CMSE 381, Fundamental Data Science Methods

November 16, 2025

Homework 8

Lowell Monis

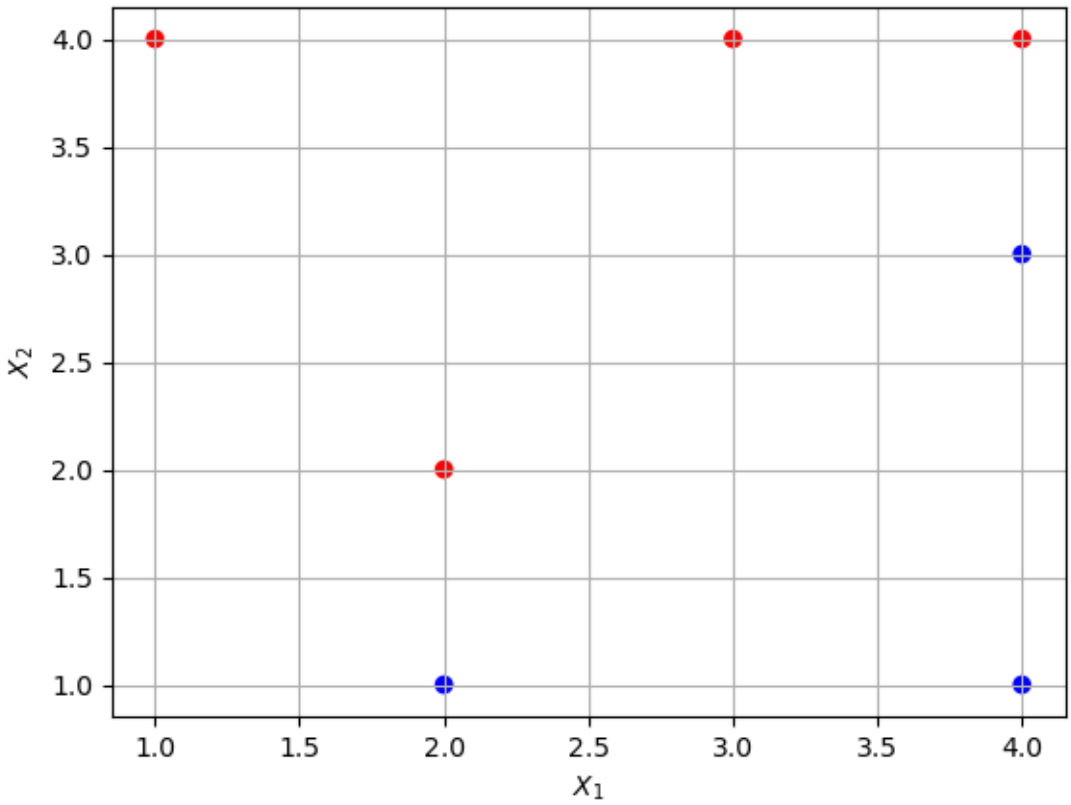
Instructor: Dr. Mengsen Zhang

Question 1: ISLP § 9.7.3

Only a subset of these problems are to be answered. Here we explore the maximal margin classifier on a toy data set.

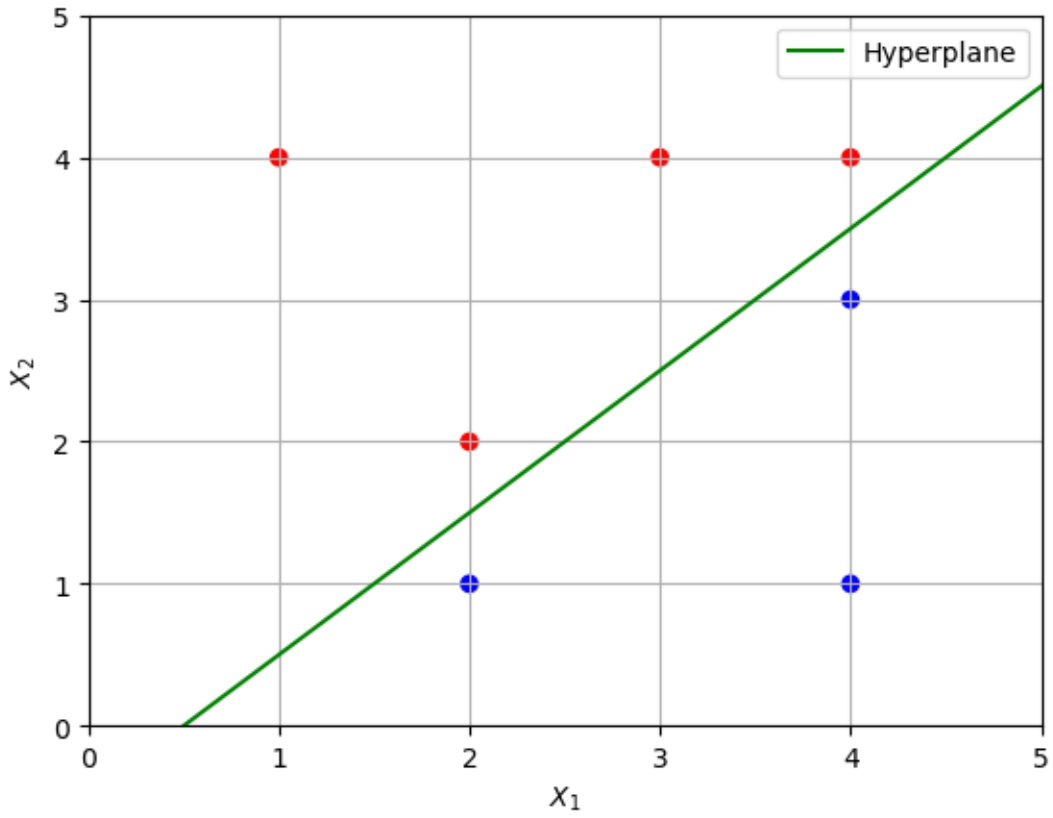
(a) We are given $n = 7$ observations in $p = 2$ dimensions. For each observation, there is an associated class label. Sketch the observations.

Obs.	$X1$	$X2$	Y
1	3	4	Red
2	2	2	Red
3	3	4	Red
4	1	4	Red
5	2	1	Blue
6	4	3	Blue
7	4	1	Blue



(b) Sketch the optimal separating hyperplane, and provide the equation for this hyperplane (of the form in ISLP Equation § 9.1).

$$Y = \frac{1}{2} - X_1 + X_2$$



(c) Describe the classification rule for the maximal margin classifier. It should be something along the lines of "Classify to Red if $\beta_0 + \beta_1 X_1 + \beta_2 X_2 > 0$, and classify to Blue otherwise." Provide the values for β_0, β_1, and β_2. The classification rule for the maximal margin classifier is as follows:

Classify to Red if $\frac{1}{2} - X_1 + X_2 > 0$

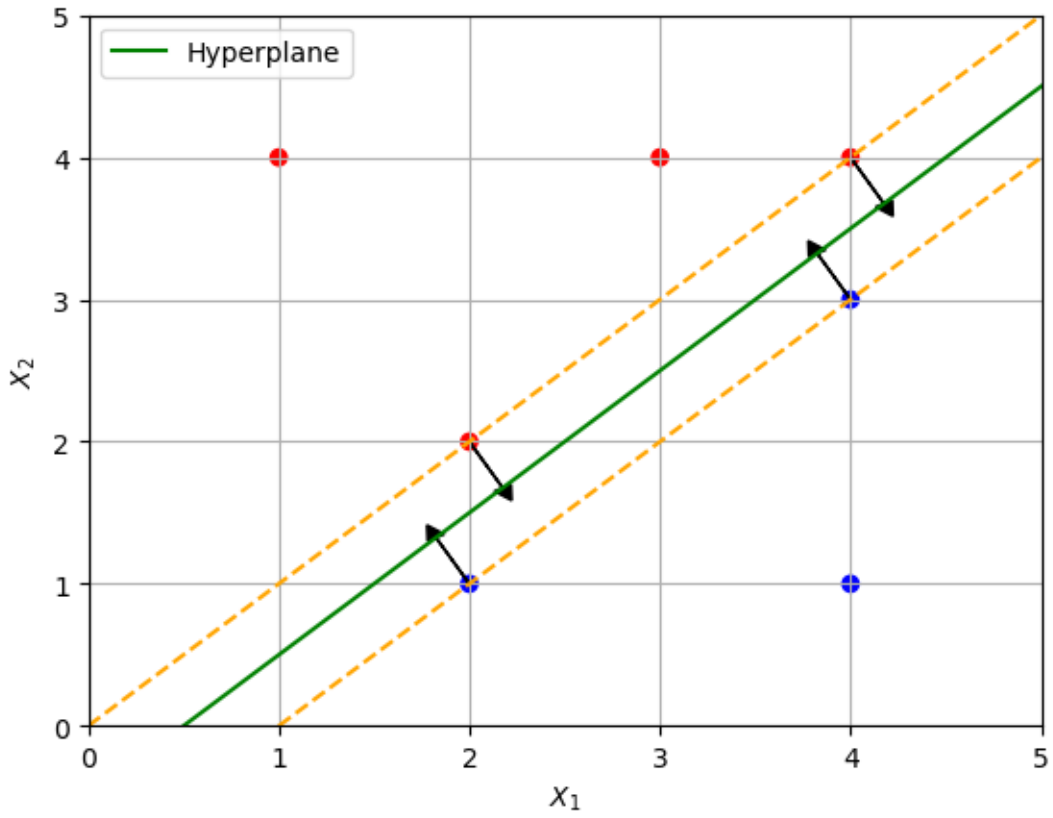
Classify to Blue if $\frac{1}{2} - X_1 + X_2 < 0$

Here, $\beta_0 = \frac{1}{2}, \beta_1 = -1, \beta_2 = 1$.

(d) On your sketch, indicate the margin for the maximal margin hyperplane. The margin is the minimum distance from the support vectors to the hyperplane. We can compute this margin using $Y(\beta_0 + \beta_1 X_1 + \beta_2 X_2)$. We can then sketch the margin on the existing plot from the support vectors.

[4]:
```
min(abs(-np.array(x1)+np.array(x2)+0.5))
```

[4]:
```
0.5
```



The margin is 0.5 units.

(e) Indicate the support vectors for the maximal margin classifier. The support vectors are the points that are closest to the hyperplane. All of the support vectors are equidistant from the hyperplane, with the distance being equal to the margin. There are four such support vectors for this maximal margin classifier. These are $(2, 1)$, $(4, 3)$, $(2, 2)$, and $(4, 4)$.

(f) Argue that a slight movement of the seventh observation would not affect the maximal margin hyperplane. Observation 7, which is $(4, 1)$, is not a support vector. Thus, if this observation is moved slightly, it will not change the margins, or the hyperplane, since these depend on the points closest to the optimal boundary, or the support vectors.

Question 2: ISLP § 9.7.8

This problem involves the `OJ` data set which is part of the `ISLP` package. The problem has been slightly modified from the textbook version by the instructors.

(a) Create a training set containing a random sample of 800 observations, and a test set containing the remaining observations.

[6]:
```
oj=pd.read_csv('../data/OJ.csv').rename({'Unnamed: 0': ''}, axis=1).set_index('')
oj_train, oj_test = train_test_split(oj, train_size=800, random_state=381)
```

(b) Fit a support vector classifier to the training data using `C = 1`, with `Purchase` as the response and the other variables as predictors. How many support points are there? First, I will separate the predictors and the response variables

[7]:
```
X_train = pd.get_dummies(oj_train.drop('Purchase', axis=1), drop_first=True)
X_test = pd.get_dummies(oj_test.drop('Purchase', axis=1), drop_first=True)
y_train = oj_train['Purchase']
y_test = oj_test['Purchase']
```

We can now train a support vector classifier to the training data using the given hyperparameter.

[8]:
```
svc = SVC(C=1, kernel='linear', random_state=381)
svc.fit(X_train,y_train)
```

[8]:
```
SVC(C=1, kernel='linear', random_state=381)
```

We can now query the number of support points used.

[9]:
```
len(svc.support_)
```

[9]:
```
345
```

(c) What are the training and test error rates? We can compute the training and test classification errors as follows:

[10]:
```
1-accuracy_score(y_train, svc.predict(X_train))
```

[10]:
```
0.15375000000000005
```

[11]:
```
1-accuracy_score(y_test, svc.predict(X_test))
```

[11]:
```
0.18148148148148147
```

The training error rate is 15.375%, and the test error is slightly higher (as expected), at 18.148%.

(d) Use cross-validation to select an optimal `C`. Consider values in the range 0.01 to 10. First, I am defining a good list of candidate values for `C`. I will then use a `GridSearchCV` model fitted to identify the optimal value using 5-fold cross-validation. I will continue using the `linear` kernel.

[12]:
```
C_list = [0.01, 0.1, 1, 10]
tuned_parameters = [{'C': C_list}]
clf = GridSearchCV(SVC(kernel='linear'), tuned_parameters, cv=5,␣
 ↪scoring='accuracy')
clf.fit(X_train, y_train)
```

[12]:
```
GridSearchCV(cv=5, estimator=SVC(kernel='linear'),
             param_grid=[{'C': [0.01, 0.1, 1, 10]}], scoring='accuracy')
```

I will now query and store the optimal value of `C` from the tuned model.

[13]:
```
opt_C=clf.best_params_
print(opt_C)
```

```
{'C': 1}
```

(e) Compute the training and test error rates using this new value for `C`. I will use the `predict()` method of the `clf` object, which will use `C=1` as its optimal parameter.

[14]:
```
1-accuracy_score(y_train, clf.predict(X_train))
```

[14]:
```
0.15375000000000005
```

[15]:
```
(1-accuracy_score(y_test, clf.predict(X_test)))
```

[15]:
```
0.18148148148148147
```

The test and training errors are the same as the ones found in (c), since the optimal value for `C` was the same as the one used for parts (b) and (c) of this question.

(f) Repeat parts (b) through (e) using a support vector machine with a radial kernel. Use the default value for `gamma`. We first use `C=1` to train the SVC model.

[16]:
```
svc_rbf = SVC(C=1, kernel='rbf', random_state=381)
svc_rbf.fit(X_train,y_train)
len(svc_rbf.support_)
```

[16]:
```
634
```

There are 634 support vectors in this model. We can now compute the errors.

[17]:
```
1-accuracy_score(y_train, svc_rbf.predict(X_train))
```

[17]:
```
0.39375000000000004
```

[18]:
```
1-accuracy_score(y_test, svc_rbf.predict(X_test))
```

[18]:
```
0.37777777777777777
```

It looks like the error rates have increased with training error set at 39.375% and test error at 37.778%.

I can now proceed with a 5-fold cross-validation to determine the ideal value for `C`, keeping the value of `gamma` at its default.

[19]:
```
C_list = [0.01, 0.02, 0.05, 0.1, 0.2, 0.5, 1, 2, 5, 10]
tuned_parameters = [{'C': C_list}]
clf_rbf = GridSearchCV(SVC(kernel='rbf'), tuned_parameters, cv=5,␣
↪scoring='accuracy')
clf_rbf.fit(X_train, y_train)
print(clf_rbf.best_params_)
```

```
{'C': 0.01}
```

Finally, I will print the error rates associated with using the optimal `C` value.

[20]:
```
1-accuracy_score(y_train, clf_rbf.predict(X_train))
```

[20]:
```
0.39375000000000004
```

[21]:
```
1-accuracy_score(y_test, clf_rbf.predict(X_test))
```

[21]:
```
0.37777777777777777
```

The training error is once again set at 39.375% and test error at 37.778%.

(g) Repeat parts (b) through (e) using a support vector machine with a polynomial kernel. Set `degree = 2`. We first use `C=1` to train the SVC model.

[22]:
```
svc_poly = SVC(C=1, kernel='poly', degree=2, random_state=381)
svc_poly.fit(X_train,y_train)
len(svc_poly.support_)
```

[22]:
```
632
```

There are 632 support vectors in this model. We can now compute the errors.

[23]:
```
1-accuracy_score(y_train, svc_poly.predict(X_train))
```

[23]:
```
0.39375000000000004
```

[24]:
```
1-accuracy_score(y_test, svc_poly.predict(X_test))
```

[24]:
```
0.37777777777777777
```

The training error is once again set at 39.375% and test error at 37.778%.

I can now proceed with a 5-fold cross-validation to determine the ideal value for `C`.

[25]:
```
C_list = [0.01, 0.02, 0.05, 0.1, 0.2, 0.5, 1, 2, 5, 10]
tuned_parameters = [{'C': C_list}]
clf_poly = GridSearchCV(SVC(kernel='poly', degree=2), tuned_parameters, cv=5,␣
↪scoring='accuracy')
```

```
clf_poly.fit(X_train, y_train)
print(clf_poly.best_params_)
```

```
{'C': 0.01}
```

Finally, I will print the error rates associated with using the optimal `C` value.

[26]:
```
1-accuracy_score(y_train, clf_poly.predict(X_train))
```

[26]:
```
0.39375000000000004
```

[27]:
```
1-accuracy_score(y_test, clf_poly.predict(X_test))
```

[27]:
```
0.37777777777777777
```

The training error is once again set at 39.375% and test error at 37.778%.

(h) Overall, which approach seems to give the best results on this data? From the linear, radial, and polynomial kernels, it looks like using a linear kernel with `C=1` has given us the most optimal result with the least classification error rates. An interesting finding is that both the radial and polynomial kernels returned the same error rates after finding their respective optimal hyperparameters.