752
17 -0.458930 -0.158067 -0.106356 -0.060421  0.142882 -0.097469 -0.219752
18  0.070605 -0.237764 -0.266892 -0.355158 -0.361408 -0.196171 -0.242642
19 -0.326546 -0.237764 -0.266892 -0.296211 -0.445457 -0.196171 -0.219752

[20 rows x 100 columns]
```

Modality 2: Image; Feature 3: SURF

In [13]:
```
im_features = mp.img_feature_vectorizer(100, option='SURF')
surf_features_df = pd.DataFrame(im_features)
surf_features_df
```

```
(56887, 64)
```

Out[13]:
```
           0         1         2         3         4         5         6  \
0  -0.555816 -0.344591  0.249576 -0.229416  0.120386 -0.296812 -0.410300
1  -0.095728 -0.073881  0.337922 -0.229416 -0.481543 -0.265194 -0.160371
2  -0.184777 -0.378430 -0.689095 -0.229416 -0.481543 -0.288908 -0.160371
3  -0.184777 -0.130279 -0.346756 -0.229416  0.722315 -0.233576 -0.410300
4  -0.348034 -0.231795 -0.390929 -0.229416 -0.481543 -0.344239  0.172868
5  -0.051203 -0.231795  0.128101 -0.229416 -0.481543 -0.273099 -0.326990
6  -0.258985 -0.412269 -0.678052 -0.229416 -0.481543 -0.249385 -0.535265
7   0.126895 -0.119000 -0.435102 -0.229416  2.528103 -0.281003 -0.368645
8  -0.184777 -0.265634 -0.048590 -0.229416  0.120386 -0.249385  0.047903
9   4.282530  4.280036  4.114694  4.358899  3.130032  4.343099  4.171731
10  0.023004 -0.164118  0.161231 -0.229416 -0.481543 -0.075487 -0.285336
11 -0.051203  0.185549 -0.313627 -0.229416 -0.481543 -0.170341 -0.202026
12 -0.333193  0.264506  0.183317 -0.229416 -0.481543 -0.162436  0.089558
13 -0.229302 -0.446107 -0.523447 -0.229416 -0.481543 -0.028061  0.714380
14 -0.570658 -0.130279  0.172274 -0.229416 -0.481543 -0.344239 -0.410300
15 -0.362876 -0.479946 -0.722225 -0.229416 -0.481543 -0.201959 -0.410300
16 -0.184777 -0.254355 -0.147979 -0.229416 -0.481543 -0.186150 -0.493610
17 -0.125411 -0.322032 -0.136936 -0.229416 -0.481543 -0.122914 -0.118716
18 -0.125411 -0.446107 -0.589706 -0.229416 -0.481543 -0.209863 -0.410300
19 -0.585499 -0.299473 -0.324670 -0.229416  0.120386 -0.360048 -0.493610

           7         8         9    ...           90        91        92  \
0  -0.473430 -0.452367 -0.496392    ...    -0.315104 -0.149644 -0.153375
1  -0.279402 -0.379169 -0.181607    ...     0.360119 -0.387832 -0.203662
```

2	-0.318207	-0.349889	-0.359178	...	-0.315104	-0.408544	-0.304236
3	-0.589848	-0.320610	-0.100893	...	0.360119	-0.289450	-0.002514
4	-0.551042	-0.540206	-0.464106	...	-0.315104	-0.361942	-0.203662
5	-0.357013	-0.247411	-0.254249	...	0.135045	0.140324	0.148347
6	-0.512236	-0.423088	-0.367249	...	-0.765254	-0.631198	-0.605958
7	0.108656	0.060023	-0.100893	...	-0.540179	-0.196246	-0.505384
8	-0.318207	-0.408448	-0.359178	...	0.135045	-0.227314	-0.103088
9	3.834011	4.042027	4.201169	...	3.961312	4.205053	4.070731
10	-0.589848	-0.423088	-0.351106	...	0.135045	-0.175534	0.852364
11	-0.007761	-0.174213	0.125107	...	-0.540179	-0.123754	-0.153375
12	0.147462	-0.188852	0.254249	...	0.810268	-0.046084	0.299208
13	1.311635	1.070163	0.601320	...	-0.540179	-0.165178	-0.605958
14	-0.589848	-0.496287	-0.520606	...	-0.315104	0.559742	-0.656245
15	0.690743	0.513854	-0.302678	...	-0.540179	-0.501748	-0.253949
16	-0.589848	-0.349889	-0.391463	...	-0.540179	-0.128932	-0.555671
17	-0.162984	-0.291330	-0.294606	...	-0.765254	-0.429256	-0.002514
18	-0.085373	-0.130294	-0.262321	...	-0.540179	-0.641554	-0.555671
19	-0.667459	-0.510926	-0.375321	...	0.135045	-0.040906	-0.505384

	93	94	95	96	97	98	99
0	-0.348333	0.310685	-0.488406	0.384755	-0.709324	-0.613490	-0.486306
1	-0.194939	-0.683507	0.201108	-0.714545	-0.464730	-0.711648	-0.486306
2	-0.412247	-0.683507	-0.373487	-0.714545	-0.097838	-0.122698	-0.226943
3	-0.233287	-0.186411	-0.450100	-0.714545	-0.342432	-0.711648	-0.421465
4	-0.386682	1.056329	0.047883	0.384755	-0.464730	-0.613490	-0.421465
5	-0.054327	-0.683507	-0.105343	-0.714545	0.146757	-0.024540	0.226943
6	-0.514510	1.553424	0.009577	0.384755	0.513649	0.957044	-0.291784
7	-0.207722	3.044712	0.162802	-0.714545	-0.464730	-0.515332	0.097261
8	0.047936	-0.186411	-0.258568	0.384755	0.391351	0.073619	-0.162102
9	4.215150	1.553424	4.184971	1.484054	3.937973	3.803638	4.247073
10	-0.258853	-0.683507	0.162802	-0.714545	0.635946	0.564411	0.291784
11	0.214113	0.062137	-0.067036	-0.714545	0.146757	-0.024540	-0.226943
12	0.431422	-0.434959	-0.258568	-0.714545	-0.709324	-0.711648	0.032420
13	-0.514510	-0.434959	-0.105343	-0.714545	-0.464730	-0.515332	-0.226943
14	0.137416	-0.683507	-0.794857	-0.714545	-0.831622	-0.711648	-0.486306
15	-0.514510	-0.683507	-0.220262	1.484054	-0.220135	0.269936	-0.097261
16	-0.322767	-0.186411	-0.296874	1.484054	0.269054	-0.515332	-0.356624
17	-0.246070	-0.683507	-0.028730	2.583354	-0.342432	0.662569	-0.291784
18	-0.527293	-0.683507	-0.603325	-0.714545	-0.097838	-0.122698	-0.291784
19	-0.309985	-0.683507	-0.718245	-0.714545	-0.831622	-0.417173	-0.421465

[20 rows x 100 columns]

Feature Concatenation

Text: BOW(3000 col) + TF-IDF(13 col) + Word2Vec(300 col)

In [28]:
```
text_features_df = pd.concat([bow_feature_df, tfidf_feature_df, wv_feature_df], axis = 1)
text_features_df
```

Out[28]:

	aa	aa insane	aaron	aaron family	ability	ability handle	ablility	\
0	0	0	1	1	0	0	0	
1	0	0	0	0	0	0	0	
2	0	0	0	0	0	0	0	
3	0	0	0	0	0	0	0	
4	0	0	0	0	0	0	0	
5	0	0	0	0	0	0	0	
6	0	0	0	0	0	0	0	
7	1	1	0	0	0	0	0	
8	0	0	0	0	0	0	0	
9	0	0	0	0	0	0	0	
10	0	0	0	0	0	0	0	
11	0	0	0	0	0	0	0	
12	0	0	0	0	0	0	0	
13	0	0	0	0	0	0	0	
14	0	0	0	0	0	0	0	
15	0	0	0	0	1	1	1	
16	0	0	0	0	0	0	0	
17	0	0	0	0	0	0	0	
18	0	0	0	0	0	0	0	
19	0	0	0	0	0	0	0	

	ablility handle	ac	ac adaptor	...	290	291	292	\
0	0	0	0	...	0.026471	-0.001058	-0.037982	
1	0	0	0	...	0.024618	0.002757	-0.038037	
2	0	0	0	...	0.027248	0.004349	-0.034061	
3	0	0	0	...	0.025188	0.004164	-0.036158	
4	0	0	0	...	0.026218	0.001301	-0.038024	
5	0	0	0	...	0.025125	0.005245	-0.033732	
6	0	0	0	...	0.028442	0.001760	-0.040036	
7	0	0	0	...	0.029046	0.004711	-0.032712	
8	0	0	0	...	0.026903	0.001145	-0.034262	
9	0	0	0	...	0.025240	0.010071	-0.036973	
10	0	1	1	...	0.026863	0.002797	-0.037480	
11	0	0	0	...	0.025233	0.002316	-0.034669	
12	0	0	0	...	0.023964	0.001423	-0.035920	
13	0	1	0	...	0.025282	0.003004	-0.036937	
14	0	0	0	...	0.025719	0.006645	-0.034013	
15	1	0	0	...	0.027127	0.004633	-0.036959	
16	0	0	0	...	0.027467	0.002993	-0.035956	
17	0	0	0	...	0.026705	0.000457	-0.033467	
18	0	0	0	...	0.028083	0.004155	-0.033117	

```
19              0   0          0    ...     0.025136  0.002955 -0.030935

         293       294       295       296       297       298       299
0   0.033718 -0.010165 -0.062011 -0.041459 -0.007137 -0.040725  0.056349
1   0.032780 -0.010477 -0.061677 -0.045496 -0.005746 -0.040618  0.050063
2   0.034251 -0.012408 -0.062688 -0.042718 -0.010747 -0.041362  0.057279
3   0.030534 -0.011656 -0.061273 -0.038613 -0.006891 -0.038747  0.056370
4   0.032200 -0.012778 -0.061774 -0.042119 -0.007339 -0.041163  0.060847
5   0.030699 -0.013031 -0.059054 -0.043476 -0.002783 -0.042552  0.053156
6   0.032523 -0.013378 -0.064923 -0.041276 -0.008187 -0.041436  0.057905
7   0.031138 -0.014588 -0.060625 -0.044377 -0.004594 -0.043658  0.052576
8   0.034708 -0.011263 -0.061471 -0.042379 -0.007524 -0.041796  0.054347
9   0.031803 -0.011111 -0.061931 -0.046313 -0.003641 -0.039562  0.055385
10  0.034381 -0.013811 -0.063596 -0.045905 -0.007685 -0.041709  0.056482
11  0.033251 -0.013138 -0.062028 -0.042660 -0.004591 -0.036896  0.054570
12  0.032694 -0.012121 -0.062201 -0.041078 -0.005902 -0.038998  0.054119
13  0.034426 -0.012254 -0.062353 -0.045223 -0.006469 -0.038880  0.055237
14  0.033189 -0.016299 -0.062951 -0.044412 -0.006223 -0.039079  0.054843
15  0.032797 -0.013859 -0.062631 -0.042387 -0.007902 -0.043068  0.054968
16  0.033559 -0.014305 -0.062611 -0.042249 -0.004880 -0.041806  0.054143
17  0.034944 -0.013156 -0.059124 -0.044696 -0.003734 -0.036333  0.055879
18  0.034657 -0.013079 -0.062221 -0.043816 -0.008999 -0.045358  0.055255
19  0.033070 -0.018818 -0.062221 -0.041482 -0.003093 -0.036871  0.061221

[20 rows x 3313 columns]
```

Image: ORB(100 col) + SIFT(100 col) + SURF(100 col)

In [29]:
```python
img_features_df = pd.concat([orb_features_df, sift_features_df, surf_features_df], axis = 1)
img_features_df
```

Out[29]:
```
           0         1         2         3         4         5         6  \
0   0.276563 -1.108874 -1.319950  0.770594 -0.813383  0.662637  0.506502
1   0.276563  0.431229  0.913812 -0.177830  1.272215 -0.261973 -0.578860
2  -1.259899 -0.492833 -0.913812 -1.126253 -1.647622 -1.186583 -1.302434
3  -0.030729 -0.492833  0.913812 -0.889147  0.437975 -1.186583  3.039013
4   0.583856  2.279352 -1.116881  2.193230  0.020856 -0.261973 -0.940647
5  -0.645314 -1.108874  0.507673 -0.652041 -1.647622 -0.878380 -0.217072
6  -1.567192 -0.800853  1.116881 -1.126253  0.855095  1.895451 -0.578860
7  -0.645314  0.123208 -0.101535  1.007700 -0.396264 -0.878380 -1.302434
8  -0.645314  1.663311 -0.304604  1.007700  1.272215 -0.261973  0.144715
9  -0.030729 -0.492833 -0.304604 -1.126253  1.272215  1.279044  1.591864
10 -0.952607 -1.416895  0.913812 -0.889147  0.437975 -0.261973  0.868290
11 -0.952607 -0.492833 -0.507673  1.481912  1.272215 -0.261973 -0.217072
12  0.276563  1.355290  0.101535  0.533489  0.855095 -0.878380 -0.578860
13  0.891148  0.739249 -1.319950 -0.889147 -0.396264  0.046231  0.144715
14  1.198441  1.355290 -1.319950  0.533489 -0.396264  1.895451 -0.940647
15  0.583856  0.123208  0.304604  0.059276 -0.813383  1.279044 -0.578860
```

```
16 -0.030729 -1.108874  0.507673 -1.363359 -0.813383  0.970841  0.144715
17  1.505733  0.123208 -0.304604 -0.652041  1.272215 -0.570176  0.506502
18 -1.259899 -0.492833  2.741435  0.533489 -0.813383 -1.494786 -0.217072
19  2.427611 -0.184812 -0.507673  0.770594 -1.230503  0.354434  0.506502

          7         8         9    ...          90        91        92  \
0  -0.066140 -0.332694 -0.916949    ...   -0.315104 -0.149644 -0.153375
1  -0.595257  0.450116 -1.269622    ...    0.360119 -0.387832 -0.203662
2  -1.124374 -0.332694 -0.564277    ...   -0.315104 -0.408544 -0.304236
3   2.314888 -1.115505 -1.269622    ...    0.360119 -0.289450 -0.002514
4  -0.595257 -1.506910 -0.211604    ...   -0.315104 -0.361942 -0.203662
5   2.314888 -0.332694 -0.211604    ...    0.135045  0.140324  0.148347
6   0.727536  0.058711  0.493742    ...   -0.765254 -0.631198 -0.605958
7  -0.330698  0.841521  1.199088    ...   -0.540179 -0.196246 -0.505384
8  -0.066140  2.407142  0.493742    ...    0.135045 -0.227314 -0.103088
9  -0.595257 -0.332694 -0.916949    ...    3.961312  4.205053  4.070731
10  0.198419 -0.332694 -0.564277    ...    0.135045 -0.175534  0.852364
11 -0.859815 -1.506910  2.609779    ...   -0.540179 -0.123754 -0.153375
12 -0.859815 -0.332694 -0.564277    ...    0.810268 -0.046084  0.299208
13 -1.388933 -0.724100  0.493742    ...   -0.540179 -0.165178 -0.605958
14 -0.595257  0.841521 -0.916949    ...   -0.315104  0.559742 -0.656245
15 -0.859815  0.450116  0.141069    ...   -0.540179 -0.501748 -0.253949
16  0.727536 -0.332694 -0.564277    ...   -0.540179 -0.128932 -0.555671
17  0.198419 -0.724100  1.551760    ...   -0.765254 -0.429256 -0.002514
18  0.992095  2.015737 -0.211604    ...   -0.540179 -0.641554 -0.555671
19  0.462978  0.841521  1.199088    ...    0.135045 -0.040906 -0.505384

         93        94        95        96        97        98        99
0  -0.348333  0.310685 -0.488406  0.384755 -0.709324 -0.613490 -0.486306
1  -0.194939 -0.683507  0.201108 -0.714545 -0.464730 -0.711648 -0.486306
2  -0.412247 -0.683507 -0.373487 -0.714545 -0.097838 -0.122698 -0.226943
3  -0.233287 -0.186411 -0.450100 -0.714545 -0.342432 -0.711648 -0.421465
4  -0.386682  1.056329  0.047883  0.384755 -0.464730 -0.613490 -0.421465
5  -0.054327 -0.683507 -0.105343 -0.714545  0.146757 -0.024540  0.226943
6  -0.514510  1.553424  0.009577  0.384755  0.513649  0.957044 -0.291784
7  -0.207722  3.044712  0.162802 -0.714545 -0.464730 -0.515332  0.097261
8   0.047936 -0.186411 -0.258568  0.384755  0.391351  0.073619 -0.162102
9   4.215150  1.553424  4.184971  1.484054  3.937973  3.803638  4.247073
10 -0.258853 -0.683507  0.162802 -0.714545  0.635946  0.564411  0.291784
11  0.214113  0.062137 -0.067036 -0.714545  0.146757 -0.024540 -0.226943
12  0.431422 -0.434959 -0.258568 -0.714545 -0.709324 -0.711648  0.032420
13 -0.514510 -0.434959 -0.105343 -0.714545 -0.464730 -0.515332 -0.226943
14  0.137416 -0.683507 -0.794857 -0.714545 -0.831622 -0.711648 -0.486306
15 -0.514510 -0.683507 -0.220262  1.484054 -0.220135  0.269936 -0.097261
16 -0.322767 -0.186411 -0.296874  1.484054  0.269054 -0.515332 -0.356624
17 -0.246070 -0.683507 -0.028730  2.583354 -0.342432  0.662569 -0.291784
18 -0.527293 -0.683507 -0.603325 -0.714545 -0.097838 -0.122698 -0.291784
19 -0.309985 -0.683507 -0.718245 -0.714545 -0.831622 -0.417173 -0.421465
```

```
[20 rows x 300 columns]
```

Then for the sparse feature set, we might use NB, Decision Tree for baseline, Ensemble methods or construct a Neural Network for further exploration.

Q4

4.(25 pts.) Suppose you have been hired by an AI consulting firm. Your clients are in a position to acquire software for data driven knowledge acquisition using one or more of the following: perceptron, decision tree, random forest, naive bayes. Indicate which algorithm you would choose in each of the following applications. In each case, briefly justify your recommendation.

- (a) Your client, has a database of patient records containing symptoms and expert diagnosis. She would like to build a diagnosis system. The attributes can be numeric (e.g., patient's temperature), as well as categorical (e.g., whether the patient is pregnant). In addition to performing accurate diagnosis of patients, your client would like to use the system to obtain insight regarding the relationships between features for different diseases.
- (b) Your client has a web-based information system for a large organization. She would like to enhance its functionality to support customization of information retrieved and presented to different users. He is able to record each users's actions when presented with specific documents in a given context. She would like to use such a database of records for designing a proactive information assistant for each user that goes out and retrieves documents that might be of interest to each user.
- (c) Your client is a financial organization which has managed to gather a large database of credit related information on its customers. Experts in the organization are convinced that they can automate the decision to approve or deny a loan based on a simple rule that checks whether one or more (usually a small number) of a set of possible conditions are satisfied. You are told that it is important that the decision-making process be transparent – that is, it should be easy to understand why a loan was approved or denied in each case.

(a)

NB, Perceptron probably work for this diagnosis system, but regarding the requirement to obtain insight regarding the relationships between features for different diseases, Naive Bayes is probably better.

- classification
  * Diagnosis of patients seems more like a multiclass classification problem, which could be solved by Generative model (NB) or Discriminative Model (MLP) with softmax.
- relationship between features for different diseases
  * if NB is applied for this task, then the relationship of certain features of different diseases could be further analyzed because NB attempts to solve intermediate problems of modeling the joint probability distribution of the features and classes

(b)

NB might be more optimal than the other three

Because in this context, if the a web-based search engine is to be built, then the goal should include document importance ranking (classification) and the ability to update the ranking based the new user action record that are collected by the system after each retrieve.

For instance, say there are two user feedback records x_1, x_2, and the label as θ

So

- first update:

$$p(\theta|x_1) = \frac{p(x_1|\theta)p(\theta)}{\sum p(x_1|\theta')p(\theta')}$$

then if we reuse the posterior to update the previous prior

- second update

$$p(\theta|x_1, x_2) = \frac{p(x_2|x_1, \theta)p(\theta|x_1)}{\sum p(x_2|x_1, \theta')p(\theta'|x_1)}$$

and since NB assumes that there exists no relations between x_1 and x_2 (otherwise it would turn into a markov model)

so

$$p(x_2|x_1, \theta) = p(x_2|\theta)$$

hence

$$p(\theta|x_1, x_2) = \frac{p(x_2|\theta)p(\theta|x_1)}{\sum p(x_2|\theta')p(\theta'|x_1)}$$

......

n updates could be deduced in this logic

(c)

Decision Tree is better, the classification accuracy of DTree might be not as high as Random Forest, however, if the transparency of the decision-making process is very important, so Decision Tree is more optimal than RF in this case because it should be easy to visualize the decision process to understand why a loan was approved or denied in each case.

For instance:

```
In [100]: loan = pd.read_csv('/home/karen/Downloads/data/loan.csv')
          y = loan.grade
          X = loan.drop(columns=['member_id', 'id', 'grade', 'sub_grade', 'emp_title', 'desc'])
          X = X.fillna(value=0)
          X = X[['int_rate', 'home_ownership', 'annual_inc', 'loan_amnt', 'loan_status', 'installment']]
          X.head()
```

Out[100]:

	int_rate	home_ownership	annual_inc	loan_amnt	loan_status	installment
0	10.65	RENT	24000.0	5000.0	Fully Paid	162.87
1	15.27	RENT	30000.0	2500.0	Charged Off	59.83
2	15.96	RENT	12252.0	2400.0	Fully Paid	84.33
3	13.49	RENT	49200.0	10000.0	Fully Paid	339.31
4	12.69	RENT	80000.0	3000.0	Current	67.79

In [102]:
```
from sklearn.preprocessing import MinMaxScaler, LabelBinarizer
```

In [103]:
```
cat_cols = X.columns[X.dtypes==object]
```

In [114]:
```
def Encoder(data, var_list):
    try:
        encode_cat_df = pd.DataFrame()
        for var in var_list:
            encoder = LabelBinarizer()
            res = encoder.fit_transform(data[var])
            col = list(map(lambda x: '{} :'.format(var)+x,
                           encoder.classes_))
            encode_cat_df = pd.concat([encode_cat_df,
                                       pd.DataFrame(res, columns=col)],
                                       axis=1)
        return encode_cat_df
    except Exception as e:
        print(e)
```

In [115]:
```
X_cat_df = Encoder(X, cat_cols)
X_encoded = pd.concat([X[X.columns[X.dtypes!='object']], X_cat_df], axis=1)
X_encoded.head()
```

Out[115]:

	int_rate	annual_inc	loan_amnt	installment	home_ownership :ANY	\
0	10.65	24000.0	5000.0	162.87	0	
1	15.27	30000.0	2500.0	59.83	0	
2	15.96	12252.0	2400.0	84.33	0	
3	13.49	49200.0	10000.0	339.31	0	
4	12.69	80000.0	3000.0	67.79	0	

	home_ownership :MORTGAGE	home_ownership :NONE	home_ownership :OTHER	\
0	0	0	0	
1	0	0	0	
2	0	0	0	
3	0	0	0	
4	0	0	0	

	home_ownership :OWN	home_ownership :RENT	loan_status :Charged Off	\
0	0	1	0	
1	0	1	1	
2	0	1	0	
3	0	1	0	

```
4                  0                   1                      0

   loan_status :Current  loan_status :Default  \
0                  0                    0
1                  0                    0
2                  0                    0
3                  0                    0
4                  1                    0

   loan_status :Does not meet the credit policy. Status:Charged Off  \
0                                                0
1                                                0
2                                                0
3                                                0
4                                                0

   loan_status :Does not meet the credit policy. Status:Fully Paid  \
0                                                0
1                                                0
2                                                0
3                                                0
4                                                0

   loan_status :Fully Paid  loan_status :In Grace Period  loan_status :Issued  \
0                     1                            0                    0
1                     0                            0                    0
2                     1                            0                    0
3                     1                            0                    0
4                     0                            0                    0

   loan_status :Late (16-30 days)  loan_status :Late (31-120 days)
0                            0                                0
1                            0                                0
2                            0                                0
3                            0                                0
4                            0                                0
```

In [119]:
```
from sklearn import tree
from graphviz import Source
clf = DecisionTreeClassifier(criterion = 'entropy',
                             random_state = 100,
                             max_depth = 10,
                             min_samples_leaf = 2)
clf.fit(X_encoded, y)
dot_data = tree.export_graphviz(clf, out_file=None, feature_names=X_encoded.columns)
graph = Source(dot_data)
graph.render('DtreeVis')
```

Out[119]:
```
'DtreeVis.pdf'
```