CSC411: Project #3 Bonus

Due on Saturday, March 24, 2018

Tyler Gamvrelis and Izaak Niksan

Part 1)

a) For this bonus, support vector machines were investigated in the context of classifying fake news. First, as a reference on how well SVMs performed relative to the classifiers which were already tested, the same datasets as before were used. The words contained in the training, validation, and test sets (70%,15%, and 15% respectively) were used to create input vectors, such that a 1 corresponded to a word being in the headline and a 0 corresponded to a word not being present in the headline. This is the same input format used for logistic regression. The nu support vector classifier SVM was built using sklearn; first, the default parameters were used and yielded comparable performances to what was seen by the logistic regression classifier. 94% of the training set and 76% of the validation set were correctly classified. Then one parameter of the sklearn SVM implementation, specifically nu, was varied. nu was varied as it directly controlled the fraction of margin errors, and thus how much the model was forced to fit the training data. It was iteratively increased from 0.1 to 1 in increments of 0.1; however, it was found that values above 0.7 prevented the classifier from being built. Each of the possible kernels for the nu support vector classifier was tried in this manner, but the radial basis function kernel yielded the best results. In this way, the kernel was selected using cross-validation.

By testing the performance on the validation set, it was determined that the ideal value of nu was 0.5 (which was the same default value as before). The worst value of nu was 0.1, yielding performances of 74% and 66% on the training and validation sets respectively. Listing 1 contains part of the code used for tuning the parameters.

Listing 1: Optimizing the sklearn SVM implementation

```

print('Tuning Performance by varying nu: \n')
max_train_perf=0
max_train_nu=-1
max_val_perf=0
max_val_nu=-1
for i in range(1,8): #since nu of 0.8 to 1.0 are 'infeasible'
    nu= i/10

    clf = svm.nuSVC(nu=nu, kernel='sigmoid', degree=10, gamma='auto', coef0=0.0,
                    shrinking=True, probability=True, tol=0.001,
                    cache_size=200, class_weight=None, verbose=False,
                    max_iter=-1, decision_function_shape='ovr',
                    random_state=None
                )
    clf.fit(x_train, y_train)

    #performance on training set:
    predict=clf.predict(x_train)
    train_performance=0
    for i in range(x_train.shape[0]):
        if y_train[i]==predict[i]:
            train_performance=train_performance+1
    train_performance=100*train_performance/x_train.shape[0]
    print('Training performance for nu =',nu,'is ',train_performance)
    if train_performance>max_train_perf:
        max_train_perf=train_performance
        max_train_nu=nu

    #performance on validation set:
    predict=clf.predict(x_val)
```

```
        val_performance=0
        for i in range(x_val.shape[0]):
            if y_val[i]==predict[i]:
                val_performance=val_performance+1
        val_performance=100*val_performance/x_val.shape[0]
        print('Validation performance for nu =',nu,'is ',val_performance)
        if val_performance>max_val_perf:
            max_val_perf=val_performance
            max_val_nu=nu
```

Once this optimal value for nu was determined, the value for gamma was varied. For the radial basis function, gamma is inversely proportional to the variance of the gaussian used to compute the similarity measure. Table 2 displays the training and validation accuracy obtained for 4 settings of gamma. Note that when `gamma = 'auto'` was used, gamma was tuned automatically to 2.07×10^{-4}.

Table 1: Performance for 4 gamma values, with all other parameters held constant

gamma	**Training %**	**Validation %**	**# support vectors**
'auto'	93.7	76.9	1438
1×10^{-6}	69.5	65.3	1144
1	99.8	61.0	2237
100	99.96	60.6	2279

Thus, gamma was left set to auto. In summary, the optimization parameters used to train the classifier were:

- nu = 0.5
- radial basis function kernel
- gamma = 2.07×10^{-4}
- decision function shape: one-versus-rest
- probability estimates: enabled
- everything else: default

Next, as an interesting further test on this classifier's performance, the top 5 fake headlines from *ABC News* were inputted into the SVM and evaluated. These headlines (available at http://abcnews4.com/news/nation-world/photo-gallery-top-fake-news-headlines-of-the-week) were:

"Trump just ended Obama's vacation scam and sent him a bill you have to see to believe"

"NASA confirms earth will experience 15 days of darkness in November 2017"

"Furious Chelsea Clinton thrown to the floor and handcuffed after senator Lindsay scandal linked to Clinton foundation"

"Cannibals arrested in Florida claim eating human flesh cures diabetes and depression"

"Democrat Maxine Waters has shown up to only 10% of congressional meetings for 35 years"

3 of the 5 of these were correctly classified, which is not an ideal performance. However, it should be

noted that the provided dataset seemed to mainly contain information regarding politics, while the *ABC* headlines were not restricted in this way. Furthermore, if one were to do further investigation into classifying fake news, it is advised that a more comprehensive dataset comprised of headlines from other sources is used to train the classifier.

b) A support vector machine is a supervised learning algorithm that performs classification by finding the hyperplane that separates the classes with maximum margin. The *support vectors* are the data points constraining the width of the margin and thus defining the hyperplane, i.e. they are the training examples closest to the hyperplane. The training procedure selects the best possible hyperplane (and thus support vectors) given the training data and the optimization parameters. Often, data might not be linearly separable, so the kernel trick is employed. The kernel trick is a technique by which the problem of linearly separating the data is solved in a higher-dimensional implicit feature space. This space is called implicit because a given data point x only ever exists in this space in the context of computing the kernel $\phi(x)$, which takes the form of an inner product. For this project, the radial basis function kernel was used, which theoretically can map into an infinite-dimensional feature space. To complete the description of how this method works, new data would simply be classified based on which "side" of the hyperplane they were on, using the same hypothesis function that was employed during training, which in general can be expressed as in eq. 1.

$$h(x) = sgn(\sum_{i}^{n} y_i \alpha_i K(x_i, x) + b) \tag{1}$$

where:

- sgn is the signum function
- y_i is the class label for the i^{th} training example $\{1, -1\}^n$
- α_i is the weight of the i^{th} training example, indicating how much its restriction of the hyperplane should be taken into account when classifying
- $K(,) = \phi(x_i) \cdot \phi(x)$ is the kernel function, which was selected to be a radial basis function
- b is the bias, which is an intercept for the hyperplane

With this in mind, the support vectors were observed using `clf.support_vectors_`, where `clf` was the trained classifier instance. Interestingly, as seen in Table 2, there were 1438 support vectors in the most optimal model found, which means 63% of the training data was necessary to constrain the hyperplane.

Furthermore, the product $y_i\alpha_i$ could be viewed for each support vector by using the python line `clf.support_vectors_`. The mean of this quantity was 0 and the standard deviation was 786.3. Some more statistics are presented in Table **??**. Note that these were rounded to the nearest whole number where necessary. Furthermore, it is emphasized that the support vectors with positive coefficients corresponded to words that were deemed important for classifying news as real, and the support vectors with negative coefficients corresponded to words that were deemed important for classifying news as fake.

Table 2: Descriptive statistics for support vector coefficients

	Count	**Mean**	**Median**	**Std. dev**	**Max**	**Min**
Positive coefficients	756	703	930	276	930	5.4
Negative coefficients	682	-779	-930	253	-1.4	-930

Due to the fact that the coefficients for many support vectors saturated to the maximum and minimum values, it was not entirely feasible to interpret any particular words as being the most indicative of real or fake news. However, 5 words appearing in “real news” support vectors with the most positive coefficients and 5 words appearing in “fake news” support vectors with the most negative coefficients (where “real” implies a positive coefficient multiplying the support vector and “fake” implies a negative sign multiplying the support vector) were determined to be as follows:

1. Real news:
   - mccain
   - shows
   - trump
   - says
   - jerusalem
2. Fake news:
   - trump
   - ignores
   - gain
   - in
   - the

Listing 2 presents the code used to determine the statistics and words.

Listing 2: Statistics and Important Words

```
import numpy as np
print("Number of support vectors: ")
print(clf.support_vectors_.shape)

c = clf._dual_coef_
v = clf.support_vectors_
print("Mean of dual coefficients: ")
print(np.mean(c))
print("Std. dev of dual coefficients: ")
print(np.sqrt(np.var(c)))

cpos = list()
cneg = list()
cposPositional = c.copy()
cnegPositional = c.copy()
for i in range(c.shape[1]):
    # Don't need to worry about coefficients equal to 0 as they would hold no
    # weight and thus aren't important at all
    if c[0, i] > 0:
        cpos.append(c[0, i])
        cposPositional[0, i] = c[0, i]
        cnegPositional[0, i] = 0
    else:
        cneg.append(c[0, i])
        cnegPositional[0, i] = c[0, i]
        cposPositional[0, i] = 0
cpos = np.array(cpos)
cneg = np.array(cneg)

print("Count of positive coefficients: ", np.count_nonzero(cpos))
print("Mean of positive coefficients: ", np.mean(cpos))
print("Median of positive coefficients: ", np.median(cpos))
print("Std. dev of positive coefficients: ", np.sqrt(np.var(cpos)))
print("Max of positive coefficients: ", np.max(cpos))
print("Min of positive coefficients: ", np.min(cpos))
print("Example of 5 words important for real news")
cpp = cposPositional.copy()
words = list()
for k in range(5):
    idx = np.argmax(cpp)
    cpp[0, idx] = 0
    a = 0
    while(True):
        a = np.argmax(v[[idx],:])
        w = all_words[a]
        if w in words:
            v[[idx], a] = 0
        else:
            words.append(w)
            break
    print(all_words[a])

print("Count of negative coefficients: ", np.count_nonzero(cneg))
print("Mean of negative coefficients: ", np.mean(cneg))
print("Median of negative coefficients: ", np.median(cneg))
print("Std. dev of negative coefficients: ", np.sqrt(np.var(cneg)))
print("Max of negative coefficients: ", np.max(cneg))
print("Min of negative coefficients: ", np.min(cneg))
cnp = cnegPositional.copy()
words = list()
for k in range(5):
    idx = np.argmin(cnp)
    cnp[0, idx] = 0
    a = 0
    while(True):
        a = np.argmax(v[[idx],:])
        w = all_words[a]
        if w in words:
            v[[idx], a] = 0
        else:
            words.append(w)
            break
    print(all_words[a])
```

Part 2)
Omitted