46-926, Fall 2017: Homework #6

Due 5:20PM, Monday, February 26, 2018.

Homework should be submitted electronically on canvas before the deadline. Please submit your homework as a single file (pdf or html). We recommend using a jupyter notebook to prepare your homework.

Note: For this homework, try to do all problems in python.

Please post any questions on the Piazza discussion board.

Corrections:

- 2/22/18 at 9:25am: Added missing term to the Naive Bayes expressions in problem 2.
- 2/23/18 at 1:10am: Made the corrections outlined in the announcement.

1. *Different forms of SVM.* An important piece of optimization transforming one optimization problem to another equivalent optimization problem. We will consider optimization problems to be equivalent if they give the same solution at their optimum point, or if you can easily transform between their solutions.

   We're going to go through a series of steps transforming the SVM optimization into other equivalent forms. Our end goal will be to see that the SVM can equivalently be rewritten as the minimization of

   $$\sum_{i=1}^{n} \left(1 - y_i(x_i^T\beta + \beta_0)\right)_+ + \frac{\lambda}{2}\|\beta\|_2^2$$

   where $(z)_+ = \max(z, 0)$, the positive part function. This is the most classic form of the SVM, and has an interesting interpretation which we will discuss next class.

   Note: This is a very important and classic style of approach/proof. I'm going to work through much of it with you, since you haven't seen these before. Please read everything carefully anyway and give a lot of attention to what's happening, even though you're not doing it from scratch. I've **Bolded** questions that you need to answer as you go along.

   You will get a chance to check your Form 2-5 of the problem when you reach the end, since Form 5 should exactly match the equation above. You can go back and tweak your choices at the other steps so that everything works out.

We start with

Maximize M (Form 1)

$$\text{Subject to } \sum_{i=1}^{n} \varepsilon_i \leq C,\ \varepsilon_i \geq 0,\ y_i(x_i^T\beta + \beta_0) \geq M(1-\varepsilon),\ \|\beta\|_2 = 1$$

Note that this is exactly how we defined the SVM in class (with an explict intercept).

(a) Consider the inequality $y_i(x_i^T\beta + \beta_0) \geq M(1-\varepsilon_i)$. Since we know $\|\beta\|_2 = 1$, we can rewrite this as

$$y_i(x_i^T\beta + \beta_0) \geq M(1-\varepsilon_i)\|\beta\|_2$$

without changing anything.

Show that if you multiply β_0 and β by a fixed constant b, nothing changes about the meaning of this inequality.

Since you only really care about the direction of β, this means that you can set the length of β to be anything you want without changing the rest of the optimization problem.

Choose a value of $\|\beta\|_2$ (the Euclidean length of β) that lets you eliminate M from your optimization problem. **Rewrite** Form 1 using this substitution, so that M no longer appears in the problem. (Note, you can switch the "Maximize" to a "Minimize", and you can drop the $\|\beta\|_2 = 1$ constraint). *Call your new optimization "Form 2"*

Question: Your M parameter used to carry information about the width of your margin. Where is this information now encoded?

Question: What is the relationship between the β that solves this problem and the β that solved the previous optimization problem?

(b) Now we're going to work with the $\sum_{i=1}^{n} \varepsilon_i \leq C$ term. Suppose that we added a term $D\sum_{i=1}^{n} \varepsilon_i$ to the objective from Form 2, so that your minimization objective looked like

$$(\text{Form 2 objective}) + D\sum_{i=1}^{n} \varepsilon_i$$

and remove the constraint that $\sum_{i=1}^{n} \varepsilon_i < C$.

Write out the new formulation where this substitution has been made, and call it "Form 3".

Argue that for any C in Form 2, there exists a D in Form 3 that gives the same solution to the optimization problem. (Just a sentence or two.)

Question: If I make C bigger in Form 2, does D get bigger or smaller in the equivalent problem of Form 3?

(c) We're getting close! Now comes the cool and tricky part. Think about a set of parameters that solves Form 3. In particular, think about a particular i and ε_i. The "Subject to" piece of Form 3 can be thought of as two constraints on ε_i, if everything else is held fixed.

Rewrite these as two inequality constraints on ε_i, where ε_i appears alone on one side.

Now think about the objective function. At the optimum, you know that ε_i should be as small as possible, subject to the constraints, because of the $\sum_{i=1}^{n} \varepsilon_i$ term in the objective.
From these two pieces, you can figure out exactly what ε_i must be at the optimum!

Write out ε_i as a function of $y_i(x_i^T\beta + \beta_0)$. You will probably want to use the "positive part" function, $(z)_+ = \begin{cases} x & x > 0 \\ 0 & x < 0 \end{cases}$.

Substitute this quantity for ε_i in your objective and write down the new optimization form, called Form 4. After you've made this substitution, you should no longer have any ε_i.

(d) Finally, you should be able to divide through by a constant to get an optimization problem exactly of the form we set out to show. Call this Form 5.

Question: What is λ as a function of D to make the form identical to formula stated at the beginning of the problem?

Question: We know that, when C goes up, the margin width increases and the separating plane becomes potentially more biased but less variable. When C increases, do D (Form 4) and λ (Form 5) each increase or decrease? What happens to the length of β?

2. *Implementing your own Naive Bayes.* One nice thing about Naive Bayes is its flexibility. You can model each variable (conditional on class) with a separate univariate distribution. However, machine learning packages usually only implement a few versions of Naive Bayes, limiting your choices. `sklearn` implements Gaussian, Multinomial, and Bernoulli. In this problem, you'll build your own simple Naive Bayes classifer on a small data set, using a different model for each variable. To do this problem, don't directly use any Naive Bayes packaged functions from python.

   We'll use the simple default data set we've plotted a few times in class. You can download `default_train.csv` and `default_test.csv` from Canvas. Fit models below on the training data and evaluate them on the test data.

   (a) You will be predicting default based on credit balance, income, and whether or not an individual is a student. The first two are continuous, and we'll just model them as Gaussian within each class for simplicity. The last is binary, which we'll model as Bernoulli within class.

   Plot histograms of each of the variables (using the training data).

   Also, To see if you believe the Naive Bayes model – which pretends that all variables are independent conditional on the value of default – for each pair of predictor variables, make side-by-side plots such that:

   - One plot corresponds to all data points with `default==1` and the other corresponds to all data points with `default==0`.
   - The two predictor variables are plotted against one another: scatter plots if looking at two continuous variables, overlaid histograms if looking at a continuous and a discrete variable.

   Based on these plots, do you see anything that makes you doubt the Naive Bayes model?

   (Whatever you decide... we're still going to fit one.)

   (b) To build a Naive Bayes model, you need to fit a distribution to every predictor conditioned on each state of default. For example:

   - For balance, fit a Gaussian to balance for everyone who defaulted, and a separate Gaussian to balance for everyone who did not default. We will call these $f(\text{balance}|\text{default})$ and $f(\text{balance}|\text{nodefault})$.
   - For income, fit a Gaussian to income for everyone who defaulted, and a separate Gaussian to income for everyone who did not default. We will call these $f(\text{income}|\text{default})$ and $f(\text{income}|\text{no default})$.
   - For student, fit a Bernoulli to student for everyone who defaulted, and a separte Bernoulli to student for everyone who did not default. You will get a $P(\text{student}|\text{default})$ and $P(\text{student}|\text{no default})$.

Use the usual maximum likelihood estimates for the parameters of each distribution. These estimates should be formed using the training data. Report the parameters of your distributions.

You will also need an estimate of the prior probability of default. Form a simple estimate, based on your training data, of $P(Y = 1)$

(c) Finally, we get to classify! Remember that we want to classify to the group with the largest $P(X|Y = k)P(Y = k)$. Because we'll be multiplying a bunch of densities and probabilities, we'll find it easier to work with the log. To classify each point, you will compare

$$\log P(\text{default}) + \log f(\text{balance}|\text{default}) + \log f(\text{income}|\text{default} + \log P(\text{student}|\text{default})$$

to

$$\begin{aligned}&\log P(\text{no default}) + \log f(\text{balance}|\text{no default}) + \log f(\text{income}|\text{no default})\\ &\quad + \log P(\text{student}|\text{default})\end{aligned}$$

Carry out these classifications on your test set.

Produce:

- A confusion matrix of your results, if you just classify to the $\{\text{default}, \text{no default}\}$ with the highest value.
- A misclassification rate corresponding to the same classifier.
- An ROC curve for the classifier. To do this, you need a score, rather than a binary classification. You can use:

$$\begin{aligned}&(\log P(\text{default}) + \log f(\text{balance}|\text{default}) + \log f(\text{income}|\text{default} + \log P(\text{student}|\text{default}))\\ &- (\log P(\text{no default}) + \log f(\text{balance}|\text{no default}) + \log f(\text{income}|\text{no default})\\ &\quad + \log P(\text{student}|\text{default}))\end{aligned}$$

  Feel free to use the built in ROC function here.

3. `Experimenting with class weights.` We talked a couple classes about unbalanced data sets and issues that they can cause for classifiers. We'll use the SVM as a demonstration of one of these issues and a potential fix.

   The file `generate_imbalance.py` contains a function to generate data in this problem. We'll look at a simple classification scenario under different proportions of 0 and 1s.

Hint: To get `plot_decision_regions` to work when you don't have any `y` values, you might need to assign `y` to be a constant vector of zeros or ones.

(a) Call `gen_data(500,100)` to draw *training* set with 500 observations in class 0 and 100 observations in class 1. Call it again with `gen_data(500,500)` to generate a *test* set (with different class balances).

Fit a linear SVM on the training set. (We will use $C = 1$ throughout this problem, rather than worry about tuning.)

Make a plot of the decision boundary on the *training* set, using `plot_decision_regions` from `mlxtend.plotting`. It should look pretty reasonable.

(b) Now make predictions on the *test* set using the SVM you trained in the previous part. Report a confusion matrix and a misclassification error. Do you notice anything strange? (Hint: Keep in mind that the test set is a balanced classification problem: there are an equal number of 0 and 1 in the data set.)

Make a plot of the decision boundary for the same classifier using `plot_decison_regions`, but this time plot it for the *test* set points. Can you see why you're getting weird classifications in this balanced problem?

(c) Retrain your SVM on the *training* sample, but this time set `class_weight='balanced'` inside your call to `SVC()`. (Read the documentation for `SVC` to see how this option is explained.) This puts greater losses on points from your rare class to counteract your imbalance.

With this new classifier, reproduce:

- Your confusion matrix on the test data.
- Your misclassification error on the test data.
- Your plot of the decision boundary on the test data.

How have all of these changed?

4. *Anomaly detection with SVMs.* When we talked about Mahalanobis distance, we hinted at its use for detecting rare events. It turns out that the SVM can also be modified for this purpose. I'll give you a brief explanation how next class, but for this assignment we'll just explore the idea.

A one-class SVM classifies whether points are "like your data" (guesses 1) or not (guesses 0). It is trained on a bunch of samples X, but no labels, since every observation is essentially a 1. Otherwise everything works like the SVM you're familiar with. The other difference is that the cost tuning parameter `C` is replaced by a similar tuning parameter `nu`. You'll explore `nu` a bit in this problem as well.

To fit the SVM, use the function `OneClassSVM` from `svm` in `sklearn`. For example:

```
svm.OneClassSVM(nu=0.1, kernel="rbf", gamma=0.5).fit(X,y)
```

The file `anomaly.R` contains starter code to generate data for this problem. You're going to start by trying a few different settings to see how this method behaves. Generate a data set with `get_data(500)`.

(a) Create a scatter plot of your data. It should not look anything like a Gaussian, so Mahalanobis distance would not work well.

(b) Fit the one-class SVM to your data. Use the `rbf` kernel, with `nu=0.1` and `gamma=0.5`. Plot the resulting decision regions, along with your data, using `plot_decision_regions`.

You should see a boundary around the sampled points. This is what the one-class SVM does, it tries to estimate a boundary around a small region that will tend to contain your future points.

(c) Experiment with changing `gamma` to be higher (to 2) and lower (to 0.1) and plot the results. What changes about the resulting boundary?

(d) Experiment with changing `nu` to be higher (0.4) and lower (0.01), while keeping `gamma` fixed at 0.5. Plot the results. What changes about the resulting boundary?

(e) Finally, let's actually use this! For a simple illustration, we'll use the adjusted closing prices of the DJIA from 2003-2009. We will try to look for anomalous patters in this time series.

One simple way to look for anomalous patterns in time series is to slide a fixed window of length K along the time series, and for each position in the time series, create a length K observation in a data set. This results in a $(n - K + 1) \times K$ data set, where each observation corresponds to a chunk of time. We can then compare to this data set to see if future observations differ significantly from past chunks of K days.

To make things easier, I've provided you with `djia_transformed.csv`, where this process has already been carried out. For this example, I used $K = 4$.

Fit a one-class SVM to the first 467 rows. These correspond to 2003-2004. Use an rbf kernel with `gamma=0.0000001` and `nu=0.01` (though the results are pretty insensitive).

Now we'll make predictions on the entire data set (2003-2009). We'll try to see if anything strange happened in the market during this time...

Plot the predictions as a function of time. Because nothing is really stationary here, points will pretty quickly look like anomalies outside of your training region, making this plot a little uninteresting.

To get a better signal for anomalies, we can plot the distance from the SVM boundary (including sign) as a function of time. To get this distance, use the `decision_function(X)` method of your fitted SVM instead of the predict function. Plot this quantity as a function of time. A negative score is a signal that a point looks anomalous. Do you see anything particularly anomalous, and does it correspond to anything that you expect?