Lab 4: Support Vector Machine Text Classification

Copyright (C) Edward Hunter
edward.a.hunter@gmail.com

1. Introduction

In this lab we customize our template program to perform support vector machine (SVM) text classification and evaluate the results. In addition to the data used in Labs 2 and 3, we introduce the Congressional Record dataset "senate," and look at how effective SVMs are at detecting political ideology from Senatorial speech. This experiment closely follows some results in the readings (Diermeier et al., 2011).

2. Create the SVM program

Clear our or remove the soc290/models/ and soc290/reports/ directories so that prior data does not clutter with our new results. From your /soc290 project directory make a copy of the template_supervised.py file called svm.py and open the copy in your editor. The docstrings and globals edits are similar to before

```
#!/usr/bin/env python

"""
@package css
@file css/svm.py
@author Edward Hunter
@author Your Name Here
@brief Support vector machine supervised learning and evaluation methods.
"""

[COPYRIGHT AND LICENSE OMITTED]

# Import common modules and utilities.
from common import *

# Define method and models available.
METHOD = 'SVM'
MODELS = ('linear','poly','rbf')
```

The docstring prelude now carries the correct file and package name. As before add your name as an @author. We have also defined the METHOD global as the string 'SVM' and the MODELS variable as a tuple with the strings 'linear', 'poly', and 'rbf', for linear, polynomial and radial-basis SVM kernels described in lecture.

3. Editing the train() Function

To customize our program for SVMs, we have to edit the how we create the model as we did in the prior labs. We also have to add additional optional parameters to the program that are used by SVM models. The first to-do in train() deals with constructing the vectorizer and clf objects for SVMs. The full code to add to train() looks like this

```
############################################################
# Create feature extractor, classifier.
############################################################
vectorizer = TfidfVectorizer(stop_words='english',sublinear_tf=True,
                min_df=df_min, max_df=df_max)

# SVM specific parameters.
svm_c = kwargs.get('svm_c', 1.0)
svm_tol = kwargs.get('svm_tol', 1e-3)
svm_max_iter = kwargs.get('svm_max_iter', -1)
svm_degree = kwargs.get('svm_degree', 3)
svm_gamma = kwargs.get('svm_gamma', 0.0)
svm_coef0 = kwargs.get('svm_coef0', 0.0)
svm_gs = kwargs.get('svm_gs',False)
svm_top = kwargs.get('svm_top',0)

# Grid search support for SVMs.
if svm_gs:
    C_range = 10.0 ** np.arange(-1, 3)
    degree_range = [2, 3, 4, 5]
    coef0_range = [0, 1, 10]
    gamma_range = 10.0 ** np.arange(-2, 3)
    param_grid = dict(
        kernel=[model],
        tol=[svm_tol],
        max_iter=[svm_max_iter],
        C=C_range,
        )
    if model == 'linear':
        pass

    elif model == 'poly':
        param_grid['degree']=degree_range
        param_grid['coef0']=coef0_range
        param_grid['gamma']=gamma_range

    elif model == 'rbf':
        param_grid['gamma']=gamma_range

    svr = SVC()
    clf = GridSearchCV(svr, param_grid=param_grid, n_jobs=-1, verbose=5)

# Specified parameter SVMs.
else:
    if model == 'linear':
            clf = SVC(kernel='linear', C=svm_c, tol=svm_tol,
                    max_iter=svm_max_iter)
    elif model == 'poly':
            clf = SVC(kernel='poly', C=svm_c, tol=svm_tol,
                    max_iter=svm_max_iter, degree=svm_degree,
                    gamma=svm_gamma, coef0=svm_coef0)
    elif model == 'rbf':
            clf = SVC(kernel='rbf', C=svm_c, tol=svm_tol,
                    max_iter=svm_max_iter, gamma=svm_gamma)
if fselector:
    fselector.__normalize = True
############################################################
```

This code is a bit more complicated than the other labs because we need to add support for additional options required by SVMs as well as support for parameter grid search. To make the discussion easier, we'll discuss the above edits in three steps.

1. First, at the top of the block, we add a TFIDF vectorizer just like in prior labs. Then in the "SVM specific parameters" section are eight lines of code that extract the SVM options passed to train() by the entry point into local variables named **svm_xxx**. The kwargs variable is a parameter passed to train to collect any method specific options required, and was not used in prior labs that didn't require this. Edit the file to add these lines.
2. An if-block called "Grid search support for SVMs" follows, keyed on the value of option svm_gs. This option signals running the grid search function, and will be True if –svm_gs is specified on the command line. Grid search involves retraining SVMs over a grid of possible parameters and testing for the best combination. This is especially useful with models like SVMs that have parameters that may not be easy to select intuitively. In the above code, the **xxx_range** parameters set up a default set of test points for each parameter we search over (c, degree, coef0, and gamma). The notation degree_range = [2,3,4,5] tells us that we will search over polynomials of degrees 2-5 for polynomial kernel SVMs, and similarly for coef0_range. The notation 10.0 ** np.arange(-1,3) is convenient shorthand for something similar. Here a quick experiment with the interpreter can show us exactly what it is doing

```
(css)$ python
Python 2.7.6 (default, Jan  2 2014, 14:32:56)
[GCC 4.2.1 Compatible Apple LLVM 4.2 (clang-425.0.28)] on darwin
Type "help", "copyright", "credits" or "license" for more information.
>>> import numpy as np
>>> np.arange(-1,3)
array([-1,  0,  1,  2])
>>> 10.0 ** np.arange(-1,3)
array([   0.1,    1. ,   10. ,  100. ])
```

   So we see that arange() is a numpy function (imported as np) that expands to an array of integers [-1, 0, 1, 2] (the end value 3 is not inclusive). Taking the power of 10.0 to this array ("**" is the exponent operator) returns the element-wise powers of ten: [0.1, 1.0, 10.0, 100.0]. So these are the values of gamma that will be searched simultaneously with the other parameter ranges. The same occurs for the C parameter range.

   A dictionary called **param_grid** is then created and populated with parameters that are needed for all SVM kernel types: C, max_iter, tol and kernel. In this initial dictionary, only C contains a range, the other parameters are single values as they are not searched over and common defaults are used for all cases.

   The nested if-elif-block that follows is keyed on model, which is the kernel type. So, in each of these we add to param_grid those parameter ranges that are required for that type of kernel. For example, the linear kernel does not require any additional parameters so we **pass**, and the RBF kernel requires the gamma parameter range defined above and so is added under elif model == 'rbf'.

In the final part of this section, a scikit SVM classifier (called **SVC** in scikit) is created and assigned to **svr**. svr and the param_grid dictionary are passed in to create a scikit **GridSearchCV** object **clf**, the same variable name used to create non-grid search SVMs, which we discuss next. Since the GridSearchCV and SVC objects expose the same training functions, we assign one or the other to the variable clf and use the same training code for either. This is another instance of code reuse though polymorphism and so we don't have to edit the training code provided by the template.

There are two additional parameters passed to GridSearchCV: **n_jobs=-1** and **verbose=5**. The n_jobs parameter tells grid search to probe the computer and attempt to parallelize the search with multiple processes running simultaneously if the system is capable. This can speed up the search significantly. verbose tells grid search to output status on the search while it runs so we can monitor its progress and make sure we don't have any bugs.

3. The else-block "Specified parameter SVMs" constructs SVM classifiers directly when svm_gs is absent, with specified kernel type and parameters. A nested if-elif statement keys on model and constructs **clf** with parameters appropriate for the three different types of kernels. Finally, if dimension reduction was specified with the –d option, then the fselector is defined and its attribute __normalize (note: two undercores!) is set to True. This is done regardless of grid search or kernel type as all use TFIDF feature vectors that must be normalized to unit length.

The table below briefly reviews what each option is used for. In our experiments we will only be concerned with c, gamma, gs and top.

Option	**Description**	**Kernels**	**Default**
svm_c	Penalty for misclassified data	All	1.0
svm_tol	Tolerance stopping criteria	All	0.001
svm_max_iter	Max number of iterations	All	None
svm_degree	Degree	Polynomial	3
svm_gamma	Gamma	Polynomial, RBF	1.0
svm_coef0	Constant term	Polynomial	0.0
svm_gs	Perform grid search	All	False
svm_top	Display top features	2-class linear	0

In the second to-do in train(), we set the grid search and top features output variables. Grid search output is enabled when the svm_gs option is specified. Top feature output is enabled when svm_top is specified with a 2-class linear kernel.

```
############################################################
# Grid search and top feature output for SVMs.
############################################################
grid_search_output = opts.svm_gs
top_features_output = model == 'linear' and svm_top>0 \
    and len(data['target_names'])==2
############################################################
```

Grid search output prints the optimal parameters found. Top feature output gives us a way to see which terms are most important in separating categories like liberal vs. conservative ideological speech. The output code itself is provided by the template and does not need to be added.

You may wonder why we allow the svm_top option only for linear SVMs. The reason is that for the linear SVM, the feature importance is relatively easy to determine. This is not true of nonlinear kernels like polynomial and RBF. It is an open research problem to develop a technique to calculate feature importance for these types of SVMs.

4. Editing the Entry Point

In the above edits to train() we used a number of new options that are not part of the default template. Here we add them to the entry point option parsing so that they are made available to train() through the command line. First we add these lines to the first to-do to tell the OptionParser object about the new options

```
############################################################
# Add method specific options.
############################################################
p.add_option('--svm_c', action='store', dest='svm_c', type='float',
             help='SVM penalty term, default=1.0.')
p.add_option('--svm_tol', action='store', dest='svm_tol', type='float',
             help='SVM tolerance, default =1e-3.')
p.add_option('--svm_max_iter', action='store', dest='svm_max_iter',
             type='int', help='SVM max iterations, default=no max.')
p.add_option('--svm_degree', action='store', dest='svm_degree', type='int',
             help='SVM degree (poly), default=3.')
p.add_option('--svm_gamma', action='store', dest='svm_gamma', type='float',
             help='SVM gamma (poly, rbf), default=1/n_features.')
p.add_option('--svm_coef0', action='store', dest='svm_coef0', type='float',
             help='SVM independent coefficient (poly), default=0.0.')
p.add_option('--svm_gs',action='store_true',dest='svm_gs',
             help='Grid Search for SVM')
p.add_option('--svm_top',action='store',dest='svm_top',type='int',
             help='Show top N features for SVM (2-class linear only), default=0.')
p.set_defaults(fappend=None, dim=None, confusion=None, overwrite=False,
               svm_c=1.0, svm_tol=1e-3, svm_max_iter=-1, svm_degree=3,
               svm_gamma=0.0, svm_coef0=0.0,svm_gs=False,svm_top=0,
               df_min=1.0, df_max=1.0)
############################################################
```

Note the set_defaults call that adds default values for options not specified. This should replace the one that was already defined in the template so make sure you remove the default one above.

The second to-do in the entry point follows the argument parsing code. In this to-do we extract any SVM options passed to the program from the command line and add them to the method_kwargs dictionary that is passed along for use by train()

```
    ############################################################
    # Extract method specific options.
    ############################################################
    method_kwargs ['svm_c'] = opts.svm_c
    method_kwargs ['svm_tol'] = opts.svm_tol
    method_kwargs ['svm_max_iter'] = opts.svm_max_iter
    method_kwargs ['svm_degree'] = opts.svm_degree
    method_kwargs ['svm_gamma'] = opts.svm_gamma
    method_kwargs ['svm_coef0'] = opts.svm_coef0
    method_kwargs ['svm_gs']=opts.svm_gs
    method_kwargs ['svm_top']=opts.svm_top
    ############################################################
```

With that, the file is now ready to use to train and test SVMs.

5. Collecting SVM Performance Data on 20 Newsgroups and Reuters Data

```
(css)$ python svm.py --help
Usage: svm.py [options] model dataset
       model = ('linear', 'poly', 'rbf')
       dataset = ('20news', '20news4', '20news5', 'reuters21578-10', 'senate')

Train and evaluate support vector supervised classifiers.

Options:
  -h, --help            show this help message and exit
  -f FAPPEND, --fappend=FAPPEND
                        File name appendix string.
  -d DIM, --dim=DIM     Reduced feature dimension integer.
  -c CONFUSION, --confusion=CONFUSION
                        Save confusion image. Options: linear, log
  -o, --overwrite       Overwrite existing files.
  --df_min=DF_MIN       Minimum document frequency proportion (default=None).
  --df_max=DF_MAX       Maximum document frequency proportion (default=1.0).
  --svm_c=SVM_C         SVM penalty term, default=1.0.
  --svm_tol=SVM_TOL     SVM tolerance, default =1e-3.
  --svm_max_iter=SVM_MAX_ITER
                        SVM max iterations, default=no max.
  --svm_degree=SVM_DEGREE
                        SVM degree (poly), default=3.
  --svm_gamma=SVM_GAMMA
                        SVM gamma (poly, rbf), default=1/n_features.
  --svm_coef0=SVM_COEF0
                        SVM independent coefficient (poly), default=0.0.
  --svm_gs              Grid Search for SVM
  --svm_top=SVM_TOP     Show top N features for SVM (2-class linear only),
                        default=0.
```

First we ask the svm program for help to verify our new options are now available, shown above. If everything looks good, we run the SVM algorithm for 20news with a default linear kernel

```
(css)$ python svm.py -c linear linear 20news
Loading: ./data/20news.pkz
[. . . DATA SUMMARY OUTPUT OMITTED . . .]
Extracting text features...
Extracted in 2.622787 seconds.
Feature dimension: 101323
Feature density: 0.000659
Training classifier...
Trained in 77.247033 seconds.
Classifier written to file ./models/SVM_linear_20news_clf
Feature extractor written to file ./models/SVM_linear_20news_vec
Read classifer from file: ./models/SVM_linear_20news_clf
Read feature extractor from file: ./models/SVM_linear_20news_vec
[. . . REPORT AND CONFUSION MATRIX OMITTED . . .]
```

The –c linear argument specifies a linear confusion matrix image. The second linear argument specifies the type of SVM kernel, as described in the help text.

Run the same SVM on 20news5 and 20news4 and the reuters21578-10. The command lines are (output omitted)

```
(css)$ python svm.py -c linear linear 20news5
(css)$ python svm.py -c linear linear 20news4
(css)$ python svm.py -c linear linear reuters21578-10
```

As usual, the report files and confusion images will now appear in soc290/reports.

6. Collecting SVM Performance Data on Congressional Record Data

Next we will collect results on the ability of SVM to predict liberal vs. conservative ideology as reflected in roll-call voting, directly from Senatorial speech data. The senate dataset uses all Senatorial speeches from Congresses 107-109 as training data to predict voting behavior in Congresses 110-112. Liberal and conservative labels for training and testing data are derived from the DW-Nominate scores discussed in the first lecture (see http://voteview.com/dwnomin.htm for more details). We begin by trying default linear and RBF SVMs (output omitted)

```
(css)$ python svm.py linear senate
(css)$ python svm.py rbf senate
```

Note that when running SVMs on the senate data, we don't bother with generating a confusion matrix as there are only 2 classes and it is easy to inspect the numerical matrix in the report directly.

If you inspect the reports for linear and RBF kernels on the senate data you see something surprising: although the RBF kernel theoretically has more discriminating power, its performance is weaker. This is because the default RBF parameters svm_c and svm_gamma are not optimized for the senate dataset. But how can we select them? The best way is to run a grid search over the parameter space to see which parameters optimize performance (and why we went to the trouble to add that code above). To do this we rerun the RBF SVM with the --svm_gs option. In addition, we will use the –f option to ensure the resulting model filenames are distinct from the default RBF models we just generated.

Run the following SVM RBF grid search to find the optimal parameters and performance. This will take longer since we are training the SVM over many parameter settings during the search. The command line and output looks like this

```
(css)$ python svm.py --svm_gs -f grid_search rbf senate
Loading: ./data/senate.pkz
Class Names:
    0                   liberal  train size:       75, test size:       75
    1              conservative  train size:       75, test size:       74
                        Totals:  train size:      150, test size:      149
Extracting text features...
Extracted in 28.963929 seconds.
Feature dimension: 81034
Feature density: 0.110266
Training classifier...
Fitting 3 folds for each of 20 candidates, totalling 60 fits
[GridSearchCV] kernel=rbf, C=0.1, max_iter=-1, gamma=0.01, tol=0.001 ..........
[GridSearchCV] kernel=rbf, C=0.1, max_iter=-1, gamma=0.01, tol=0.001 ..........
[GridSearchCV] kernel=rbf, C=0.1, max_iter=-1, gamma=0.01, tol=0.001 ..........
[GridSearchCV] kernel=rbf, C=0.1, max_iter=-1, gamma=0.1, tol=0.001 ...........
[GridSearchCV] kernel=rbf, C=0.1, max_iter=-1, gamma=0.1, tol=0.001 ...........
[GridSearchCV] kernel=rbf, C=0.1, max_iter=-1, gamma=0.1, tol=0.001 ...........
[GridSearchCV] kernel=rbf, C=0.1, max_iter=-1, gamma=1.0, tol=0.001 ...........
[GridSearchCV] kernel=rbf, C=0.1, max_iter=-1, gamma=1.0, tol=0.001 ...........
[GridSearchCV]  kernel=rbf, C=0.1, max_iter=-1, gamma=0.01, tol=0.001, score=0.840000 -   1.7s
[GridSearchCV]  kernel=rbf, C=0.1, max_iter=-1, gamma=0.01, tol=0.001, score=0.900000 -   1.8s
[Parallel(n_jobs=-1)]: Done   1 jobs       | elapsed:    1.9s
[GridSearchCV]  kernel=rbf, C=0.1, max_iter=-1, gamma=0.01, tol=0.001, score=0.900000 -   1.7s

[. . . MUCH MORE GRID SEARCH PROGRESS OUTPUT . . .]

[Parallel(n_jobs=-1)]: Done  59 out of  60 | elapsed:   16.5s remaining:    0.3s
[GridSearchCV]  kernel=rbf, C=100.0, max_iter=-1, gamma=100.0, tol=0.001, score=0.520000 -   1.6s
[GridSearchCV]  kernel=rbf, C=100.0, max_iter=-1, gamma=100.0, tol=0.001, score=0.500000 -   1.4s
[Parallel(n_jobs=-1)]: Done  60 out of  60 | elapsed:   16.6s finished
Trained in 18.398150 seconds.
Best score: 0.973333333333
Optimal parameters:
kernel=rbf
C=10.0
max_iter=-1
tol=0.001
gamma=0.1
Classifier written to file ./models/SVM_rbf_senate_grid_search_clf
Feature extractor written to file ./models/SVM_rbf_senate_grid_search_vec
Read classifer from file: ./models/SVM_rbf_senate_grid_search_clf
Read feature extractor from file: ./models/SVM_rbf_senate_grid_search_vec
[. . . REPORT AND CONFUSION MATRIX OUTPUT OMITTED . . .]
```

Does the performance look better now? Note that when complete, the optimal RBF parameters are reported: C=10.0 and gamma=0.1. The senate dataset only has 150 documents to train over, so the time required is not too bad, about 18s. For 20news and other cases where there are 1000s or more documents, grid search can take minutes or hours to complete, and will also be influenced by your chip architecture (multicore CPUs will benefit from parallelization).

To see which bag-of-words features are most predictive of liberal-vs-conservative ideology, we rerun the default linear SVM with the –svm_top option and we specify –o to overwrite the existing linear SVM model that was previously saved to disk and force the SVM to retrain. The command line is (output omitted)

```
(css)$ python svm.py –svm_top=20 –o linear senate
```

Clip out the top features output and save it for your report. You should now see the following report data in your soc290/reports/ directory

```
(css)$ ls -aoF reports/
total 336
drwxr-xr-x  13 edward    442 Mar 24 19:18 ./
drwxr-xr-x  25 edward    850 Mar 24 18:40 ../
-rw-r--r--   1 edward  28860 Mar 24 18:47 SVM_linear_20news4_confusion.png
-rw-r--r--   1 edward    696 Mar 24 18:47 SVM_linear_20news4_report.txt
-rw-r--r--   1 edward  30098 Mar 24 18:47 SVM_linear_20news5_confusion.png
-rw-r--r--   1 edward    778 Mar 24 18:47 SVM_linear_20news5_report.txt
-rw-r--r--   1 edward  40921 Mar 24 18:40 SVM_linear_20news_confusion.png
-rw-r--r--   1 edward   3723 Mar 24 18:40 SVM_linear_20news_report.txt
-rw-r--r--   1 edward  35031 Mar 24 18:49 SVM_linear_reuters21578-10_confusion.png
-rw-r--r--   1 edward   1418 Mar 24 18:49 SVM_linear_reuters21578-10_report.txt
-rw-r--r--   1 edward    485 Mar 24 19:14 SVM_linear_senate_report.txt
-rw-r--r--   1 edward    494 Mar 24 19:07 SVM_rbf_senate_grid_search_report.txt
-rw-r--r--   1 edward    482 Mar 24 19:18 SVM_rbf_senate_report.txt
```

7. Analyzing the Data and Preparing the Lab Report

Collect your report data and confusion images (no confusion images for senate data) into these data results pages.

```
1. Data 1: Linear SVM 20news, Default Parameter C=1
2. Data 2: Linear SVM 20news5 and 20news4, Default Parameter C=1
3. Data 3: Linear SVM reuters21578-10, Default Parameter C=1
4. Data 4: Linear SVM senate, Default Parameter C=1, Top Features
5. Data 5: RBF SVM senate, Default Parameters C=1, Gamma=1/Dim
           RBF SVM senate, Optimal Parameters C=10.0, Gamma=0.1
```

Results Analysis

Summary: For each of the datasets (20news, 20news5, 20news4, reuters21578-10, senate) and test conditions (default linear SVM, default RBF SVM, optimal RBF svm):

1. How would you characterize the performance of overall in each of the tests? Justify your argument with specific references to the average precision and recall and their distributions across document categories in each case.

2. Summarize the top features found in the liberal-vs-conservative senate test. Although we would have to examine these terms in context to know for sure, which terms may be suggestive of true ideological differences and which may not be or are difficult to interpret. If there is interest in class, we can show a quick way to spot check the terms to understand their context to help in these cases using the UNIX grep program with raw speech data. How might we use traditional content analysis methods to validate the predictions the SVM is making are representative of ideological differences?

Comparisons:

1. How do the default Linear SVMs compare against the Naïve Bayes results from Lab 2 on the 20 newsgroups and Reuters datasets? If the data are true representative samples of the underlying data probabilities what does this suggest? Do you think it is likely that these data are accurate samples of the term probabilities given the categories and sample size?
2. We know from lecture that SVM have certain performance generalization guarantees. Describe in your own words what these are. If the data samples are not precisely representative of the underlying probabilities, what differences could we expect from Naïve Bayes and SVM models if they were presented lots of new testing data?
3. Compare the performance of the default linear and optimal RBF SVMs on the senate dataset. Although the RBF has more powerful discrimination ability, is it needed here? Explain why you think this is so.