STA208: Homework 3

Prof. James Sharpnack

Due 4/25 in class

In the following, show all your work. Feel free to do all the analytical questions first and then include the code and output second, but the different parts and which question that you are answering should be clearly marked. Code should be as modular as possible, points will be deducted for code that is not reusable (i.e. not broken into general purpose functions), and in the case of gratuitous hard coding.

1. This question is about your final project.
   (a) List the members of your group.
   (b) What is the title of your project?
   (c) Give a brief description of what you intend to do.
2. Consider the following linear classification loss function:

$$\ell(\boldsymbol{\beta}; y_i, \mathbf{x}_i) = (1 - y_i \mathbf{x}_i^\top \boldsymbol{\beta})_+^2$$

where $z_+ = \max\{0, z\}$. Consider the modified support vector machine program,

$$\min_{\boldsymbol{\beta} \in \mathbb{R}^p} \frac{1}{n} \sum_{i=1}^n \ell(\boldsymbol{\beta}; y_i, \mathbf{x}_i) + \lambda \|\boldsymbol{\beta}\|_2^2. \tag{1}$$

   (a) Compute the gradient of the objective in (1) and verify that it is continuous.
   (b) Write the pseudocode for gradient descent with exact line search for this gradient (you do not need to write out the bisection algorithm for exact line search, only state what exact line search is.).
   (c) Suppose that we make a transformation $\mathbf{z}_i = \Phi(\mathbf{x}_i)$ and we use this as the design matrix. Rewrite the minimization for (1) with the generalized kernel trick (Hint: make the substitution $\boldsymbol{\beta} = \mathbf{Z}^\top \boldsymbol{\gamma}$ and write it as a function of $\boldsymbol{\gamma}$ and define $\mathbf{K} = \mathbf{Z}\mathbf{Z}^\top$.).

Solution:

(a) The gradient of the loss is

$$\nabla \ell(\beta; y_i, \mathbf{x}_i) = -2 y_i (1 - y_i \mathbf{x}_i^\top \boldsymbol{\beta})_+ \mathbf{x}_i.$$

We can see this by the chain rule when $y_i \mathbf{x}_i^\top \boldsymbol{\beta} < 1$ and otherwise the gradient it clearly 0. Thus, the gradient of the objective is

$$-\frac{2}{n} \sum_{i=1}^n y_i (1 - y_i \mathbf{x}_i^\top \boldsymbol{\beta})_+ \mathbf{x}_i + 2\lambda \boldsymbol{\beta}$$

because the gradient of $\|\boldsymbol{\beta}\|_2^2$ is $2\boldsymbol{\beta}$.

(b) The pseudocode is

1. Initialize $\boldsymbol{\beta} = \mathbf{1}$ for example.

2. Compute the gradient $\mathbf{g} = -\frac{2}{n}\sum_{i=1}^n y_i(1 - y_i\mathbf{x}_i^\top\boldsymbol{\beta})_+\mathbf{x}_i + 2\lambda\boldsymbol{\beta}$.
3. Compute
$$\hat{\eta} = \underset{\eta\geq 0}{\operatorname{argmin}} \frac{1}{n}\sum_{i=1}^n (1 - y_i\mathbf{x}_i^\top(\boldsymbol{\beta} - \eta\mathbf{g}))_+^2 + \lambda\|\boldsymbol{\beta} - \eta\mathbf{g}\|_2^2$$
4. Increment $\boldsymbol{\beta} \leftarrow \boldsymbol{\beta} - \hat{\eta}\mathbf{g}$.
5. Repeat (2)-(4) until some stopping criteria.

(c) Define $\boldsymbol{\beta} = \mathbf{Z}^\top\boldsymbol{\gamma}$ and then the loss is $(1 - y_i\mathbf{k}_i^\top\boldsymbol{\gamma})_+^2$ where $\mathbf{k}_i$ is the ith row of $\mathbf{K}$. Also, $\|\boldsymbol{\beta}\|_2^2 = \boldsymbol{\gamma}^\top\mathbf{K}\boldsymbol{\gamma}$. So, we have that gradient descent is

1. Initialize $\boldsymbol{\gamma} = \mathbf{1}$ for example.
2. Compute the gradient $\mathbf{g} = -\frac{2}{n}\sum_{i=1}^n y_i(1 - y_i\mathbf{k}_i^\top\boldsymbol{\gamma})_+\mathbf{k}_i + 2\lambda\mathbf{K}\boldsymbol{\gamma}$.
3. Compute
$$\hat{\eta} = \underset{\eta\geq 0}{\operatorname{argmin}} \frac{1}{n}\sum_{i=1}^n (1 - y_i\mathbf{k}_i^\top(\boldsymbol{\gamma} - \eta\mathbf{g}))_+^2 + \lambda(\boldsymbol{\gamma} - \eta\mathbf{g})^\top\mathbf{K}(\boldsymbol{\gamma} - \eta\mathbf{g})$$
4. Increment $\boldsymbol{\gamma} \leftarrow \boldsymbol{\gamma} - \hat{\eta}\mathbf{g}$.
5. Repeat (2)-(4) until some stopping criteria.

3. Consider the following way in which we encode logical statements. For $\mathbf{x} \in \{0,1\}^2$, we will encode the "and" and "or" statements: ($x_1 > 0$ and $x_2 > 0$) is the same as ($x_1x_2 > 0$); ($x_1 > 0$ or $x_2 > 0$) is the same as ($x_1 + x_2 > 0$).

(a) Suppose that $y_i = 1$ if and only if ($x_{i,1} = 1$ and $x_{i,2} = 1$) or $x_{i,3} = 1$ and $y_i = 0$ otherwise. Write a polynomial function $f(\mathbf{x}_i)$ such that $f(\mathbf{x}_i) > 0$ if and only if $y_i = 1$.

(b) Define an embedding of $\mathbf{x}$ such that the dataset has a separating hyperplane, and write an equation for the separating hyperplane as a function of $\mathbf{z}_i = \Phi(\mathbf{x}_i)$. Specifically, consider any dataset that is consistent with this rule ($y_i = 1$ iff this rule holds) then your separating hyperplane should separate the positive from negative examples.

(c) From this embedding construct the kernel function $k(\mathbf{x}_i, \mathbf{x}_j)$.

(d) Suppose that $y_i = 1$ if and only if ($x_{i,1} = 1$ or $x_{i,2} = 1$) and ($x_{i,3} = 1$ and $x_{i,4} = 1$) and $y_i = 0$ otherwise. Write a polynomial function $f(\mathbf{x}_i)$ such that $f(\mathbf{x}_i) > 0$ if and only if $y_i = 1$.

(e) Define an embedding of $\mathbf{x}$ such that this new dataset has a separating hyperplane, and write an equation for the separating hyperplane as a function of $\mathbf{z}_i = \Phi(\mathbf{x}_i)$, as you did in (b).

(f) From this embedding construct the kernel function $k(\mathbf{x}_i, \mathbf{x}_j)$.

(g) For simplicity let $p = 3$ (if you did it more generally then great!). Define the degree of a logical statement as the maximum number of variables involved in any monomial after we apply the substitutions defined above. Give an embedding and kernel that separates a dataset where $y_i = 1$ if and only if some logical statements of degree at most 3 is true. For example, a possible statement of degree 3 is $x_1 = 1$ and $x_2 = 1$ and $x_3 = 0$, which would be encoded as the polynomial $x_1x_2(1-x_3) = x_1x_2 - x_1x_2x_3$, so $y_i = 1\{x_1x_2 - x_1x_2x_3 > 0\}$ (i.e. the binary response variable is 1 iff the statement is true). You must provide an embedding that separates the positive and negative cases for **any** such logical statement (of degree 3).

Solution:

(a) So we have that for $x_1, x_2 \geq 0$, $x_1 x_2 > 0$ iff $x_1 > 0$ and $x_2 > 0$. Similarly $x_1 + x_2 > 0$ iff $x_1 > 0$ or $x_2 > 0$. So, ($x_{i,1} = 1$ and $x_{i,2} = 1$) or $x_{i,3} = 1$ iff $f(\mathbf{x}_i) = x_{i,1}x_{i,2} + x_{i,3} > 0$.

(b) There are several suitable embeddings, and as long as $f(\mathbf{x}_i)$ can be written as a linear combination of the basis elements then it is correct. Here are some options:

$$\begin{aligned}\Phi(\mathbf{x}_i) &= (x_{i,1}x_{i,2}, x_{i,3}) \\ \Phi(\mathbf{x}_i) &= (x_{i,1}x_{i,2} + x_{i,3}) \\ \Phi(\mathbf{x}_i) &= (x_{i,1}, x_{i,2}, x_{i,3}, x_{i,1}x_{i,2}, x_{i,3}x_{i,1}, x_{i,2}x_{i,3})\end{aligned}$$

Then a separating hyperplane for this embedding is $(1,1)^\top \Phi(\mathbf{x}_i) > \epsilon$ for any $\epsilon > 0$ small enough.

(c) We use $k(\mathbf{x}_i, \mathbf{x}_j) = \Phi(\mathbf{x}_i)^\top \Phi(\mathbf{x}_j)$ which will differ for different embeddings. For the first one above, we have

$$k(\mathbf{x}_i, \mathbf{x}_j) = \Phi(\mathbf{x}_i)^\top \Phi(\mathbf{x}_j) = (x_{i,1}x_{i,2}, x_{i,3})^\top (x_{j,1}x_{j,2}, x_{j,3}) = x_{i,1}x_{i,2}x_{j,1}x_{j,2} + x_{i,3}x_{j,3}$$

(d) Similarly, we can write out the separating function as

$$f(\mathbf{x}_i) = (x_{i,1} + x_{i_2})(x_{i,3}x_{i,4}) = x_{i,1}x_{i,3}x_{i,4} + x_{i,2}x_{i,3}x_{i,4}.$$

(e) An embedding is $\Phi(\mathbf{x}_i) = (x_{i,1}x_{i,3}x_{i,4}, x_{i,2}x_{i,3}x_{i,4})$. Then a separating hyperplane for this embedding is $(1,1)^\top \Phi(\mathbf{x}_i) > \epsilon$ for any $\epsilon > 0$ small enough.

(f) The kernel is $k(\mathbf{x}_i, \mathbf{x}_j) = (x_{i,1}x_{i,3}x_{i,4}, x_{i,2}x_{i,3}x_{i,4})^\top (x_{j,1}x_{j,3}x_{j,4}, x_{j,2}x_{j,3}x_{j,4}) = x_{i,1}x_{i,3}x_{i,4}x_{j,1}x_{j,3}x_{j,4} + x_{i,2}x_{i,3}x_{i,4}x_{j,2}x_{j,3}x_{j,4}$.

(g) An embedding that can separate any statement of degree 3 is

$$\Phi(\mathbf{x}) = (1, x_1, x_2, x_3, x_1x_2, x_1x_3, x_2x_3, x_1x_2x_3).$$

For general p there are $\binom{p}{1}$ monomials of degree 1, $\binom{p}{2}$ monomials of degree 2, and $\binom{p}{3}$ monomials of degree 3.

4. Download the training and test datasets from the website for the Reuters data. The first column is the response variable and the rest are the counts of each term in the dictionary for each document.

(a) Tune a linear SVM with the raw counts as the design matrix and tune with 5-fold cross validation.

(b) Apply the tf-idf transformation and then tune a linear SVM with this as the design matrix, again with 5-fold cross validation.

(c) Fit logistic regression to these and tune it with 5-fold cross validation.

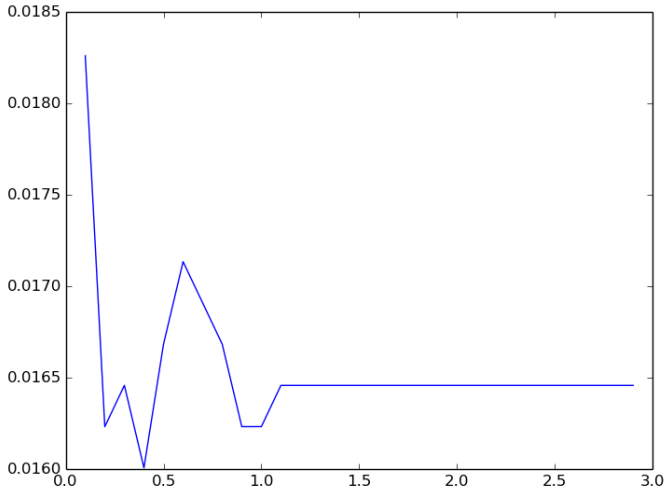
(d) Construct scores for each of these methods for the test set (this should be $\mathbf{x}_i^\top \boldsymbol{\beta}$), and construct the ROC and PR curves for them.

Solution:

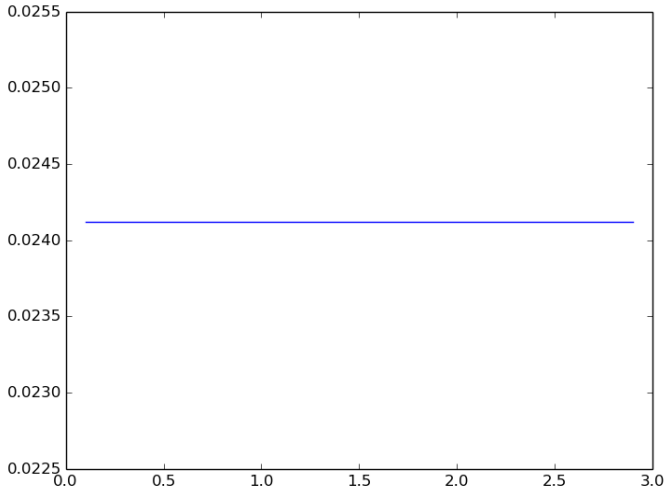
(a)
```
import numpy as np
import csv, os
from scipy import sparse
from sklearn import linear_model, neighbors, preprocessing, cross_validation, svm, metrics
from matplotlib import pyplot as plt

def TFIDF(X,I):
```

```
    n, p = X.shape
    DF = I.T.dot(np.ones(n))
    IDF = np.log(n / DF)
    IDFM = sparse.diags(IDF,0)
    Z = X.dot(IDFM)
    return Z

filename = 'spmats/train_Xsp.csv'
rowa = []

with open(filename, 'rb') as csvfile:
    rtr = csv.reader(csvfile, delimiter=',',quotechar='|', quoting=csv.QUOTE_MINIMAL)
    for row in rtr:
        rowa.append(row)

y = np.loadtxt("spmats/train_y.csv")

m = len(rowa)
Spm = np.array(rowa,dtype="int")
n,p = np.max(Spm[:,0])+1, np.max(Spm[:,1])+1
X = sparse.lil_matrix((n,p),dtype="int")
I = sparse.lil_matrix((n,p),dtype="int")
for i,j,c in Spm:
    X[i,j] = c
    I[i,j] = 1

K = X.dot(X.T).toarray()
kf = cross_validation.KFold(n, n_folds=5)

def test_C(C):
    errors = []
    countsvm = svm.SVC(kernel="precomputed",C = C)
    for train, test in kf:
        Ktr = K[train,:][:,train]
        Kcr = K[train,:][:,test]
        ytr = y[train]
        countsvm.fit(Ktr,ytr)
        yhat = countsvm.predict(Kcr.T)
        errors.append(sum((yhat - y[test])**2.) / len(test))
    return np.mean(errors)

Cs = np.arange(1,30)/10.
CVerr = [test_C(C) for C in Cs]

plt.plot(Cs,CVerr,'-')
plt.show()
```

The following is the cross-validated percent missed as a function of C. (I selected $C = 1$)



(b) I just replaced $X = TFIDF(X, I)$ and then reran the above code. The C is barely changing, so I chose $C = 1$ again. The following is the cross-validated percent missed as a function of C. (I selected $C = 1$)

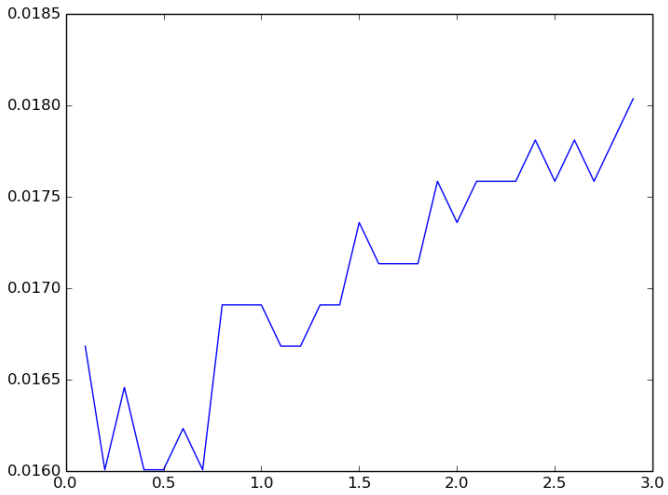


(c) It turns out that scikit-learn's LogisticRegression can accept sparse matrices! This was a surprise for me, but it makes this run very fast.

```
def test_C(C):
    errors = []
    countlog = linear_model.LogisticRegression(penalty='l2', dual=False, tol=0.0001, C=C)
    for train, test in kf:
        Xtr = X[train,:]
        Xte = X[test,:]
        ytr = y[train]
        countlog.fit(Xtr,ytr)
        yhat = countlog.predict(Xte)
```

```
        errors.append(sum((yhat - y[test])**2.) / len(test))
    return np.mean(errors)

Cs = np.arange(1,30)/10.
CVerr = [test_C(C) for C in Cs]
```

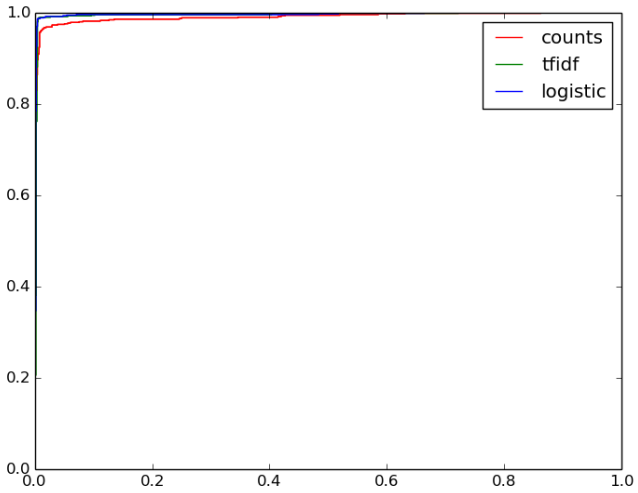
The following is the cross-validated percent missed as a function of C. (I selected $C = 0.5$)



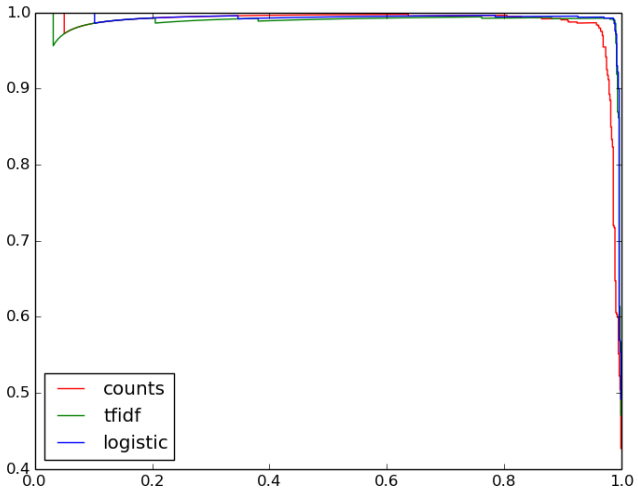
(d) I read the test set in the same way that I constructed the training. Then we can generate the ROC and PR curves.

```
betac = countsvm.coef_[0].T
cs = X.dot(betac)
cs = cs.toarray()
cs.shape = (n,)
betat = tfsvm.coef_[0].T
ct = X.dot(betat)
ct = ct.toarray()
ct.shape = (n,)
betal = tflog.coef_[0].T
cl = X.dot(betal)

fpr, tpr, thresholds = metrics.roc_curve(y, cs, pos_label=1)
plt.plot(fpr,tpr,"r")
fpr, tpr, thresholds = metrics.roc_curve(y, ct, pos_label=1)
plt.plot(fpr,tpr,"g")
fpr, tpr, thresholds = metrics.roc_curve(y, cl, pos_label=1)
plt.plot(fpr,tpr,"b")
plt.legend(["counts","tfidf","logistic"])
plt.show()

prec, rec, thresholds = metrics.precision_recall_curve(y, cs, pos_label=1)
plt.plot(rec,prec,"r")
prec, rec, thresholds = metrics.precision_recall_curve(y, ct, pos_label=1)
plt.plot(rec,prec,"g")
prec, rec, thresholds = metrics.precision_recall_curve(y, cl, pos_label=1)
```

```
plt.plot(rec,prec,"b")
plt.legend(["counts","tfidf","logistic"],loc=3)
plt.show()
```



ROC curve



PR curve
I would select the tfidf for sure, but there is no real difference between the svm and logistic regression.