Computer Science M146, Homework 3

Michael Wu
UID: 404751542

February 27th, 2018

Problem 1

The VC dimension of H is 3. An example of 3 points x such that $x \in R$ that can be shattered are $x_1 = -1$, $x_2 = 0$, and $x_3 = 1$. Then the following table shows how we can shatter these points, with $h \in H$ being the classifier used.

x_1 label	x_2 label	x_3 label	h
0	0	0	$\text{sgn}(-1)$
0	0	1	$\text{sgn}(x - 0.5)$
0	1	0	$\text{sgn}(-x^2 + 0.5)$
0	1	1	$\text{sgn}(x + 0.5)$
1	0	0	$\text{sgn}(-x - 0.5)$
1	0	1	$\text{sgn}(x^2 - 0.5)$
1	1	0	$\text{sgn}(-x + 0.5)$
1	1	1	$\text{sgn}(1)$

For any set $S = \{x_1, x_2, x_3, x_4\}$ of four points where $x_1 < x_2 < x_3 < x_4$, we cannot shatter S if we label $x_1 = 1$, $x_2 = 0$, $x_3 = 1$, $x_4 = 0$. This is because any classifier in H at most splits R into three distinct regions where the quadratic $ax^2 + bx + c$ is above or below zero, and we have 4 distinct regions of classification in our set. So our hypothesis space cannot shatter S because no h can correctly classify this training set.

Problem 2

$$\begin{aligned}
K_\beta(\mathbf{x},\mathbf{z}) &= (1+\beta\mathbf{x}\cdot\mathbf{z})^3 \\
&= (1+\beta(x_1z_1+\ldots+x_Dz_D))^3 \\
&= 1+3\beta\sum_{i=1}^{D}x_iz_i+3\beta^2\left(\sum_{i=1}^{D}x_iz_i\right)^2+\beta^3\left(\sum_{i=1}^{D}x_iz_i\right)^3 \\
&= 1+3\beta\sum_{i=1}^{D}x_iz_i+3\beta^2\sum_{i,j=1}^{D}x_iz_ix_jz_j+\beta^3\sum_{i,j,k=1}^{D}x_iz_ix_jz_jx_kz_k \\
&= \left\langle 1, \sqrt{3\beta}x_i\Big|_{i=1}^{D}, \sqrt{3}\beta x_ix_j\Big|_{i,j=1}^{D}, \beta^{\frac{3}{2}}x_ix_jx_k\Big|_{i,j,k=1}^{D}\right\rangle \\
&\quad\cdot\left\langle 1, \sqrt{3\beta}z_i\Big|_{i=1}^{D}, \sqrt{3}\beta z_iz_j\Big|_{i,j=1}^{D}, \beta^{\frac{3}{2}}z_iz_jz_k\Big|_{i,j,k=1}^{D}\right\rangle
\end{aligned}$$

Thus we have

$$\phi_\beta(\mathbf{x}) = \left\langle 1, \sqrt{3\beta}x_i\Big|_{i=1}^{D}, \sqrt{3}\beta x_ix_j\Big|_{i,j=1}^{D}, \beta^{\frac{3}{2}}x_ix_jx_k\Big|_{i,j,k=1}^{D}\right\rangle$$

The kernel $K(\mathbf{x},\mathbf{z}) = (1+\mathbf{x}\cdot\mathbf{z})^3$ is equivalent to setting $\beta = 1$, and corresponds to the feature map

$$\phi_1(\mathbf{x}) = \left\langle 1, \sqrt{3}x_i\Big|_{i=1}^{D}, \sqrt{3}x_ix_j\Big|_{i,j=1}^{D}, x_ix_jx_k\Big|_{i,j,k=1}^{D}\right\rangle$$

The parameter β scales the features up and down by a constant. It effectively replaces the vector $\mathbf{x}$ by $\sqrt{\beta}\mathbf{x}$ such that $\phi_\beta(\mathbf{x}) = \phi_1(\sqrt{\beta}\mathbf{x})$.

Problem 3

a) We wish to find $\mathbf{w}^* = \langle w_1, w_2\rangle$ such that we minimize $\frac{1}{2}\sqrt{w_1^2+w_2^2}$ subject to $w_1+w_2 \geq 1$ and $-w_1 \geq 1$. Equivalently we would like to find the point closest to the origin such that $w_2 \geq 1-w_1$ and $w_1 \leq -1$. Thus we get

$$\mathbf{w}^* = \langle -1, 2\rangle$$

and the margin is $\gamma = \frac{1}{\sqrt{5}}$.

b) We wish to find $\mathbf{w}^* = \langle w_1, w_2 \rangle$ such that we minimize $\frac{1}{2}\sqrt{w_1^2 + w_2^2}$ subject to $w_1 + w_2 + b \geq 1$ and $-w_1 - b \geq 1$. The classifier changes by making the decision boundary a horizontal line that crosses $\left(0, \frac{1}{2}\right)$, and the margin will increase compared to our previous results. We get

$$\mathbf{w}^* = \langle 0, 2 \rangle$$

and

$$b^* = -1$$

Our margin is $\gamma = \frac{1}{2}$. This makes sense because our two points $\mathbf{x}_1 = (1, 1)$ and $\mathbf{x}_2 = (1, 0)$ have a distance of 1 between them, so our margin is exactly half of this distance. Our solution with offset has a higher margin γ and smaller magnitude of $\mathbf{w}^*$ than our solution without offset.

Problem 4

4.1

a) I created the dictionary using the following code

```
word_list = {}
count = 0
with open(infile, 'rU') as fid :
   for line in fid:
      wordListLine = extract_words(line)
      for word in wordListLine:
         if word not in word_list:
            word_list[word]=count
            count+=1
return word_list
```

b) I extracted the feature vectors using the following code

```
with open(infile, 'rU') as fid :
lineNum=0
for line in fid:
   wordListLine = extract_words(line)
   for word in wordListLine:
      feature_matrix[lineNum,word_list[word]]=1
   lineNum+=1
return feature_matrix
```

c) I split the features and labels into train and test sets using the following code

```
trainX = X[0:560]
trainy = y[0:560]
testX = X[560:630]
testy = y[560:630]
```

d) The feature matrix has the dimensions $(630, 1811)$. The trainX set has dimensions $(560, 1811)$, the trainy set has dimensions $(560, 1)$, the testX set has dimensions $(70, 1811)$, and the testy set has the dimensions $(70, 1)$.

4.2

a) I implemented performance using the following code

```
score = 0
if metric=="accuracy":
   score=metrics.accuracy_score(y_true,y_label)
if metric=="f1-score":
   score=metrics.f1_score(y_true,y_label)
if metric=="auroc":
   score=metrics.roc_auc_score(y_true,y_pred)
return score
```

b) I implemented cv_performance using the following code

```
return cross_val_score(clf, X, y, scoring=metric, cv=kf).mean()
```

and in main I added

```
kf=StratifiedKFold(trainy, 5)
```

It is beneficial to use a stratified K-fold so that the percentage of positive and negative reviews are the same across folds because this ensures that our cross validation gets a good representation of our training data in each fold. Otherwise we may accidentally divide the folds such that one gets a small number of negative or positive reviews, making our classifier performance inaccurate.

c) I implemented select_param_linear with the following code

```
best=0
cBest=0
for c in C_range:
   score=cv_performance(SVC(kernel="linear",C=c),X,y,kf,metric)
   print "C="+str(c)+" score="+str(score)
   if score>best:
      best=score
      cBest=c
return cBest
```

d) My results were as follows

C	Accuracy	F1-score	AUROC
10^{-3}	0.7089	0.8297	0.8105
10^{-2}	0.7107	0.8306	0.8111
10^{-1}	0.8060	0.8755	0.8576
10^{0}	0.8146	0.8749	0.8712
10^{1}	0.8182	0.8766	0.8696
10^{2}	0.8182	0.8766	0.8696
best C	10^{1}	10^{1}	10^{0}

4.3

a) I chose the hyperparameter $c = 10$ and trained my classifier using the following code

```
clf=SVC(kernel="linear", C=10)
clf.fit(trainX, trainy)
```

b) I implemented performance_test using the following code

```
y_pred=clf.decision_function(X)
print "Performance test with metric "+str(metric)+":"
return performance(y, y_pred, metric)
```

c) My results were as follows

Metric	Score
Accuracy	0.7429
F1-score	0.4375
AUROC	0.7454