In [ ]:
```
#Group 8 - Wk8 - Factorization and Embeddings
#Group Member: Athena Zhang, Pratheek Praveen Kumar, Weifeng Li, Wenke Yu, Ziq
iao Wei
```

Make sure you installed ***gensim***, ***sklearn***, ***matplotlib*** and ***numpy*** if you use your local machine

In [1]:
```
!pip install gensim
```

```
Looking in indexes: https://pypi.org/simple, https://us-python.pkg.dev/colab-
wheels/public/simple/
Requirement already satisfied: gensim in /usr/local/lib/python3.7/dist-packag
es (3.6.0)
Requirement already satisfied: numpy>=1.11.3 in /usr/local/lib/python3.7/dist
-packages (from gensim) (1.21.6)
Requirement already satisfied: six>=1.5.0 in /usr/local/lib/python3.7/dist-pa
ckages (from gensim) (1.15.0)
Requirement already satisfied: smart-open>=1.2.1 in /usr/local/lib/python3.7/
dist-packages (from gensim) (6.0.0)
Requirement already satisfied: scipy>=0.18.1 in /usr/local/lib/python3.7/dist
-packages (from gensim) (1.4.1)
```

In [2]:
```
%matplotlib inline
import matplotlib.pyplot as plt
import numpy as np
import sklearn
from sklearn import datasets
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.decomposition import TruncatedSVD
from sklearn.neighbors import NearestNeighbors, KNeighborsClassifier
import pandas as pd
import matplotlib.cm as cm
from sklearn.manifold import TSNE, Isomap, LocallyLinearEmbedding, MDS, Spectr
alEmbedding
from sklearn.preprocessing import *
from sklearn.metrics import accuracy_score
pd.set_option('display.max_colwidth', -1)
```

```
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:15: FutureWarnin
g: Passing a negative integer is deprecated in version 1.0 and will not be su
pported in future version. Instead, use None to not limit the column width.
  from ipykernel import kernelapp as app
```

In [3]:
```
categories = ['soc.religion.christian', 'sci.space', 'rec.sport.hockey', 'com
p.os.ms-windows.misc','talk.politics.guns']
# categories = ['alt.atheism', 'soc.religion.christian']
# categories = ['comp.sys.ibm.pc.hardware', 'comp.sys.mac.hardware']
# categories = ['rec.sport.baseball', 'rec.sport.hockey']
# 'alt.atheism','comp.graphics','comp.os.ms-windows.misc','comp.sys.ibm.pc.har
dware',
# 'comp.sys.mac.hardware','comp.windows.x', 'misc.forsale', 'rec.autos',
# 'rec.motorcycles',  'rec.sport.baseball', 'rec.sport.hockey', 'sci.crypt',
# 'sci.electronics', 'sci.med', 'sci.space', 'soc.religion.christian', 'talk.p
olitics.guns',
# 'talk.politics.mideast', 'talk.politics.misc', 'talk.religion.misc'
train = sklearn.datasets.fetch_20newsgroups(subset='train', categories=categor
ies, remove=('headers', 'footers', 'quotes'),)
test = sklearn.datasets.fetch_20newsgroups(subset='test', categories=categorie
s, remove=('headers', 'footers', 'quotes'),)
print('train data size:', len(train.data))
print('test data size:', len(test.data))
```

```
train data size: 2929
test data size: 1949
```

In [4]:
```
features = TfidfVectorizer(lowercase=True, stop_words='english', min_df=2, max
_df=0.5, ngram_range = (1,2))
train.vecs = features.fit_transform(train.data)
test.vecs = features.transform(test.data)
```

In [5]:
```
num_neighs = 10
metric = 'cosine'
nbrs_vecs = NearestNeighbors(n_neighbors=num_neighs, algorithm='brute', metric
=metric).fit(train.vecs)
```

In [6]:
```
len(features.get_feature_names())
```

```
/usr/local/lib/python3.7/dist-packages/sklearn/utils/deprecation.py:87: Futur
eWarning: Function get_feature_names is deprecated; get_feature_names is depr
ecated in 1.0 and will be removed in 1.2. Please use get_feature_names_out in
stead.
  warnings.warn(msg, category=FutureWarning)
```

Out[6]: 48061

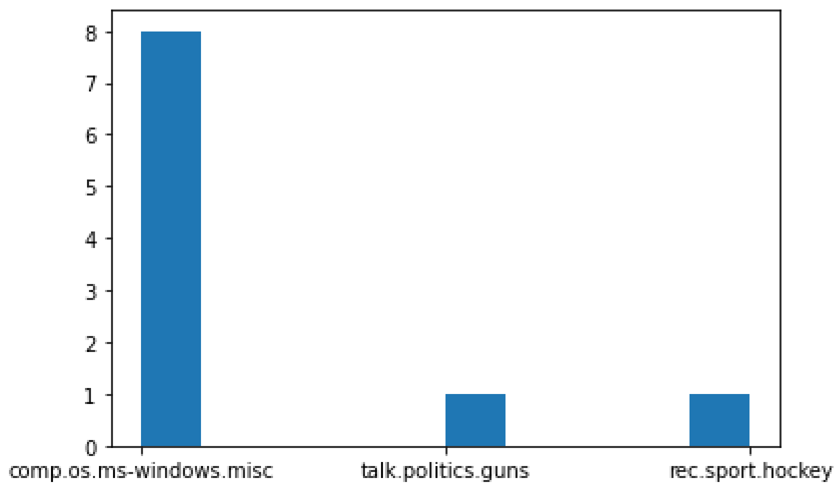
In [7]:
```
idx = 200
inst = test.data[idx]
test.target_names[test.target[idx]]
pd.DataFrame.from_dict({'category':[test.target_names[test.target[idx]]], 'ema
il':[inst]})
```

Out[7]:

	category	email
0	comp.os.ms-windows.misc	Hi, can anyone tell me what Microsoft BBS number is ? I tried the one\nthat is given on the DOS 6 upgrade manual but that number never\nanswered the call ...

In [8]:
```
distances, indices = nbrs_vecs.kneighbors(test.vecs[idx])
plt.hist([train.target_names[nidx] for nidx in train.target[indices][0]])
```

Out[8]:
```
(array([8., 0., 0., 0., 0., 1., 0., 0., 0., 1.]),
 array([0. , 0.2, 0.4, 0.6, 0.8, 1. , 1.2, 1.4, 1.6, 1.8, 2. ]),
 <a list of 10 Patch objects>)
```



In [9]:
```
nbrs_vecs_classifier = KNeighborsClassifier(num_neighs).fit(train.vecs, train.
target)
print('vec', accuracy_score(test.target, nbrs_vecs_classifier.predict(test.vec
s)))
```

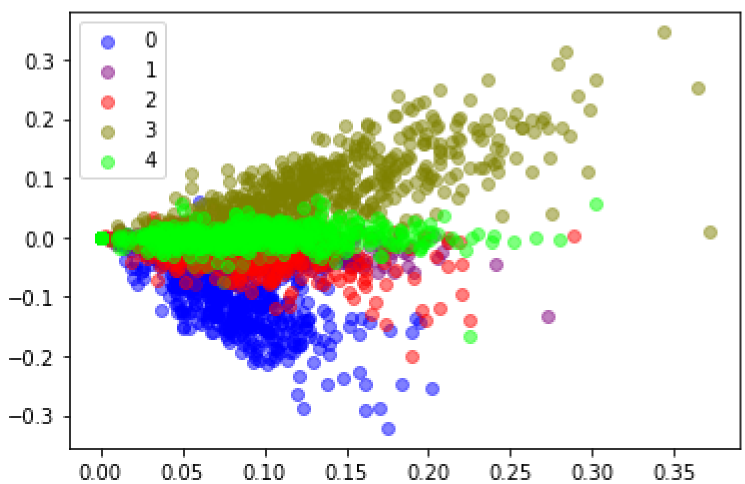
```
vec 0.22267829656233967
```

Latent Semantic Analysis (LSA)

In [10]:
```
def do_plot(X_fit, labels=train.target):
  dimension = X_fit.shape[1]
  label_types = list(set(labels))
  num_labels = len(list(set(labels)))
  colors = cm.brg(np.linspace(0, 1, num_labels))
  if num_labels == X_fit.shape[0]:
    label_types = sorted(label_types, key=lambda k: np.where(labels==k))
    colors = cm.seismic(np.linspace(0, 1, num_labels))
  if dimension == 2:
    for lab, col in zip(label_types, colors):
      plt.scatter(X_fit[labels==lab, 0],
                  X_fit[labels==lab, 1],
                  label=lab,
                  c=col, alpha=0.5)
  else:
    raise Exception('Unknown dimension: %d' % dimension)
  plt.legend(loc='best')
  plt.show()
```

In [11]:
```
factors = TruncatedSVD(2).fit_transform(train.vecs.toarray())
do_plot(factors)
```

```
*c* argument looks like a single numeric RGB or RGBA sequence, which should b
e avoided as value-mapping will have precedence in case its length matches wi
th *x* & *y*.  Please use the *color* keyword-argument or provide a 2-D array
with a single row if you intend to specify the same RGB or RGBA value for all
points.
*c* argument looks like a single numeric RGB or RGBA sequence, which should b
e avoided as value-mapping will have precedence in case its length matches wi
th *x* & *y*.  Please use the *color* keyword-argument or provide a 2-D array
with a single row if you intend to specify the same RGB or RGBA value for all
points.
*c* argument looks like a single numeric RGB or RGBA sequence, which should b
e avoided as value-mapping will have precedence in case its length matches wi
th *x* & *y*.  Please use the *color* keyword-argument or provide a 2-D array
with a single row if you intend to specify the same RGB or RGBA value for all
points.
*c* argument looks like a single numeric RGB or RGBA sequence, which should b
e avoided as value-mapping will have precedence in case its length matches wi
th *x* & *y*.  Please use the *color* keyword-argument or provide a 2-D array
with a single row if you intend to specify the same RGB or RGBA value for all
points.
*c* argument looks like a single numeric RGB or RGBA sequence, which should b
e avoided as value-mapping will have precedence in case its length matches wi
th *x* & *y*.  Please use the *color* keyword-argument or provide a 2-D array
with a single row if you intend to specify the same RGB or RGBA value for all
points.
```



In [12]:
```
factors[0]
```

Out[12]:
```
array([ 0.10681913, -0.02249651])
```

In [13]:
```
svd = TruncatedSVD(300)
train.svd = svd.fit_transform(train.vecs.toarray())
test.svd = svd.transform(test.vecs.toarray())
train.svd.shape, test.svd.shape
```

Out[13]:
```
((2929, 300), (1949, 300))
```

In [14]:
```
test.svd[0]
```

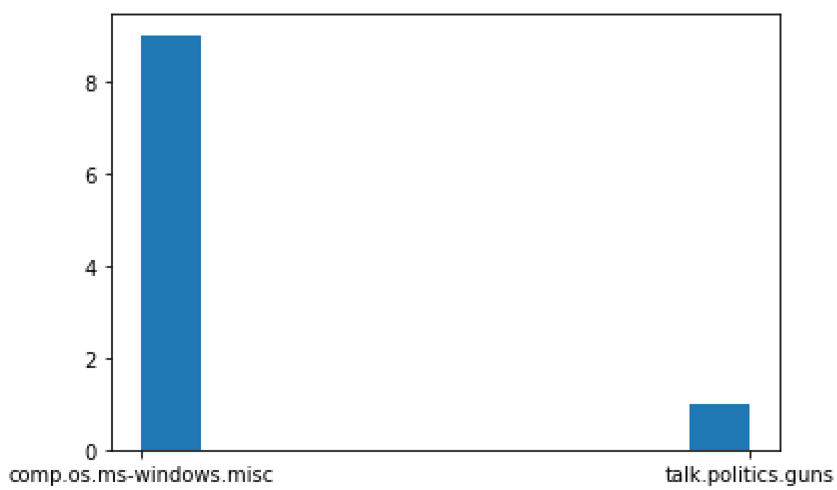
Out[14]:
```
array([ 1.25321198e-01, -1.48266000e-02,  3.81248353e-02,  6.72477328e-02,
       -1.46419326e-03,  5.21709408e-02,  1.58178314e-02, -2.72060763e-02,
       -1.12261329e-02, -3.47200305e-02,  8.03806108e-03,  3.62409950e-03,
        1.72493077e-02, -1.91896233e-02, -1.51880709e-02,  3.85531994e-02,
       -1.70361698e-02,  3.53939240e-03, -8.30662294e-03, -2.36528846e-02,
       -5.72654405e-03, -1.88096378e-02,  1.89145151e-02, -9.25402046e-03,
       -5.61902679e-03, -3.82808167e-02, -1.74956101e-02,  1.26219454e-03,
        1.35475347e-02,  1.94385970e-02,  8.76261007e-03,  5.17530670e-02,
        7.89949199e-02,  7.77657589e-03,  9.94460160e-03, -1.49980947e-02,
       -4.21810664e-02,  2.64693928e-02,  3.82854543e-04, -2.63121328e-02,
        3.05936622e-02,  1.55824429e-02, -2.17101721e-02, -5.43490712e-03,
       -1.04535934e-02,  2.74055622e-02,  1.83673818e-04, -3.55704524e-02,
        4.81355296e-03,  4.87310055e-02,  6.00548881e-04, -4.24654747e-03,
        1.77073772e-03, -3.25958943e-02,  1.09529235e-02, -2.38986665e-02,
        3.55183490e-03, -2.48773249e-03, -7.92832675e-03,  2.32702119e-02,
        2.94866075e-04, -6.00251280e-03, -6.94433075e-03, -1.70469584e-02,
       -5.80060453e-03, -1.80487236e-02,  1.74294046e-02, -2.22944637e-02,
       -3.19650998e-02,  7.54950107e-03, -3.25033859e-02,  8.37144306e-03,
        2.30185431e-03,  4.94820893e-03, -1.62564190e-02,  1.20581210e-02,
        3.24051396e-03, -2.76033501e-03,  1.51293754e-02,  2.63619254e-03,
       -2.61577467e-02,  1.11121935e-02,  8.85525198e-03, -1.19408006e-02,
       -1.09520165e-02,  1.03492669e-02,  2.36504044e-02,  1.13748677e-02,
       -1.25545916e-02, -3.70897729e-03, -3.42446652e-02,  1.89130740e-02,
       -9.46718080e-04, -2.65290360e-02,  1.05780362e-02, -2.13994284e-02,
       -1.00156855e-02,  2.11493971e-03, -1.82260789e-02, -1.74380413e-03,
       -2.19822869e-02,  2.84435979e-02,  1.60516947e-02,  1.41035527e-02,
       -1.18039440e-02,  2.46216168e-02, -1.13605011e-02, -1.21634087e-02,
        3.64374869e-03, -1.72980917e-02,  2.14444017e-02, -1.41944280e-02,
        2.03039188e-02,  1.46787970e-02,  1.05863674e-02,  5.29457734e-03,
        9.97815051e-03,  2.33095377e-02,  7.47692211e-03, -1.36601417e-02,
        9.59734825e-03, -5.53294841e-03,  3.79623268e-02,  1.53435847e-02,
        1.34354443e-02,  6.41118404e-03, -3.35147885e-03,  6.11252955e-03,
        2.50158284e-03, -1.38884131e-02,  1.25130158e-02,  1.15605967e-03,
        3.46763262e-02, -9.13626993e-04, -1.98050391e-02,  1.52303497e-02,
       -1.40129369e-03, -2.16456834e-02,  1.76089029e-02, -1.27218105e-02,
       -1.02716194e-02,  3.49762811e-03,  1.46893901e-02, -1.40856146e-02,
       -1.86749865e-03,  1.11190278e-02, -1.64212670e-02, -1.03792650e-02,
       -5.26357494e-03, -2.98756423e-03, -1.92909111e-02,  1.91152998e-02,
        7.35617422e-03, -1.35730703e-02,  4.78609395e-03, -1.79148689e-02,
       -1.83183876e-02,  2.19466574e-02,  2.15956182e-02, -4.81095575e-03,
        9.12050754e-03,  3.95421056e-03, -1.00906941e-02,  1.09702935e-03,
        3.80533320e-03,  1.79271941e-02,  5.89009306e-03, -9.30195012e-03,
        1.17795126e-03, -2.84163652e-04,  8.95565466e-03,  6.80064376e-03,
        1.52397759e-03,  4.30253226e-03,  8.16007325e-03,  2.25976183e-02,
       -3.13411310e-03,  2.20978279e-02,  3.79939782e-03, -3.20067999e-03,
       -4.19542206e-03,  9.53476731e-03,  3.36149234e-02, -1.15666004e-02,
       -2.57455942e-02, -7.16514497e-03, -7.67847599e-03, -1.80633803e-02,
        2.96236275e-02,  3.00680752e-03,  1.55377336e-02, -1.20510329e-02,
       -1.52270010e-04, -1.74679047e-03, -2.84052028e-02, -7.82362228e-03,
        1.79150056e-02, -3.89667669e-04,  1.09253136e-02,  1.45252117e-04,
       -1.39083261e-02,  3.32194969e-02,  5.64168575e-03, -6.42107982e-03,
       -1.16389530e-03, -8.46771021e-04,  3.76166574e-03, -1.69452205e-03,
       -3.41689511e-02,  3.49525105e-02,  1.49831917e-02,  3.86148703e-03,
        2.82502834e-02, -1.10256279e-02,  1.00180930e-02,  1.02146658e-03,
        1.94753349e-03,  8.80067997e-03, -1.16571295e-02, -6.50687893e-03,
       -3.16927594e-03,  4.98793990e-03,  4.62036865e-03,  1.68032151e-03,
       -5.28985149e-03, -2.06015978e-03, -1.60815028e-02,  8.98600972e-03,
```

```
        4.72956184e-03, -2.68247223e-02,  2.76992470e-03,  7.71500704e-06,
       -4.21543400e-03, -2.09880632e-02,  3.58628758e-02,  1.77792255e-02,
       -8.08009076e-03,  1.36145051e-03,  2.06942409e-03,  3.32021273e-03,
        2.36672687e-02, -3.78313463e-03,  3.40024298e-02,  1.05432991e-02,
        2.97868461e-03, -1.54697891e-02, -1.85129424e-02, -2.55839847e-02,
        2.73953919e-02, -1.19613706e-03, -2.57255336e-03,  6.99701690e-03,
       -1.84913080e-02,  7.04764589e-03,  1.29755458e-02, -1.71667877e-02,
       -1.52565021e-03, -1.28038395e-02,  1.47281805e-02,  2.21260123e-02,
        1.17633982e-02, -1.47635234e-02,  1.42903172e-02,  2.45091983e-02,
        9.51405864e-03, -1.03630785e-02,  5.75761695e-03, -4.10685715e-04,
        1.05512362e-02,  8.46115527e-03,  3.64245432e-03,  8.37157862e-03,
        1.20546144e-02,  8.30668091e-03,  1.62778443e-02,  5.47358954e-03,
       -1.30474231e-02, -9.54361117e-03, -1.09716725e-03, -8.58248372e-03,
       -1.67055538e-02,  2.41104733e-03, -9.44859018e-03,  2.97608502e-02,
       -1.94035152e-02, -7.43720820e-04, -1.47974939e-02,  1.05236379e-02,
        1.26814669e-02,  7.00130403e-03, -5.27079723e-03, -5.68559463e-03,
       -1.47210231e-02,  7.01552494e-03, -5.36755689e-03,  5.75472724e-03,
        1.92245025e-02,  2.35823655e-02,  1.12844371e-02,  4.76277014e-03])
```

In [15]:
```
num_neighs = 10
metric = 'cosine'
nbrs_svd = NearestNeighbors(n_neighbors=num_neighs, algorithm='brute', metric=
metric).fit(train.svd)
```

In [16]:
```
distances, indices = nbrs_svd.kneighbors(test.svd[idx].reshape(1, -1))
plt.hist([train.target_names[nidx] for nidx in train.target[indices][0]])
```

Out[16]:
```
(array([9., 0., 0., 0., 0., 0., 0., 0., 0., 1.]),
 array([0. , 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1. ]),
 <a list of 10 Patch objects>)
```



In [17]:
```
nbrs_svd_classifier = KNeighborsClassifier(num_neighs).fit(train.svd, train.ta
rget)
print('svd', accuracy_score(test.target, nbrs_svd_classifier.predict(test.svd
)))
```

```
svd 0.3206772703950744
```

In [18]:
```
print(svd.singular_values_)
```

```
[5.63850107 3.73771134 3.53781742 3.18942905 3.15625934 2.99110151
 2.6751269  2.65953153 2.48499554 2.40376838 2.38354388 2.35437232
 2.31923097 2.28716754 2.27333649 2.24139425 2.22447221 2.21047446
 2.18370957 2.1561941  2.13763945 2.12819825 2.1205137  2.10551204
 2.08564543 2.06775732 2.05326153 2.03944463 2.02827818 2.01610409
 2.00145111 1.98967376 1.9821122  1.97438537 1.95742271 1.94806465
 1.94177614 1.93196531 1.92156626 1.91620083 1.9005185  1.89663007
 1.88965615 1.88415265 1.87793649 1.8692284  1.86264857 1.85743302
 1.85572497 1.84483513 1.84075377 1.83687134 1.83030757 1.82472782
 1.81826187 1.80709663 1.79984962 1.79676276 1.79048736 1.78590533
 1.78326464 1.78192864 1.77730857 1.77188266 1.76780018 1.76402778
 1.762479   1.75683647 1.75524134 1.74659519 1.74515439 1.74261599
 1.73854014 1.73619172 1.73190534 1.72667462 1.72543462 1.72241204
 1.71743748 1.71151268 1.70742229 1.70427827 1.7020915  1.70179346
 1.69603705 1.69441768 1.68910294 1.68786622 1.68416122 1.68111387
 1.67904129 1.67468996 1.67157168 1.6702637  1.6675661  1.66560542
 1.66233862 1.65722475 1.65468119 1.65329019 1.64873321 1.64605909
 1.64547005 1.6425919  1.64220758 1.6394129  1.63499425 1.63311088
 1.62840444 1.62757149 1.62451776 1.62356887 1.62266499 1.62156934
 1.61989338 1.61862687 1.61610817 1.61280231 1.60932689 1.60561495
 1.60319587 1.60098001 1.59752243 1.59536523 1.59182699 1.59101774
 1.588678   1.58592849 1.58469316 1.58259068 1.58038061 1.57924673
 1.57592693 1.57538122 1.57344897 1.57007004 1.56747921 1.56552446
 1.56456269 1.56162322 1.55962961 1.55855203 1.55710767 1.5551917
 1.55204302 1.5515775  1.54775153 1.54726774 1.54453048 1.54236887
 1.5419652  1.54035477 1.53910468 1.53720504 1.53655214 1.53405338
 1.53340071 1.53067848 1.52693399 1.52651379 1.52455093 1.5230689
 1.52087768 1.51774498 1.51545936 1.51364152 1.51288806 1.51063305
 1.50927248 1.50900466 1.50725122 1.50620601 1.50467662 1.50114628
 1.49979216 1.4972818  1.49611599 1.49386196 1.49232046 1.4910955
 1.48980171 1.48702294 1.48474624 1.48373964 1.48116945 1.48029616
 1.47845563 1.47612296 1.47574796 1.47403561 1.4710579  1.47046788
 1.46949304 1.46780539 1.46700331 1.46432322 1.46424098 1.46029252
 1.45838179 1.45747756 1.45670346 1.45557901 1.45452063 1.45196282
 1.45099052 1.45020657 1.44813566 1.44503341 1.44479359 1.44242919
 1.44164187 1.4387567  1.43822152 1.43663751 1.4352563  1.43421037
 1.43214676 1.43026912 1.42836846 1.42672844 1.42557378 1.42320732
 1.42001361 1.4181948  1.41737979 1.41644307 1.41433165 1.41346979
 1.41140845 1.4102154  1.40971601 1.40796073 1.40731854 1.40472769
 1.40380032 1.40363516 1.40157506 1.39960795 1.39784412 1.39671035
 1.3942513  1.39208838 1.39081945 1.3892585  1.38689871 1.38346516
 1.38316553 1.38223378 1.37977545 1.37698851 1.37561564 1.37465712
 1.37424951 1.37251292 1.37044866 1.36945475 1.36712068 1.36657229
 1.36546564 1.36375502 1.36360812 1.36205405 1.36075039 1.35869068
 1.35781459 1.35607826 1.35507241 1.35318967 1.34970352 1.34809672
 1.34753476 1.34639468 1.34414887 1.34291776 1.34235118 1.33840644
 1.33678478 1.33596949 1.33380144 1.33255207 1.33150346 1.33087761
 1.32774423 1.32615047 1.32490327 1.32378612 1.32221945 1.32071667
 1.31681899 1.31640735 1.31485211 1.31109635 1.30903408 1.30718647
 1.30368641 1.30308402 1.29957454 1.29864295 1.29645385 1.29272952]
```

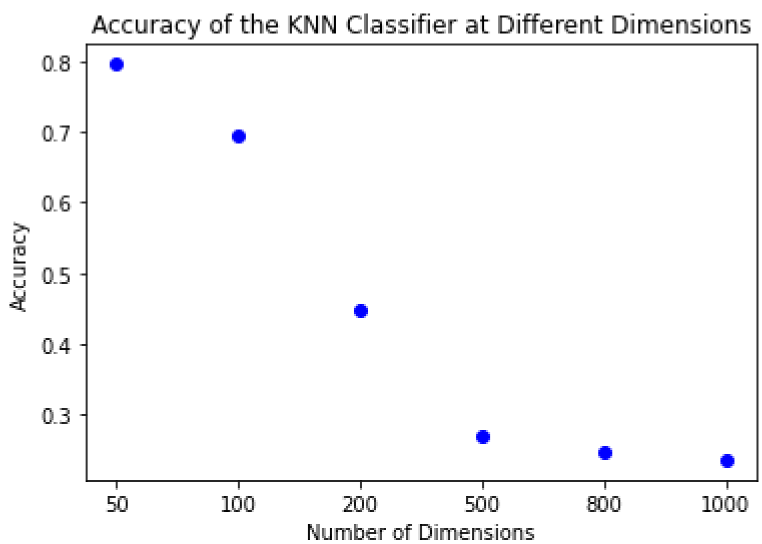
Q1. Plot the change in accuracy of the KNN classifier as we change the number of dimensions from 50-1000.

Q2. Find a document which has a very distribution of nearest neighbours if we choose a TFIDF representation vs a SVD(300) representation.

In [20]:
```
accuracies = np.array([])
for d in ([50,100,200,500,800,1000]):
  svd = TruncatedSVD(d)
  train.svd = svd.fit_transform(train.vecs.toarray())
  test.svd = svd.transform(test.vecs.toarray())
  train.svd.shape, test.svd.shape
  nbrs_svd_classifier = KNeighborsClassifier(num_neighs).fit(train.svd, train.
target)
  accuracy = accuracy_score(test.target, nbrs_svd_classifier.predict(test.svd
))
  accuracies = np.append(accuracies, [accuracy])
print(accuracies)
```

```
[0.7963058  0.69368907 0.44792201 0.26885582 0.24781939 0.23550539]
```

In [22]:
```
plt.title('Accuracy of the KNN Classifier at Different Dimensions')
plt.xlabel('Number of Dimensions')
plt.ylabel('Accuracy')
plt.plot(['50','100','200','500','800','1000'],accuracies,"ob")
plt.show()
```



In [ ]:
```
svd = TruncatedSVD(1000)
train.svd = svd.fit_transform(train.vecs.toarray())
test.svd = svd.transform(test.vecs.toarray())
train.svd.shape, test.svd.shape
```

Out[ ]:
```
((2929, 1000), (1949, 1000))
```

In [ ]:
```
nbrs_svd_classifier = KNeighborsClassifier(num_neighs).fit(train.svd, train.ta
rget)
print('svd', accuracy_score(test.target, nbrs_svd_classifier.predict(test.svd
)))
```

```
svd 0.2314007183170857
```

In [ ]:
```
svd = TruncatedSVD(2)
train.svd = svd.fit_transform(train.vecs.toarray())
test.svd = svd.transform(test.vecs.toarray())
train.svd.shape, test.svd.shape
```

Out[ ]:
```
((2929, 2), (1949, 2))
```

In [ ]:
```
nbrs_svd_classifier = KNeighborsClassifier(num_neighs).fit(train.svd, train.ta
rget)
print('svd', accuracy_score(test.target, nbrs_svd_classifier.predict(test.svd
)))
```

```
svd 0.5684966649563878
```

Q2. Find a document which has a very distribution of nearest neighbours if we choose a TFIDF representation vs a SVD(300) representation.

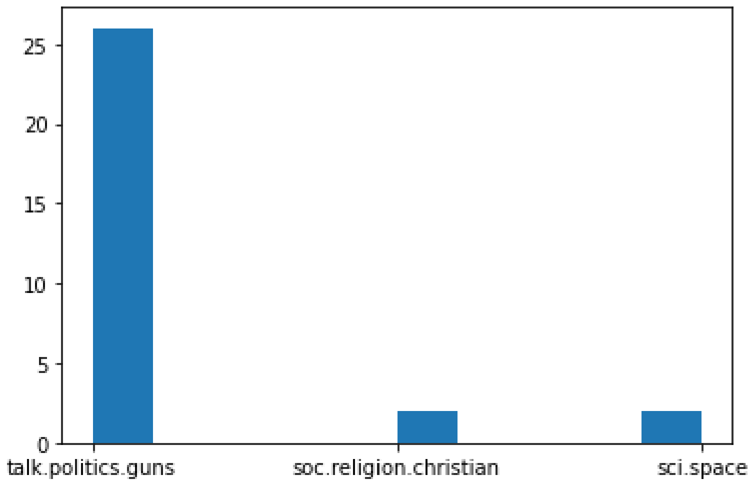
In [23]:
```
idx = 20
inst = test.data[idx]
test.target_names[test.target[idx]]
pd.DataFrame.from_dict({'category':[test.target_names[test.target[idx]]], 'email':[inst]})
```

Out[23]:

	category	email
0	talk.politics.guns	This is the AP story from Fri morning.\n\nAs the walls came tumbling down and tear gas filled the air, cult leader\nDavid Koresh sprang into action. He left his third-floor bedroom and began\nlooking around the house, making sure women and children were secure and \nchecking that everyone had their gas masks on properly. Within hours, the \ncompound became an inferno. Nine Branch Davidians excaped.\n This is their story, gleaned from lawyers who spoke with six of them\nwho are jailed on charges that include conspiracy and murder. That day the \nsix said a portable radio offered the only contact with the outside world \nsince Koresh's right-hand man, Steve Schneider, ripped out the compounds's \nphone line after FBI agents called before dawn Monday saying this was the\ncults last chance: Come out or prepare to get forced out.\n They kept their word. By dawn, tanks were battering the Mount Carmel\ncompound, punching for hours to creat holes for tear gas to enter. The BD\nmeanwhile proceeded with their daily routines. Strapped into gas masks, the\nwomen did laundry. Others read Bibles in their rooms. The 17 children, all\nunder 10, remained by their mothers' sides. Still, it was hard to ignore what \nwas happening around them. Each time a tank rammed the poorly-constructed building\nit shook violently. Cult members dodges falling gypsum wallboard and doors.\nHundreds of gas canisters hurled in from the armored vehicles were filling\nthe air with noxious fumes. The flying canisters were more frightening than\nthe tanks. At least one man was hit in the face. The gas began filling the air,\ndriven by heavy gusts of wind coming through windows and the holes the tanks\nmade. Scattered throughout the house, the cult members made no efforts to\ngather. Then the FBI sent in its biggest weapon -- a massive armored vehicle\nheaded for a chamber, lined with cinder blocks, where authorities hoped to \nfind Koresh and Schneider and fire tear gas directly at them.\n Here the cult members' story diverges from the government's version. The\nFBI says cult members set fires in three places. But each of the six cult\nmembers, in separate discussions with lawyers, consistently gave versions\nat odds with the FBI's account. They say the tank flattened a barrel of \npropane, spilling its contents. And as the tank thundered through the house,\nit tipped over lit lanterns, spitting flames that ignited the propane and\nother flammables. The home of used lumber, plywood, and wallboard tacked \ntogether with tar paper was vulnerable. The building erupted. Nine BD's\nescaped jumping through windows and dashing through other openings. Others\ndied groping in the blackness.

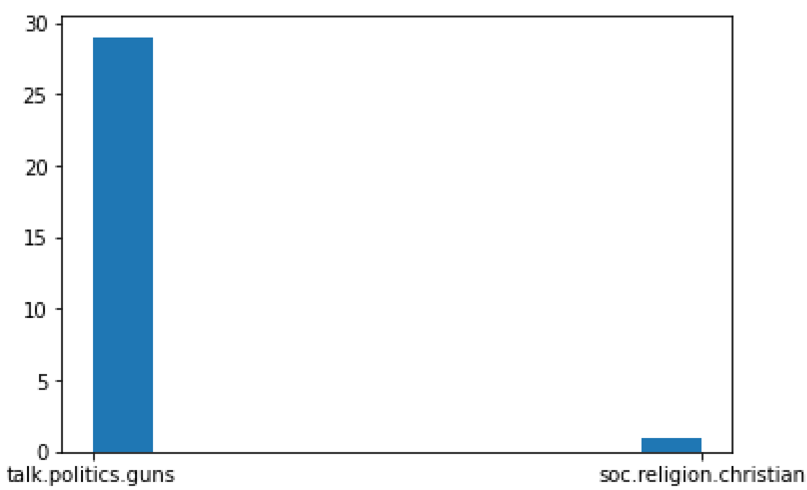
In [25]:
```
num_neighs = 30
metric = 'cosine'
nbrs_vecs = NearestNeighbors(n_neighbors=num_neighs, algorithm='brute', metric
=metric).fit(train.vecs)
distances, indices = nbrs_vecs.kneighbors(test.vecs[idx])
plt.hist([train.target_names[nidx] for nidx in train.target[indices][0]])
```

Out[25]:
```
(array([26.,  0.,  0.,  0.,  0.,  2.,  0.,  0.,  0.,  2.]),
 array([0. , 0.2, 0.4, 0.6, 0.8, 1. , 1.2, 1.4, 1.6, 1.8, 2. ]),
 <a list of 10 Patch objects>)
```



In [28]:
```
svd = TruncatedSVD(300)
train.svd = svd.fit_transform(train.vecs.toarray())
test.svd = svd.transform(test.vecs.toarray())
train.svd.shape, test.svd.shape
num_neighs = 30
metric = 'cosine'
nbrs_svd = NearestNeighbors(n_neighbors=num_neighs, algorithm='brute', metric=
metric).fit(train.svd)
distances, indices = nbrs_svd.kneighbors(test.svd[idx].reshape(1, -1))
plt.hist([train.target_names[nidx] for nidx in train.target[indices][0]])
```

Out[28]:
```
(array([29.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  1.]),
 array([0. , 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1. ]),
 <a list of 10 Patch objects>)
```



Word Embeddings word2vec

Dry run on predefined corpus

In [29]:
```
import gensim.utils
import gensim.downloader as api
```

In [ ]:

In [41]:
```
corpus = api.load('text8')
from gensim.models.word2vec import Word2Vec
model = Word2Vec(corpus, workers=4)
```

In [31]:
```
model.most_similar(['tree'])
```

```
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:1: DeprecationWa
rning: Call to deprecated `most_similar` (Method will be removed in 4.0.0, us
e self.wv.most_similar() instead).
  """Entry point for launching an IPython kernel.
```

Out[31]:
```
[('trees', 0.7119052410125732),
 ('bark', 0.6826577186584473),
 ('leaf', 0.6634393334388733),
 ('avl', 0.6235753297805786),
 ('flower', 0.5902007818222046),
 ('cactus', 0.5811855792999268),
 ('vine', 0.5737411975860596),
 ('pond', 0.5704379081726074),
 ('fruit', 0.562924861907959),
 ('cave', 0.5566340088844299)]
```

In [32]:
```
model.most_similar(['orange'])
```

```
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:1: DeprecationWa
rning: Call to deprecated `most_similar` (Method will be removed in 4.0.0, us
e self.wv.most_similar() instead).
  """Entry point for launching an IPython kernel.
```

Out[32]:
```
[('purple', 0.7383239269256592),
 ('oak', 0.7314910888671875),
 ('green', 0.7022104263305664),
 ('abalone', 0.7001031637191772),
 ('violet', 0.6877890825271606),
 ('yellow', 0.6834245324134827),
 ('deer', 0.6809104084968567),
 ('lemon', 0.677520215511322),
 ('emerald', 0.6682151556015015),
 ('haliotis', 0.6674543619155884)]
```

In [33]:
```
model.wv.most_similar(['london'])
```

Out[33]:
```
[('edinburgh', 0.6696679592132568),
 ('glasgow', 0.6654093265533447),
 ('dublin', 0.646933913230896),
 ('birmingham', 0.6339825391769409),
 ('sydney', 0.6060541868209839),
 ('adelaide', 0.6014052629470825),
 ('croydon', 0.592279314994812),
 ('bristol', 0.5892186164855957),
 ('brighton', 0.5865652561187744),
 ('manchester', 0.5860022902488708)]
```

In [34]:
```
model.wv.most_similar_cosmul(positive=['moscow', 'britain'], negative=['londo
n'], topn=5)
```

Out[34]:
```
[('ussr', 1.1546087265014648),
 ('russia', 1.132383108139038),
 ('communist', 1.077573299407959),
 ('stalin', 1.0695266723632812),
 ('macedonia', 1.055525779724121)]
```

In [35]:
```
model.wv.most_similar_cosmul(positive=['woman', 'king'], negative=['man'], top
n=5)
```

Out[35]:
```
[('queen', 0.9075999855995178),
 ('prince', 0.8859954476356506),
 ('empress', 0.8776204586029053),
 ('elizabeth', 0.8706726431846619),
 ('princess', 0.8689386248588562)]
```

In [36]:
```
len(model.wv.vocab)
```

Out[36]:
```
71290
```

In [37]:
```
model.wv['tree']
```

Out[37]:
```
array([ 1.8194003 , -0.03140939, -0.76792717,  0.15536192, -0.0543808 ,
        0.4773099 , -2.1318462 ,  1.0703834 ,  0.31561863, -0.2300938 ,
       -0.41157365,  1.1670665 , -1.5022365 , -1.9328866 , -1.5848061 ,
       -1.3641012 ,  0.59409904,  0.5191469 ,  2.1342175 , -0.8934795 ,
        1.7323943 , -0.52698463,  0.20303586, -2.950897  ,  0.81844   ,
        0.08701818, -0.5917917 , -0.19036983,  0.02591787,  0.7013831 ,
        1.4823543 ,  0.46630648,  0.57462627, -1.4043822 ,  0.93447906,
        0.3173104 ,  0.06805022,  0.23485315,  0.0450597 ,  0.5022928 ,
       -2.002514  ,  0.681074  ,  0.00444621,  0.4791323 , -1.9391463 ,
       -0.72053987, -2.3139029 ,  0.84141535,  0.4390795 , -0.40667203,
        1.5821112 ,  1.5734584 , -0.62278277,  1.6947138 , -0.15605332,
        2.859384  , -0.02675925,  1.4755529 , -1.3622793 , -1.2207825 ,
        0.00744333,  2.090161  ,  2.2049117 , -2.7640784 , -0.6433659 ,
        1.448533  , -1.1706208 , -0.39682317,  0.41091767,  1.8102112 ,
       -0.9552676 ,  1.6038991 , -1.9702508 ,  0.5643647 ,  1.1176008 ,
       -0.34473872, -0.40103665,  0.45078373, -0.84757924,  0.06165744,
       -1.7037576 , -1.0288435 ,  1.2605336 ,  1.3959775 ,  2.5115764 ,
        0.6078556 , -0.07042108, -2.1125011 , -2.1535006 , -4.091855  ,
        0.03614442,  0.08748694,  0.25431255, -0.531051  ,  0.3305083 ,
        1.0490803 ,  0.38892156, -0.46642476, -0.31434762,  2.1363664 ],
      dtype=float32)
```

In [38]:
```
model.wv['trees']
```

Out[38]:
```
array([ 1.0189222e+00,  7.9554632e-02, -6.0501516e-01,  5.9185016e-01,
        6.6371578e-01,  2.2022939e+00, -2.8559816e+00,  4.4745666e-01,
       -4.7176522e-01,  5.6677323e-02,  6.8564034e-01,  7.5502366e-01,
       -7.3846376e-01, -9.6300793e-01, -1.1109030e+00, -4.5961681e-01,
       -5.0786442e-01,  8.4369099e-01, -4.6890199e-02, -3.4123632e-01,
        1.9126745e+00, -7.3239160e-01,  2.9640761e-01, -3.2761962e+00,
       -4.3844256e-01,  4.1356033e-01, -1.1456252e+00, -1.0244526e+00,
       -7.0757699e-01,  1.5749131e+00,  2.2971642e+00,  9.7959346e-01,
        5.6917810e-01, -8.4210479e-01,  9.1993636e-01,  1.2584240e+00,
       -5.1677662e-01, -3.1778172e-01,  1.0731202e+00, -4.7414306e-01,
       -2.5105581e+00,  3.4343758e-01, -4.0530407e-01,  7.3076916e-01,
        4.6984982e-01, -1.6986617e+00, -7.9747075e-01, -1.3560939e-01,
        5.2228427e-01, -1.1169345e+00,  5.4382777e-01,  1.9080751e+00,
       -1.4663329e+00,  8.8011378e-01,  5.2432621e-01,  2.4689090e+00,
        1.7081742e+00,  6.3637239e-01, -8.1371248e-01, -1.1857074e+00,
       -3.3868071e-01,  2.6547318e+00,  3.0826327e-01, -9.7516859e-01,
        4.2919400e-03, -4.8736230e-01, -8.0012125e-01, -1.2261596e+00,
       -9.1042048e-01,  9.1772825e-01,  2.4298659e-01,  1.0019107e+00,
       -1.3594028e+00,  3.9731252e-01,  1.4146743e+00, -1.2329504e+00,
       -8.8289303e-01,  1.2673075e-01, -2.2330526e-03, -5.6272495e-01,
       -9.6041322e-01,  7.4050374e-02,  1.0127692e+00, -9.9577218e-02,
        2.3951402e+00,  2.3706295e-01,  5.9946418e-02, -2.8143948e-01,
       -1.5022514e+00, -2.0191653e+00, -1.9052912e+00, -1.1181871e+00,
        1.0521566e+00, -4.4718556e-04,  7.7778721e-01, -5.6626928e-01,
        1.8338600e+00, -7.0287591e-01, -1.4424713e+00,  1.8139629e+00],
      dtype=float32)
```

In [39]:
```
#del model
```

Q3. Find an interesting mathematical relationship between words.

In [ ]:
```
# **Optional Q4. Find the performance of the KNN with the out of the box word2vec model**
```

In [43]:
```
model.wv.most_similar_cosmul(positive=['girl', 'captain'], negative=['boy'], topn=5)
```

Out[43]:
```
[('sailor', 0.9275133609771729),
 ('ben', 0.8939493894577026),
 ('pirate', 0.8938014507293701),
 ('murderer', 0.8872220516204834),
 ('maid', 0.8839499354362488)]
```

In [45]:
```
model.wv.most_similar_cosmul(positive=['egg'], negative=['baby'], topn=5)
```

Out[45]:
```
[('organic', 1.667665958404541),
 ('tertiary', 1.6629494428634644),
 ('localized', 1.6534677743911743),
 ('sub', 1.6524354219436646),
 ('administrative', 1.6519345045089722)]
```

In [46]:
```
model.wv.most_similar_cosmul(positive=['bird'], negative=['fish'], topn=5)
```

Out[46]:
```
[('manifesto', 1.4515728950500488),
 ('incarnation', 1.443411111831665),
 ('cluetrain', 1.440385103225708),
 ('inaugural', 1.4384026527404785),
 ('lecture', 1.4215797185897827)]
```

In [48]:
```
model.wv.most_similar_cosmul(positive=['night','place'], negative=['day'], topn=5)
```

Out[48]:
```
[('room', 0.9000268578529358),
 ('sight', 0.8756493926048279),
 ('shots', 0.8712520599365234),
 ('mound', 0.8586012125015259),
 ('groom', 0.8552159667015076)]
```

In [50]:
```
model.wv.most_similar_cosmul(positive=['sex','gay'], negative=['straight'], topn=5)
```

Out[50]:
```
[('transsexual', 1.657067060470581),
 ('homosexual', 1.616520881652832),
 ('homosexuality', 1.6150213479995728),
 ('lesbian', 1.5763429403305054),
 ('bisexual', 1.5623657703399658)]
```

Training on newsgroup data

In [ ]:
```
train.toks = []
for s in train.data:
  train.toks.append(gensim.utils.simple_preprocess(s))
test.toks = []
for s in test.data:
  test.toks.append(gensim.utils.simple_preprocess(s))
```

In [ ]:
```
from gensim.models.word2vec import Word2Vec
model = Word2Vec(train.toks, size=100, window=5, min_count=5, workers=4)
word_vectors = model.wv
del model
```

In [ ]:
```
word_vectors.most_similar_cosmul(['puck']) #
```

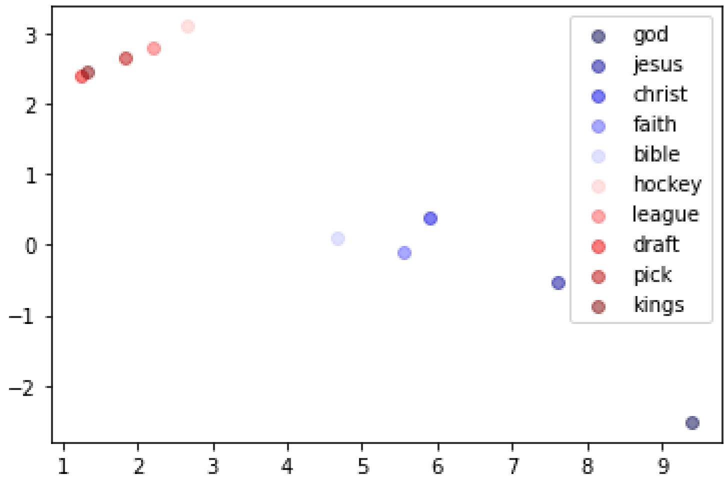
Out[ ]:
```
[('left', 0.9859283566474915),
 ('night', 0.9800158739089966),
 ('sharks', 0.9798518419265747),
 ('head', 0.9796933531761169),
 ('came', 0.9793776869773865),
 ('past', 0.9742642641067505),
 ('flyers', 0.9733327627182007),
 ('ice', 0.9732024073600769),
 ('shot', 0.9725582599639893),
 ('lead', 0.9722954034805298)]
```

In [ ]:
```
words = np.array(['god', 'jesus', 'christ', 'faith', 'bible', 'hockey', 'leagu
e', 'draft', 'pick', 'kings'])
factors = TruncatedSVD(2).fit_transform(word_vectors[words])
# do_plot(, words)
factors.shape
```

Out[ ]:
```
(10, 2)
```

In [ ]:
```
do_plot(factors, words)
```

```
*c* argument looks like a single numeric RGB or RGBA sequence, which should b
e avoided as value-mapping will have precedence in case its length matches wi
th *x* & *y*.  Please use the *color* keyword-argument or provide a 2-D array
with a single row if you intend to specify the same RGB or RGBA value for all
points.
*c* argument looks like a single numeric RGB or RGBA sequence, which should b
e avoided as value-mapping will have precedence in case its length matches wi
th *x* & *y*.  Please use the *color* keyword-argument or provide a 2-D array
with a single row if you intend to specify the same RGB or RGBA value for all
points.
*c* argument looks like a single numeric RGB or RGBA sequence, which should b
e avoided as value-mapping will have precedence in case its length matches wi
th *x* & *y*.  Please use the *color* keyword-argument or provide a 2-D array
with a single row if you intend to specify the same RGB or RGBA value for all
points.
*c* argument looks like a single numeric RGB or RGBA sequence, which should b
e avoided as value-mapping will have precedence in case its length matches wi
th *x* & *y*.  Please use the *color* keyword-argument or provide a 2-D array
with a single row if you intend to specify the same RGB or RGBA value for all
points.
*c* argument looks like a single numeric RGB or RGBA sequence, which should b
e avoided as value-mapping will have precedence in case its length matches wi
th *x* & *y*.  Please use the *color* keyword-argument or provide a 2-D array
with a single row if you intend to specify the same RGB or RGBA value for all
points.
*c* argument looks like a single numeric RGB or RGBA sequence, which should b
e avoided as value-mapping will have precedence in case its length matches wi
th *x* & *y*.  Please use the *color* keyword-argument or provide a 2-D array
with a single row if you intend to specify the same RGB or RGBA value for all
points.
*c* argument looks like a single numeric RGB or RGBA sequence, which should b
e avoided as value-mapping will have precedence in case its length matches wi
th *x* & *y*.  Please use the *color* keyword-argument or provide a 2-D array
with a single row if you intend to specify the same RGB or RGBA value for all
points.
*c* argument looks like a single numeric RGB or RGBA sequence, which should b
e avoided as value-mapping will have precedence in case its length matches wi
th *x* & *y*.  Please use the *color* keyword-argument or provide a 2-D array
with a single row if you intend to specify the same RGB or RGBA value for all
points.
*c* argument looks like a single numeric RGB or RGBA sequence, which should b
e avoided as value-mapping will have precedence in case its length matches wi
th *x* & *y*.  Please use the *color* keyword-argument or provide a 2-D array
with a single row if you intend to specify the same RGB or RGBA value for all
points.
*c* argument looks like a single numeric RGB or RGBA sequence, which should b
e avoided as value-mapping will have precedence in case its length matches wi
th *x* & *y*.  Please use the *color* keyword-argument or provide a 2-D array
with a single row if you intend to specify the same RGB or RGBA value for all
points.
```



In [ ]:
```
train.w2v = np.zeros((len(train.data), word_vectors['good'].shape[0]))
idx = 0
for s in train.toks:
  ws = []
  for w in s:
    if w in word_vectors:
      ws.append(w)
  if len(ws) is not 0:
    train.w2v[idx] = np.mean(word_vectors[ws], axis=0)
  idx += 1
test.w2v = np.zeros((len(test.data), word_vectors['good'].shape[0]))
idx = 0
for s in test.toks:
  ws = []
  for w in s:
    if w in word_vectors:
      ws.append(w)
  if len(ws) is not 0:
    test.w2v[idx] = np.mean(word_vectors[ws], axis=0)
  idx += 1
```

Nearest Neighbors

In [ ]:
```
# with factors first
num_neighs = 10
metric = 'cosine'
nbrs_vecs = NearestNeighbors(n_neighbors=num_neighs, algorithm='brute', metric
=metric).fit(train.vecs)
nbrs_svd = NearestNeighbors(n_neighbors=num_neighs, algorithm='brute', metric=
metric).fit(train.svd)
nbrs_w2v = NearestNeighbors(n_neighbors=num_neighs, algorithm='brute', metric=
metric).fit(train.w2v)
```

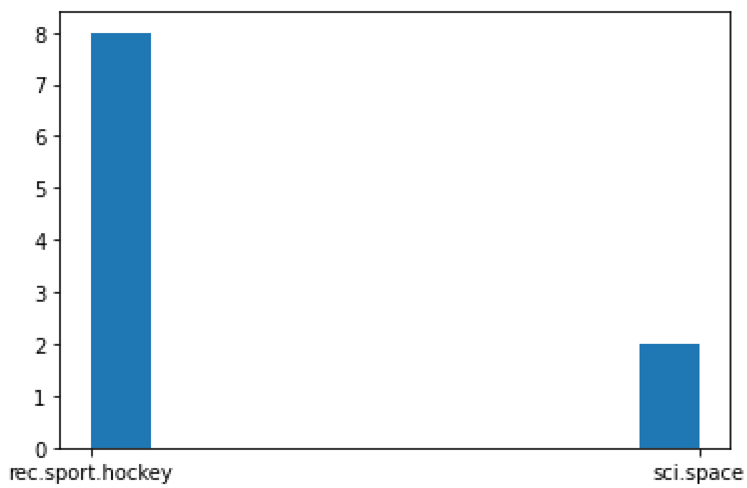
In [ ]:
```
idx = 30
inst = test.data[idx]
test.target_names[test.target[idx]]
pd.DataFrame.from_dict({'category':[test.target_names[test.target[idx]]], 'email':[inst]})
```

Out[ ]:

	category	email
0	rec.sport.hockey	\n\nOK. I know I look pretty desperate on this bboard. I think I have posted\n3 or 4 messages already on the issue of NHL telecats over the last few weeks.\nBut, hey. I am pretty desperate. What I am interested is not just a\nsportsbar with multiple screens so thast I can watch the game on one\nof those silent screens. Are there any hockey oriented bars in this area.\nOr does some Patrick division or Adams division fan have a satellite dish?\nI don't mind paying an admission fee, if necessary.\n

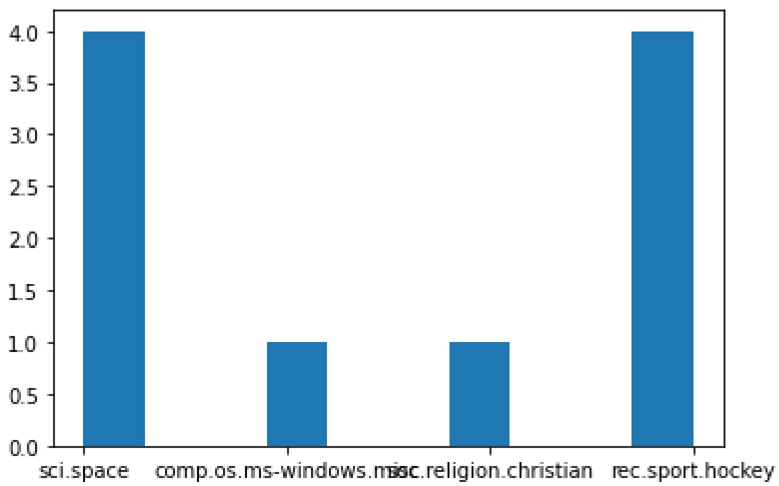
In [ ]:
```
distances, indices = nbrs_vecs.kneighbors(test.vecs[idx])
plt.hist([train.target_names[nidx] for nidx in train.target[indices][0]])
```

Out[ ]:
```
(array([8., 0., 0., 0., 0., 0., 0., 0., 0., 2.]),
 array([0. , 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1. ]),
 <a list of 10 Patch objects>)
```



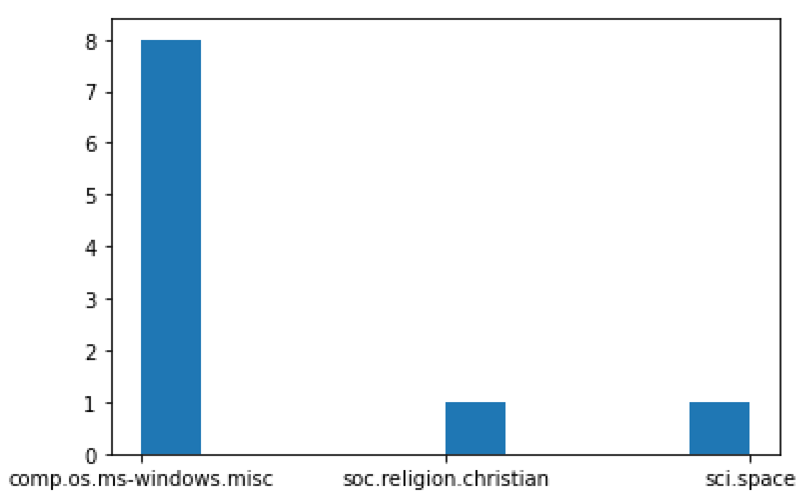
In [ ]:
```
distances, indices = nbrs_svd.kneighbors(test.svd[idx].reshape(1, -1))
plt.hist([train.target_names[nidx] for nidx in train.target[indices][0]])
```

Out[ ]:
```
(array([4., 0., 0., 1., 0., 0., 1., 0., 0., 4.]),
 array([0. , 0.3, 0.6, 0.9, 1.2, 1.5, 1.8, 2.1, 2.4, 2.7, 3. ]),
 <a list of 10 Patch objects>)
```



In [ ]:
```
distances, indices = nbrs_w2v.kneighbors(test.w2v[idx].reshape(1, -1))
plt.hist([train.target_names[nidx] for nidx in train.target[indices][0]])
```

Out[ ]:
```
(array([8., 0., 0., 0., 0., 1., 0., 0., 0., 1.]),
 array([0. , 0.2, 0.4, 0.6, 0.8, 1. , 1.2, 1.4, 1.6, 1.8, 2. ]),
 <a list of 10 Patch objects>)
```



In [ ]:
```
nbrs_vecs_classifier = KNeighborsClassifier(num_neighs).fit(train.vecs, train.
target)
nbrs_svd_classifier = KNeighborsClassifier(num_neighs).fit(train.svd, train.ta
rget)
nbrs_w2v_classifier = KNeighborsClassifier(num_neighs).fit(train.w2v, train.ta
rget)
```

In [ ]:
```
print('vecs', accuracy_score(test.target, nbrs_vecs_classifier.predict(test.ve
cs)))
print('svd', accuracy_score(test.target, nbrs_svd_classifier.predict(test.svd
)))
print('w2v', accuracy_score(test.target, nbrs_w2v_classifier.predict(test.w2v
)))
```

```
vecs 0.22267829656233967
svd 0.5684966649563878
w2v 0.5515649050795279
```

In [ ]: