INFO371 Problem Set 4: logistic regression, SVM, classification

Your name:

Deadline: Tue, Mar 5 midnight

Introduction

This problem set is long but I hope it is still doable :-) It revolves around classification, k-NN, logistic regression, and support vector machines. At the end you are asked to construct a ROC curve.

Please submit a) your code (notebooks, rmd, whatever) *and* b) the results in a final output form (html or pdf). You are free to choose either R or python for solving this problem set. In case of python, *scikit-learn* includes all the functionality. In case of R, the funcitonality is spread among different packages, I have had good experience with *FNN* for k-NN, *e1071* for svm (but *liquidSVM* is supposed to be better), logistic regression you can do using the base functionality. Most of the problems are about employing existing libraries and making plots. But two things you have to implement yourself: one is computing accuracy–precision–recall, and the other is creating a ROC curve.

You are welcome to answer some of the questions on paper but please include the result as an image in your final file. Note that you can easily include images in both notebooks and .rmd—besides of the code, both are just markdown documents.

Working together is fun and useful but you have to submit your own work. Discussing the solutions and problems with your classmates is all right but do not copy-paste their solution! Please list all your collaborators below:

1.
2. ...

Wisconsin Breast Cancer Dataset

You will work with Wisconsin Breast Cancer Dataset (WBCD), available at UCI Machine Learning Repository. You can download it from the internet but rather use the files *wdbc.csv.bz2* and *wdbc_ doc.txt* from canvas (under *files/data*) where I have added the variable names to the data. The first file is the csv with variable names, the second one a brief description of the data.

The data includes diagnosis of the tumor with "M" meaning cancer (*malignant*) and "B" no cancer (*benign*), and 10 features, describing physical properties of cell nuclei from biopsy samples. Each feature is represented three times, once for mean, once for standard error, and once for the worst values. Your task is to predict diagnosis based on this data.

1 Explore the data

As the first step, explore the data.

1. Load the data. You may drop *id* or just ignore it in the rest of your analysis.

2. Create a summary table where you show means, ranges, and number of missings for each variable. In addition, add (Pearson) correlation between the diagnosis and the corresponding feature. You may include more statistics you consider useful.

3. Graphical exploration. Make a number of scatterplots where you explore the relationship between features and the diagnosis.

   Note: you may attempt to display all 435 possible combinations as a scatterplot matrix, but that will most likely be just unreadable. Choose instead a few cases with high correlation and a few with medium/low correlation. Avoid overwhelming your reader with tens of similar figures.

2 Decision Boundary

Your next task is to plot the decision boundary when using logistic regression. You will also play a little with feature engineering.

If you are uncertain about what is decision boundary, I recommend to consult James et al. (2015) book Section 2. For instance, Figure 2.13 on p38 depicts a 2D classification case where certain X_1, X_2 values are classified as orange and others as blue. Decision boundary is the dashed winding line that separates these two regions.

There are two broad strategies to plot it. In any case, you have first to estimate (train) your model. Thereafter you have to predict the classes (cancer/no cancer here) on a regular dense grid that covers the parameter space (this is the small blue/orange dots on figure 2.13). Afterwards you can either plot your predicted values with a certain color code, or alternatively, say, set predicted M to 1 and predicted B to 0, and make a contour plot for a single contour at level 0.5. You may also combine these both methods. Some example code in R is provided here, python version will be in a separate file.

We ignore training/testing/overfitting issues for now.

2.1 kNN Case

First, let's explore the decision boundary using k-NN.

Pick two features. I recommend to use a few that show relative strong correlation with diagnosis, and that vary widely across the scatterplot (not just highly correlated narrow line). Feel free to use a combination you already plotted above.

1. Predict the diagnosis on a grid (say, 100×100, or less if this is too slow) that covers the range of the explanatory variables. Use k-NN with $k = 3..7$ (pick just one value). This gives you 100x100 predicted diagnoses. See the example code.

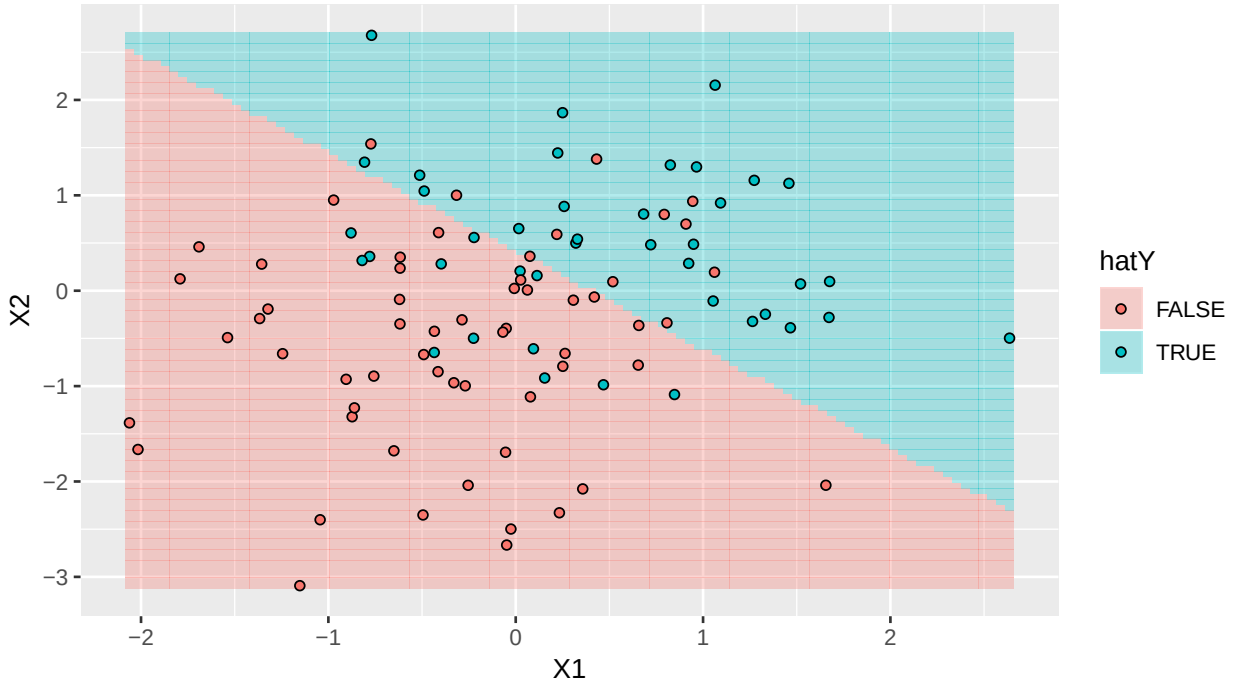
   Note: if your features are of very different scale, you should either scale these into a roughly equal scale, or use a metric that does this with you. Consult James et al. (2015, p 217).

2. Plot the actual data and the decision boundary on the same plot. Ensure that actual observations and predictions are clearly distinguishable, and that one can easily understand the color code.

   Here is an example code in R on how you may do this. We just generate some random data and use logistic regression.

```
X1 <- rnorm(100)
X2 <- rnorm(100)
Y <- X1 + X2 + rnorm(100) > 0
                              # this is a logical outcome
```

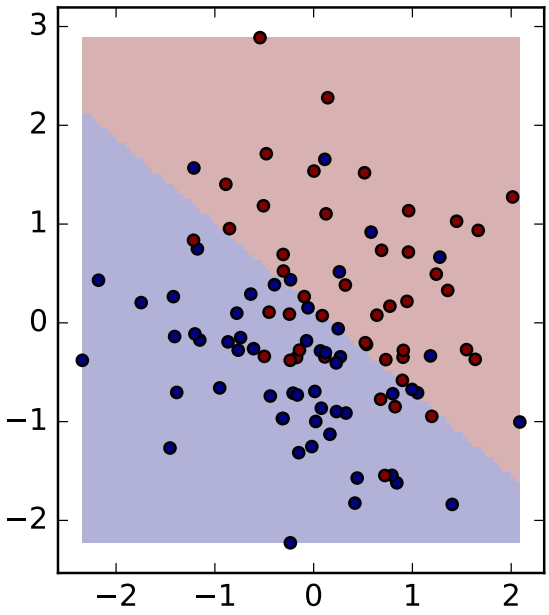
```r
m <- glm(Y ~ X1 + X2, family=binomial(link="logit"))
## Now create the regular grid:
ex1 <- seq(min(X1), max(X1), length.out=100)
ex2 <- seq(min(X2), max(X2), length.out=100)
g <- expand.grid(X1=ex1, X2=ex2)
                          # this is 100x100 dataframe of regular grid points
## Predict the outcome
## Note: glm predicts various things, 'response' means probability
hatY <- predict(m, newdata=g, type="response") > 0.5
                          # we set hatY = 1 if probability > 0.5
ggplot(g) +
   geom_tile(aes(X1, X2, fill=hatY), alpha=0.3) +
   geom_point(data=data.frame(), aes(X1, X2, fill=Y), shape=21)
```



Python example code is here:

```python
import numpy as np
import matplotlib.pyplot as plt
from sklearn.linear_model import LogisticRegression
##
X1 = np.random.normal(size=100)
X2 = np.random.normal(size=100)
Y = X1 + X2 + np.random.normal(size=100) > 0
X = np.stack((X1, X2), axis=1)
m = LogisticRegression().fit(X, Y)
## Now greate the regular grid
ex1 = np.linspace(X1.min(), X1.max(), 100)
ex2 = np.linspace(X2.min(), X2.max(), 100)
xx1, xx2 = np.meshgrid(ex1, ex2)
```

```
                            # unlike R-s 'expand.grid', meshgrid creates matrices
g = np.stack((xx1.ravel(), xx2.ravel()), axis=1)
                            # we create the design matrix by stacking the xx1, xx2
                            # after unraveling those into columns
## predict on the grid
hatY = m.predict(g).reshape(100,100)
                            # imshow wants a mtrix, so we reshape the predicted vector
into one
plt.imshow(hatY, extent=(X1.min(), X1.max(), X2.min(), X2.max()),
           interpolation='none', origin='lower',
                            # you need to specify that the image begins from below,
                            # not above, otherwise it will be flipped around
           alpha=0.3)
plt.scatter(X1, X2, c=Y)
plt.show()
```



3. Describe your observations. How good is kNN in picking up the actual shape? Does it also pick up noise?

   Note: unless you do cross-validation, you cannot know if the model picks up noise (overfit). Here I just ask your best judgement, not any formal analysis.

2.2 Logistic Regression

Now repeat the process above by logistic regression. Pick the same features as what you used for k-NN above.

1. Fit a logistic regression model with these two features.
2. Predict the diagnosis on a similar grid...

3. ...and create a similar plot.

4. Describe your observations. How does the result for kNN compare to that for Logistic Regression?

2.3 Feature Engineering

So far you were using just two of the existing features in the data. However, now let's create a few more.

1. Use the two features you used above to compute some new ones. Let's denote your original features by x_1 and x_2. Examples you may create include: x_1^2, x_2^2, $x_1 \cdot x_2$, $\mathbb{1}(x_1 > 5)$, $\mathbb{1}(x_1 < 1) \cdot x_2^2$, $\log x_1$ ...You can use all sorts of mathematical operations as long as a) you only use x_1 and x_2 and no other features, and b) the original and the engineered features remain linearly independent (that is, you don't create features like $\alpha \cdot x_1 + \beta \cdot x_2$).

2. Fit a logistic regression model. However, this time pick both x_1, x_2, and some of your engineered features.

3. Create the decision boundary plot.

   Although you now have more than two features, you still plot it using the x_1–x_2 axes as above. This is because all your engineered features are functionally dependent on x_1 and x_2 only: if we know values of x_1 and x_2, we also know exactly the values of your engineered features. We have more than two features but these are not independent (but they should still be *linearily independent*).

4. Comment on the shape of the boundary. What do you think, how well can you capture the actual boundary? What about overfitting? (Again, I ask about your judgement, not about any formal analysis).

5. Repeat the exercise a few times where you pick/engineer different new features, and try to get as reasonable boundary as you can.

   As above, I am asking "reasonable" boundary in the sense of your best judgement. No actual cross validation is necessary.

Note: I have mixed experience with `scikit-learn.linear_model.LogisticRegression`. It occasionally appears the default convergence tolerance is far too big, and the default *liblinear* solver too imprecise. Setting tolerance to 10^{-12} and solver to *lbfgs* improved the results for me, but did not make it flawless.

2.4 Support Vector Machines

As you may have guessed, here your task is to do essentially the same thing but using SVM instead of the other methods. Use the same features what what you used above for k-NN and logistic, but instead of playing with feature engineering, you will play with different kernels.

1. Fit a SVM regression model with these two features.

2. Predict the diagnosis on a similar grid...

3. ...and create a similar plot.

4. repeat with a few different kernels (check the software documentation about how to choose kernels and related parameters).

5. Describe your observations. Which kernel gives the most reasonable results? How does the result for kNN compare to that for Logistic Regression?

3 ROC curve: which estimator is the best

Finally, let's see which estimator is the best. We compare these in two ways: first using accuracy, precision, recall; and thereafter with ROC curve. We stay with the two features you have been using so far, for the logistic one you also add the engineered features. Consult James et al. (2015, p 148) for what is the ROC curve. Now we are going to compare the best models of k-NN, logistic regression, and SVM.

1. split your data into training-validation parts (say, 75-25).
2. select good models from all of your previous sections (i.e. k-NN, logistic, SVM). You may select all models you tried, but if some of those were clearly inferior feel free to skip those.
3. Train each model on training data. Using the validation data for prediction and present the confusion matrix and compute accuracy, precision, recall, and F-score.

   Note: in this point you have to **implement these algorithms** yourself! do not use existing libraries!

Now let's compare these models with ROC curves by playing with different thresholds.

4. Use the models you used above for computing the accuracy and all that. Now instead of predicting the oucome class, predict probability that the outcome belongs to a given class (say, "Malignant"). The library you use may do it already anyway.

   Note that while Logistic regression and SVM normally predict probability of the target class, at least internally, this may not be the case for k-NN. If the software you are using supports it, use the proportion of the "successes" among your k nearest neighbors as the probability (well, it only makes sense if $k > 1$).
5. Now pick a sequence of probability thresholds between 0 and 1 (for instance, 0, 0.1, 0.2, ...). For each model and each threshold, treat your predictions to be "Malignant" if it it's probability is at least as big as the threshold.
6. Based on these predictions, compute false positive rate and true positive rate for each threshold value. Plot these rates for each threshold and each model. This is your ROC curve.
7. Comment your results. Which model is the best?

References

James, G., Witten, D., Hastie, T., Tibshirani, R., 2015. An Introduction to Statistical Learning with Applications in R. Springer.