assignment_5

May 4, 2025

1 STA-6543 Assignment 5

Jason Gillette

1.1 Question 2.

For parts (a) through (c), indicate which of i. through iv. is correct. Justify your answer.

1.2 Question 2a.

The lasso, relative to least squares, is:

```
iii. Less flexible and hence will give improved prediction accuracy when its increase in bias i

    Lasso works when the benefit of reduced variance outweighs the cost of increased bias. It i
```

1.3 Question 2b.

Repeat (a) for ridge regression relative to least squares.

```
(iii) Less flexible and hence will give improved prediction accuracy when its increase in bias

    Least squares regression places no restrictions on a variable's coefficient to influence fi
```

1.4 Question 2c.

Repeat (a) for non-linear methods relative to least squares.

```
(i) More flexible and hence will give improved prediction accuracy when its increase in bias is

    Non-linear methods are not bound by a straight line and fit more complex patterns in the da
```

1.5 Question 9.

In this exercise, we will predict the number of applications received using the other variables in the College data set.

[27]:
```python
from ISLP import load_data
import pandas as pd

# Load College dataset
```

```python
college = load_data('College')

# Look at initial structure
college.info()
college.head()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 777 entries, 0 to 776
Data columns (total 18 columns):
 #   Column       Non-Null Count  Dtype
---  ------       --------------  -----
 0   Private      777 non-null    category
 1   Apps         777 non-null    int64
 2   Accept       777 non-null    int64
 3   Enroll       777 non-null    int64
 4   Top10perc    777 non-null    int64
 5   Top25perc    777 non-null    int64
 6   F.Undergrad  777 non-null    int64
 7   P.Undergrad  777 non-null    int64
 8   Outstate     777 non-null    int64
 9   Room.Board   777 non-null    int64
 10  Books        777 non-null    int64
 11  Personal     777 non-null    int64
 12  PhD          777 non-null    int64
 13  Terminal     777 non-null    int64
 14  S.F.Ratio    777 non-null    float64
 15  perc.alumni  777 non-null    int64
 16  Expend       777 non-null    int64
 17  Grad.Rate    777 non-null    int64
dtypes: category(1), float64(1), int64(16)
memory usage: 104.2 KB
```

[27]:

```
  Private  Apps  Accept  Enroll  Top10perc  Top25perc  F.Undergrad  \
0     Yes  1660    1232     721         23         52         2885
1     Yes  2186    1924     512         16         29         2683
2     Yes  1428    1097     336         22         50         1036
3     Yes   417     349     137         60         89          510
4     Yes   193     146      55         16         44          249

   P.Undergrad  Outstate  Room.Board  Books  Personal  PhD  Terminal  \
0          537      7440        3300    450      2200   70        78
1         1227     12280        6450    750      1500   29        30
2           99     11250        3750    400      1165   53        66
3           63     12960        5450    450       875   92        97
4          869      7560        4120    800      1500   76        72

   S.F.Ratio  perc.alumni  Expend  Grad.Rate
```

```
0       18.1           12    7041          60
1       12.2           16   10527          56
2       12.9           30    8735          54
3        7.7           37   19016          59
4       11.9            2   10922          15
```

[28]:

```
### adding a cleaning step due to all the value errors downstream.

# Rename columns to replace '.' with '_' for sml syntax
college.columns = [col.replace('.', '_') for col in college.columns]

# Convert 'Private' from Yes/No to 1/0 int
college['Private'] = college['Private'].cat.codes

# Drop any rows with NaNs
college = college.dropna()

# confirm casting and rename
college.info()
college.head()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 777 entries, 0 to 776
Data columns (total 18 columns):
 #   Column       Non-Null Count  Dtype
---  ------       --------------  -----
 0   Private      777 non-null    int8
 1   Apps         777 non-null    int64
 2   Accept       777 non-null    int64
 3   Enroll       777 non-null    int64
 4   Top10perc    777 non-null    int64
 5   Top25perc    777 non-null    int64
 6   F_Undergrad  777 non-null    int64
 7   P_Undergrad  777 non-null    int64
 8   Outstate     777 non-null    int64
 9   Room_Board   777 non-null    int64
 10  Books        777 non-null    int64
 11  Personal     777 non-null    int64
 12  PhD          777 non-null    int64
 13  Terminal     777 non-null    int64
 14  S_F_Ratio    777 non-null    float64
 15  perc_alumni  777 non-null    int64
 16  Expend       777 non-null    int64
 17  Grad_Rate    777 non-null    int64
dtypes: float64(1), int64(16), int8(1)
memory usage: 104.1 KB
```

[28]:
```
   Private  Apps  Accept  Enroll  Top10perc  Top25perc  F_Undergrad  \
0        1  1660    1232     721         23         52         2885
1        1  2186    1924     512         16         29         2683
2        1  1428    1097     336         22         50         1036
3        1   417     349     137         60         89          510
4        1   193     146      55         16         44          249

   P_Undergrad  Outstate  Room_Board  Books  Personal  PhD  Terminal  \
0          537      7440        3300    450      2200   70        78
1         1227     12280        6450    750      1500   29        30
2           99     11250        3750    400      1165   53        66
3           63     12960        5450    450       875   92        97
4          869      7560        4120    800      1500   76        72

   S_F_Ratio  perc_alumni  Expend  Grad_Rate
0       18.1           12    7041         60
1       12.2           16   10527         56
2       12.9           30    8735         54
3        7.7           37   19016         59
4       11.9            2   10922         15
```

1.6 Question 9a.

Split the data set into a training set and a test set.

[29]:
```python
from sklearn.model_selection import train_test_split

# set random seed for reproducibility
RANDOM_STATE = 42

# Split the College data into features (X) and target (y)
X = college.drop(columns='Apps')  # predictors
y = college['Apps']               # target: number of applications

# Perform train-test split
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.3, random_state=RANDOM_STATE
)

# sizes
print(f"Training set: {X_train.shape}, Test set: {X_test.shape}")
```

```
Training set: (543, 17), Test set: (234, 17)
```

1.7 Question 9b.

Fit a linear model using least squares on the training set, and report the test error obtained.

[30]:
```python
import pandas as pd
import statsmodels.formula.api as smf
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error

# join X and y back into single training/test DataFrame
train = X_train.copy()
train['Apps'] = y_train

test = X_test.copy()
test['Apps'] = y_test

# Fit model
# Cite OpenAi o4 here, examples in book caused syntax error.
model = smf.ols(
    'Apps ~ Private + Accept + Enroll + Top10perc + Top25perc + F_Undergrad + P_Undergrad + Outstate + Room_Board + Books + Personal + PhD + Terminal + S_F_Ratio + perc_alumni + Expend + Grad_Rate',
    data=train
).fit()

# Predict
y_pred = model.predict(test)

# Test MSE
mse_test = mean_squared_error(test['Apps'], y_pred)
print(f"Test MSE: {mse_test:.4f}")
```

```
Test MSE: 1931803.1942
```

1.8 Question 9c.

Fit a ridge regression model on the training set, with chosen by cross-validation. Report the test error obtained.

[31]:
```python
from sklearn.linear_model import RidgeCV
from sklearn.preprocessing import StandardScaler
from sklearn.pipeline import make_pipeline
import numpy as np

# Ridge requires scaled predictors
ridge_model = make_pipeline(
    StandardScaler(),
    RidgeCV(alphas=10**np.linspace(10, -2, 100), store_cv_results=True)
)

# Fit ridge on training data
ridge_model.fit(X_train, y_train)
```

```
# Predict on test data
y_pred_ridge = ridge_model.predict(X_test)

# Test MSE
mse_ridge = mean_squared_error(y_test, y_pred_ridge)
print(f"Test MSE (Ridge): {mse_ridge:.4f}")

# Best lambda chosen
best_lambda = ridge_model.named_steps['ridgecv'].alpha_
print(f"Optimal lambda (alpha) via CV: {best_lambda:.4f}")
```

```
Test MSE (Ridge): 1931467.5647
Optimal lambda (alpha) via CV: 0.0100
```

Ridge regression slightly improved test error by ~335 units, a modest gain, suggesting the least squares model may not be severely overfitting and Ridge Regression helped, but only marginally.

1.9 Question 9d.

Fit a lasso model on the training set, with chosen by cross-validation. Report the test error obtained, along with the number of non-zero coefficient estimates.

[32]:
```
from sklearn.linear_model import LassoCV
from sklearn.pipeline import make_pipeline
from sklearn.preprocessing import StandardScaler
import numpy as np
from sklearn.metrics import mean_squared_error

# Lasso model pipeline: standardize then cross-validated lasso
lasso_model = make_pipeline(
    StandardScaler(),
    LassoCV(alphas=10**np.linspace(10, -2, 100), cv=10, max_iter=10000,␣
 ↪random_state=42)
)

# Fit model on training data
lasso_model.fit(X_train, y_train)

# Predict on test set
y_pred_lasso = lasso_model.predict(X_test)

# Compute test MSE
mse_lasso = mean_squared_error(y_test, y_pred_lasso)

# Extract best lambda and coefficients
lasso_cv = lasso_model.named_steps['lassocv']
best_lambda = lasso_cv.alpha_
```

```
nonzero_coefs = np.sum(lasso_cv.coef_ != 0)

print(f"Test MSE (Lasso): {mse_lasso:.4f}")
print(f"Optimal lambda (alpha): {best_lambda:.4f}")
print(f"Number of non-zero coefficients: {nonzero_coefs}")
```

```
Test MSE (Lasso): 1931787.4564
Optimal lambda (alpha): 0.0100
Number of non-zero coefficients: 17
```

Lasso chose the same number of predictors (17), meaning none were shrunk to zero; suggests all predictors are at least marginally useful. The MSE is slightly better than least squares, but slightly worse than ridge regression.

1.10 Question 9e.

Fit a PCR model on the training set, with M chosen by cross-validation. Report the test error obtained, along with the value of M selected by cross-validation.

[35]:
```
from sklearn.decomposition import PCA
from sklearn.linear_model import LinearRegression
from sklearn.preprocessing import StandardScaler
from sklearn.pipeline import Pipeline
from sklearn.model_selection import cross_val_score
import numpy as np

# total number of features (17)
m_values = list(range(1, X_train.shape[1] + 1))
cv_errors = []

for m in m_values:
    # Create PCR pipeline
    pcr_pipeline = Pipeline([
        ('scale', StandardScaler()),
        ('pca', PCA(n_components=m)),
        ('linreg', LinearRegression())
    ])

    # Negative MSE (so higher is better) across 10 folds
    scores = cross_val_score(pcr_pipeline, X_train, y_train,
                             scoring='neg_mean_squared_error', cv=10)

    # Average MSE (flip sign)
    cv_errors.append(-scores.mean())

# Get best M
best_m = m_values[np.argmin(cv_errors)]
print(f"Best number of components (M): {best_m}")
```

```
# Final PCR model with best M
final_pcr = Pipeline([
    ('scale', StandardScaler()),
    ('pca', PCA(n_components=best_m)),
    ('linreg', LinearRegression())
])

final_pcr.fit(X_train, y_train)

# Predict on test set
y_pred_pcr = final_pcr.predict(X_test)

# Test MSE
from sklearn.metrics import mean_squared_error
mse_pcr = mean_squared_error(y_test, y_pred_pcr)
print(f"Test MSE (PCR): {mse_pcr:.4f}")
```

```
Best number of components (M): 17
Test MSE (PCR): 1931803.1942
```

The exact same test MSE as standard least squares… suggesting all variables are uniquely contributing and consistent with other model results?

1.11 Question 9f.

Fit a PLS model on the training set, with M chosen by cross-validation. Report the test error obtained, along with the value of M selected by cross-validation

[36]:
```
from sklearn.cross_decomposition import PLSRegression
from sklearn.model_selection import cross_val_score
from sklearn.preprocessing import StandardScaler
from sklearn.pipeline import Pipeline
from sklearn.metrics import mean_squared_error
import numpy as np

m_values = list(range(1, X_train.shape[1] + 1))  # Try 1 to 17 components
cv_errors = []

for m in m_values:
    # Build PLS pipeline
    pls_pipeline = Pipeline([
        ('scale', StandardScaler()),
        ('pls', PLSRegression(n_components=m))
    ])

    # Use negative MSE scoring, average over 10-fold CV
    scores = cross_val_score(pls_pipeline, X_train, y_train,
```

```
                              scoring='neg_mean_squared_error', cv=10)

    cv_errors.append(-scores.mean())  # Flip sign for MSE

# Best number of components
best_m = m_values[np.argmin(cv_errors)]
print(f"Best number of components (M): {best_m}")

# Final PLS model with best M
final_pls = Pipeline([
    ('scale', StandardScaler()),
    ('pls', PLSRegression(n_components=best_m))
])

final_pls.fit(X_train, y_train)

# Predict on test set
y_pred_pls = final_pls.predict(X_test)

# Evaluate
mse_pls = mean_squared_error(y_test, y_pred_pls)
print(f"Test MSE (PLS): {mse_pls:.4f}")
```

```
Best number of components (M): 14
Test MSE (PLS): 1930317.7199
```

PLS performed best among all models so far, reducing test MSE by about 1,485. It only 14 used components meaning it eliminated 3 components that added noise rather than predictive signal. Since it uses y during component extraction, it focused only on signal relevant to prediction.

1.12 Question 9g.

Comment on the results obtained. How accurately can we predict the number of college applications received? Is there much difference among the test errors resulting from these five approaches?

Model	Test MSE	Comments
Linear (OLS)	1,931,803.19	Baseline
Ridge (CV)	1,931,467.56	Slight improvement
Lasso (CV)	1,931,787.46	No real feature selection
PCR (CV)	1,931,803.19	Identical to OLS (M=17)
PLS (CV)	1,930,317.72	Best, using 14 components

The differences in test error are minimal, less than 0.1% between worst and best models. This suggests all 17 predictors are contributing at least modestly and regularization and dimensionality reduction helped only slightly

1.13 Question 11.

We will now try to predict per capita crime rate in the Boston dataset.

[37]:
```python
from ISLP import load_data
import pandas as pd

# Load the Boston dataset
boston = load_data('Boston')

# Preview the data
boston.info()
boston.head()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 506 entries, 0 to 505
Data columns (total 13 columns):
 #   Column   Non-Null Count  Dtype  
---  ------   --------------  -----  
 0   crim     506 non-null    float64
 1   zn       506 non-null    float64
 2   indus    506 non-null    float64
 3   chas     506 non-null    int64  
 4   nox      506 non-null    float64
 5   rm       506 non-null    float64
 6   age      506 non-null    float64
 7   dis      506 non-null    float64
 8   rad      506 non-null    int64  
 9   tax      506 non-null    int64  
 10  ptratio  506 non-null    float64
 11  lstat    506 non-null    float64
 12  medv     506 non-null    float64
dtypes: float64(10), int64(3)
memory usage: 51.5 KB
```

[37]:

	crim	zn	indus	chas	nox	rm	age	dis	rad	tax	ptratio	\
0	0.00632	18.0	2.31	0	0.538	6.575	65.2	4.0900	1	296	15.3	
1	0.02731	0.0	7.07	0	0.469	6.421	78.9	4.9671	2	242	17.8	
2	0.02729	0.0	7.07	0	0.469	7.185	61.1	4.9671	2	242	17.8	
3	0.03237	0.0	2.18	0	0.458	6.998	45.8	6.0622	3	222	18.7	
4	0.06905	0.0	2.18	0	0.458	7.147	54.2	6.0622	3	222	18.7	

	lstat	medv
0	4.98	24.0
1	9.14	21.6
2	4.03	34.7
3	2.94	33.4
4	5.33	36.2

1.14 Question 11a.

Try out some of the regression methods explored in this chapter, such as best subset selection, the lasso, ridge regression, and PCR. Present and discuss results for the approaches that you consider.

[44]:
```
from sklearn.model_selection import train_test_split

from sklearn.linear_model import RidgeCV
from sklearn.pipeline import make_pipeline
from sklearn.preprocessing import StandardScaler
import numpy as np

from sklearn.cross_decomposition import PLSRegression
from sklearn.model_selection import cross_val_score
from sklearn.pipeline import Pipeline

# Define predictors and response
X = boston.drop(columns='crim')
y = boston['crim']

# Train-test split (70/30)
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.3, random_state=1
)

from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error

# Fit and predict least squares
lm = LinearRegression()
lm.fit(X_train, y_train)
y_pred_lm = lm.predict(X_test)

# Evaluate least squares
mse_lm = mean_squared_error(y_test, y_pred_lm)
print(f"Test MSE (Least Squares): {mse_lm:.4f}")

# try ridge regression
ridge_model = make_pipeline(
    StandardScaler(),
    RidgeCV(alphas=10**np.linspace(10, -2, 100), store_cv_results=True)
)

ridge_model.fit(X_train, y_train)
y_pred_ridge = ridge_model.predict(X_test)

mse_ridge = mean_squared_error(y_test, y_pred_ridge)
best_lambda_ridge = ridge_model.named_steps['ridgecv'].alpha_
```

```
print(f"\nTest MSE (Ridge): {mse_ridge:.4f}")
print(f"Optimal lambda (Ridge): {best_lambda_ridge:.4f}")


# Try PLS
m_values = list(range(1, X_train.shape[1] + 1))
cv_errors = []

for m in m_values:
    pls_pipeline = Pipeline([
        ('scale', StandardScaler()),
        ('pls', PLSRegression(n_components=m))
    ])
    scores = cross_val_score(pls_pipeline, X_train, y_train,
                             scoring='neg_mean_squared_error', cv=10)
    cv_errors.append(-scores.mean())

# Choose best M
best_m = m_values[np.argmin(cv_errors)]
print(f"\nBest number of components (PLS): {best_m}")

# Fit final model
final_pls = Pipeline([
    ('scale', StandardScaler()),
    ('pls', PLSRegression(n_components=best_m))
])
final_pls.fit(X_train, y_train)
y_pred_pls = final_pls.predict(X_test)
mse_pls = mean_squared_error(y_test, y_pred_pls)

print(f"Test MSE (PLS): {mse_pls:.4f}")
```

```
Test MSE (Least Squares): 50.5105

Test MSE (Ridge): 50.3006
Optimal lambda (Ridge): 4.6416

Best number of components (PLS): 9
Test MSE (PLS): 50.5177
```

All three models predict per capita crime rate with similar accuracy. Ridge performed marginally better, suggesting that slight coefficient limiting helps stabilize the model. PLS didn't outperform despite reducing dimensionality, likely because most predictors carry some useful signal.

1.15 Question 11b.

Propose a model (or set of models) that seem to perform well on this data set, and justify your answer. Make sure that you are evaluating model performance using validation set error, cross-validation, or some other reasonable alternative, as opposed to using training error.

[45]:

```python
# Let's propose PCR and if poor performance, we'll go with ridge regression

from sklearn.decomposition import PCA
from sklearn.linear_model import LinearRegression
from sklearn.pipeline import Pipeline
from sklearn.model_selection import cross_val_score
from sklearn.metrics import mean_squared_error
import numpy as np

m_values = list(range(1, X_train.shape[1] + 1))  # Try 1 to 13 components
cv_errors = []

for m in m_values:
    pcr_pipeline = Pipeline([
        ('scale', StandardScaler()),
        ('pca', PCA(n_components=m)),
        ('linreg', LinearRegression())
    ])

    scores = cross_val_score(pcr_pipeline, X_train, y_train,
                             scoring='neg_mean_squared_error', cv=10)
    cv_errors.append(-scores.mean())

best_m = m_values[np.argmin(cv_errors)]
print(f"Best number of components (PCR): {best_m}")

# Final model with best M
final_pcr = Pipeline([
    ('scale', StandardScaler()),
    ('pca', PCA(n_components=best_m)),
    ('linreg', LinearRegression())
])

final_pcr.fit(X_train, y_train)
y_pred_pcr = final_pcr.predict(X_test)
mse_pcr = mean_squared_error(y_test, y_pred_pcr)

print(f"Test MSE (PCR): {mse_pcr:.4f}")
```

```
Best number of components (PCR): 12
Test MSE (PCR): 50.5105
```

Even though PCR used one fewer component than the full 13, the last principal component (PC13) must have contributed very little predictive value, so the regression performance was effectively the same as OLS with all 13 original predictors.

Proposed Model is Ridge Regression Among the regression approaches tested, ridge regression produced the lowest test mean squared error (MSE), making it the strongest candidate for modeling per capita crime rate in the Boston dataset.

1.16 Question 11c.

Does your chosen model involve all of the features in the dataset? Why or why not?

Ridge regression includes all features in the model. While it penalizes large coefficients via regularization, it does not eliminate predictors altogether like other methods such as lasso. This is useful when all features are believed to carry some predictive information, which seems to be the case here.