Kernel Methods Assignment 5

Kunal Ghosh - 546247

Question: 1

1. Let us ~~do the initial cluster assignments by~~ pick the initial set of clusters.

let us say at the t^{th} iteration. the cost is

$$\text{Cost}(t) = \frac{1}{N} \sum_{i=1}^{N} \| x_i - \mu_{C_i}^t \|_2^2$$

Now let us update the cluster centers so cluster centers change from $\mu_{C_i}^t$ to $\mu_{C_i}^{t+1}$. $= \frac{1}{|C_i^t|} \sum_{x_i \in C_i^t} \| x_i \|_2^2$.

for any given cluster $\mu_{C_i}^{t+1}$ would be the point which is closest to all the points in the cluster so, if $\mu_{C_i}^t \neq \mu_{C_i}^{t+1}$ then

$$\sum_{x_i \in C_i^t} \| x_i - \mu_{C_i}^t \|_2^2 \quad > \quad \sum_{x_i \in C_i^{t+1}} \| x_i - \mu_{C_i}^{t+1} \|_2^2$$

and. this would be true for all clusters.

so at this stage $\text{Cost}(t) \geq \text{Cost}(t_{\text{new mean}})$.

by the virtue of $\mu_{C_i}^t$ moving to a $\mu_{C_i}^{t+1}$ which is known to be at the shortest distance to all points in cluster C_i

Now based. on the new centers we could have a cluster assignment step. again such that data points. move from cluster C_K to C_L if

$$\| x_i - \mu_{C_L}^{t+1} \|_2^2 < \| x_i - \mu_{C_K}^{t+1} \|^2$$

for all the other points. the distances to their respective centers remain constant. So we can break up the cost into two parts, for points which moved & those which didn't.

$$\underbrace{\text{Cost}(t_{\text{new mean}}^{\text{not moved}})}_{\text{for datapoints which didn't switch centers}} + \underbrace{\text{Cost}(t_{\text{new mean}}^{\text{moved}})}_{\text{for datapoints which switched centers.}}$$

$Cost(t+1^{moved}) < Cost(t^{moved}_{new\ mean})$. This is because The cost is just the sum of distances of datapoints to their corresponding centers and the new distances are shorter (after shifting to the new clusters). ∴ the overall cost $Cost(t_{new\ mean}) > Cost(t+1)$.

∴ At each step of the K-mean clustering algorithm the cost we are trying to optimize decreases.

- So we observe that, if there is a change in the clustering, then the overall cost decreases.
- We know that there are only K^N different cluster assignments (K = # of clusters, N = # of datapoints).
- Therefore, since at each iteration if the cluster assignment changes & we have a finite number of assignments then the algorithm must stop.
- When the algorithm stops, that would be because it could not find a cluster assignment with lower cost. So the algorithm converges to a local optimum. However there is no way to check if the local optimum is indeed the global optimum because typically in a real world application. the true cluster centers are un-known.

Q2 a). $f(x) = \sum_i \alpha_i K(x_i, x)$. for a new datapoint x.

We know that $K(x_i, x)$ can be written as. $\phi(x_i) P^T P \phi(x)$.

Where. P is the projection matrix $\Sigma_K^{1/2} U^T$

$$\therefore \quad f(x) = \sum_i \alpha_i \phi(x_i) P^T P \phi(x)$$

$$= \alpha^T X^T P^T P \phi(x). \qquad X = U \Sigma V^T$$

$$= \alpha^T V \Sigma U^T (U \Sigma_K^{1/2})(\Sigma_K^{1/2} U^T) \phi(x).$$

$$= \alpha^T V \Sigma \underbrace{U^T U}_{I} \Sigma_K^{1/2} \Sigma_K^{1/2} U^T \phi(x).$$

$$= \alpha^T V \Sigma \Sigma_K U^T \phi(x).$$

because Σ_K has zeros in the diagonal.

$$\Sigma \Sigma_K = \Sigma_K^2$$

$$= \alpha^T V \Sigma_K^2 U^T \phi(x). \quad \square.$$

b). $f(x) = \alpha^T V \Sigma_K^2 U^T \phi(x)$ assumed given. also given is $x^T \phi(x) = t$

$$= \alpha^T V \Sigma_K \Sigma_K U^T \phi(x) \qquad \cancel{\Sigma_K^T = \Sigma_K}$$

$$= \alpha^T V \Sigma_K^* \Sigma_K U^T \phi(x). =$$

$$= \alpha^T V \Sigma_K^* I \Sigma_K U^T \phi(x).$$

$V^T V = I$ $\Rightarrow$

$$\alpha^T V \Sigma_K^* \underbrace{V^T V}_{} \Sigma_K U^T \phi(x)$$

$$\underbrace{\qquad\qquad}_{X^T}$$

$$= \alpha^T V \Sigma_K^* V^T \underbrace{X^T \phi(x)}_{t}.$$

$$= \alpha^T V \Sigma_K^* V^T t \quad \square$$

Solution to Question 5

		Latent Semantic Kernel					
C	**Generalized vector space**	**K = 1**	**K = 50**	**K = 500**	**K = 1000**	**TF-IDF**	**Bag of Words**
0.01	0.508	0.531	0.767	0.825	0.809	0.8	0.809
0.1	0.508	0.531	0.762	0.796	0.806	0.8	0.806
1	0.508	0.531	0.761	0.783	0.806	0.8	0.806
10	0.508	0.531	0.76	0.783	0.806	0.8	0.806
100	0.508	0.534	0.761	0.783	0.806	0.8	0.806

Table 1: Test set accuracy (higher is better) using different kernels and C values (used by the SVM classifier)

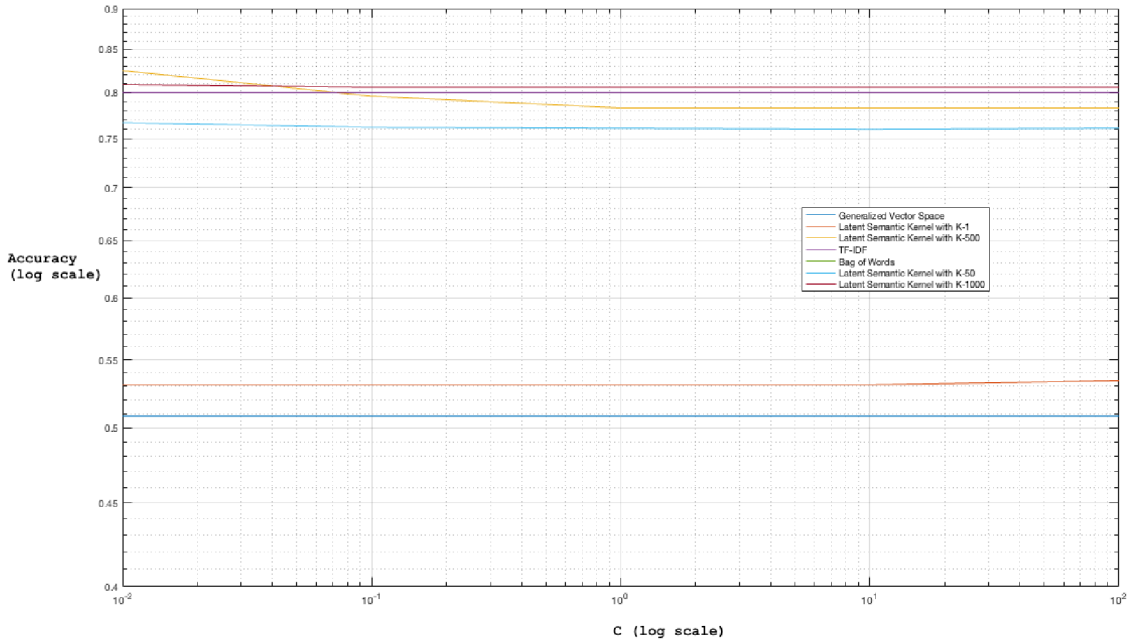

Illustration 1: Visualizes the Accuracy of the same set of Kernels using with an SVM classifier as in Table 1 above. Plotted over different values of C. Axes are in log scale.

Description of the results obtained in Table 1. The above table was created by training SVM using different Kernel Matrices (As seen in individual columns) and each such model was trained using different C values (box constraint in the SVM formulation) and these were recorded along the rows of the matrix.

The values in the table represent the accuracy (maximum possible value is 1 corresponding to 100% test accuracy) of the Support Vector Machine (SVM) classifier on the test set. We notice that the Latent Semantic Kernel using K = 500 (projection using eigenvectors corresponding to largest 500 eigenvalues) obtained the best result (highlighted in green in the above table) and the Generalized vector space kernel performed the worst.

What is the impact on the results of using a term relevance weighting or a term proximity matrix compared to the simple bag-of-words kernel ?

Term relevance weighing corresponds to the TF-IDF kernel in the table above. The term proximity matrix is used in the Generalized Vector Space Kernel in the table above.

It is quite clear that the Term relevance weighing performs almost as well as the Bag-of-Words representation. However using just the Term Proximity matrix (which corresponds to using the Generalized Vector Space) performs much poorer.

Compare the results between the generalized vector space kernel and the latent semantic kernel. In my experiments the generalized vector space kernel seemed to have performed the worst, the performance was similar to using Latent Semantic Kernel with only 1 eigenvector (corresponding to the largest eigenvalue) but slightly worse.

The performance of the Generalized kernel remained same (with quite a low accuracy) for different values of C. However, the Latent Semantic kernel (LSK) had the highest accuracy of around 82.5% for K = 500 and for C = 0.01. For the same K the performance dropped for larger C values.

Solution to Question 3 and 4 on next page

```
import pdb
import pprint
from collections import OrderedDict
import numpy as np
import scipy as sp
from scipy import sparse
from sklearn import svm

def bag_of_words_frm_str(string,delim=" "):
    bag_of_words_str = {}
    for word in string.strip().split(delim):
        try:
            bag_of_words_str[word] += 1
        except KeyError:
            bag_of_words_str[word] = 1

    return bag_of_words_str

def get_bag_of_words(filename):

    bag_of_words = []
    words = {}

    with open(filename,"r") as f:
        data = f.readlines()
        for line in data:
            bow_line = bag_of_words_frm_str(line)
            bag_of_words.append(bow_line)
            words.update(bow_line)

    return bag_of_words, set(words) # creates a set of keys implicitly

def populate_dt_matrix(doc_term_mat, documents, common_word_set):
    common_word_set_indexed = dict(zip(common_word_set, range(len(common_word_set))))
    for idx, doc in enumerate(documents):
        for term,count in doc.items():
            try:
                doc_term_mat[idx,common_word_set_indexed[term]] = count
            except KeyError:
                print("Key {} in Doc but not in common_term_set !!".format(term))

if __name__ == "__main__":
    # get bag of words from train and test files
    Xtest_bow, Xtest_words = get_bag_of_words("../data/text_test.txt")
    Xtrain_bow, Xtrain_words = get_bag_of_words("../data/text_train.txt")

    y_test = np.loadtxt("../data/y_test.txt")
    y_train = np.loadtxt("../data/y_train.txt")

    # get intersection of set of words from test and train docs
    common_word_set = Xtrain_words.union(Xtest_words)
    num_terms = len(common_word_set)

    # create the document-term matrix
    Xtest_dt  = np.zeros((len(Xtest_bow ), num_terms))
    Xtrain_dt = np.zeros((len(Xtrain_bow), num_terms))

    populate_dt_matrix(Xtest_dt, Xtest_bow, common_word_set)
    populate_dt_matrix(Xtrain_dt, Xtrain_bow, common_word_set)

    _term_1_doc = np.vstack((Xtest_dt, Xtrain_dt))

    # Identify terms which occur more than once
    occur_gt1_term = _term_1_doc.astype(np.bool).sum(axis=0) > 1

    # keep terms which occur more than once
    Xtest_dt  = Xtest_dt[:, occur_gt1_term]
    Xtrain_dt = Xtrain_dt[:, occur_gt1_term]

    kernels = {}
    kernels['Bag of Words'] = {'train': np.dot(Xtrain_dt, Xtrain_dt.T),
                     'train_test': np.dot(Xtrain_dt, Xtest_dt.T)
                    }
    print('BOW Kernel Done..')
    # Document Frequency of terms
    nt = Xtrain_dt.astype(np.bool).sum(axis=0)
    # Number of documents
    idf = np.log(1.0 + Xtrain_dt.shape[0]/(nt+10**-5))

    Xtest_tfidf = Xtest_dt * idf
```

```
    Xtrain_tfidf = Xtrain_dt * idf

    kernels['TF-IDF'] = {'train': np.dot(Xtrain_tfidf, Xtrain_tfidf.T),
                         'train_test': np.dot(Xtrain_tfidf, Xtest_tfidf.T)}
    print('TFIDF Kernel Done..')

    Cov_train = (1.0/Xtrain_dt.shape[0]) * np.dot(Xtrain_dt.T, Xtrain_dt)
    Cov_test  = (1.0/Xtest_dt.shape[0]) * np.dot(Xtest_dt.T, Xtest_dt)

    kernels['Generalized Vector Space'] = {'train': np.dot(np.dot(Xtrain_dt,Cov_train),
Xtrain_dt.T),
                     'train_test': np.dot(np.dot(Xtrain_dt, Cov_test), Xtest_dt.T)}
    print('Vector space kernel Done..')

    U,s,V = np.linalg.svd(Xtrain_dt.T) # V is the Eigen Value of covariance matrix
    # for k in [1, 2, 5, 10]:
    for k in [1, 50, 500, 1000]:
        uNew = U[:,:k]
        uuT = np.dot(uNew, uNew.T)
        kernels['Latent Semantic Kernel with K-{}'.format(k)] = {'train': np.dot(np.dot
(Xtrain_dt,uuT), Xtrain_dt.T),
                         'train_test': np.dot(np.dot(Xtrain_dt, uuT), Xtest_dt.T)}
        print('Latent Semantic kernel for k = {} Done..'.format(k))

    C = 10.0**np.arange(-2,3)
    results = {}
    for kernel,val in kernels.items():
        Ktrain, Ktrain_test = val['train'], val['train_test']
        print("=== {} ===".format(kernel))
        for c in C:
            model = svm.SVC(C=c, kernel='precomputed')
            model.fit(Ktrain, y_train)
            y_predict = model.predict(Ktrain_test.T)
            results[(kernel, c)] = np.mean(y_predict==y_test)
            print("C = {} Accuracy = {}".format(c, results[(kernel, c)]))

    pdb.set_trace()
```