Solutions 11

Jumping Rivers

Penalised Regression

To really see how the effect of penalised regression we are going to use some synthetic data. The `sklearn.datasets` module has some data generators which are useful for exploring different modelling techniques. The reason this is useful is that by construction we know what we should be expecting to find when using a given technique.

```
from sklearn.datasets import make_regression

X, y, ground_truth = make_regression(
  n_samples=1000, n_features=1000,
  n_informative=100, effective_rank=20,
  random_state=2019, noise=0.2, coef=True
)
```

This code generates some synthetic data that has a large number of useless features and correlation amongst the input variables. This is a situation where we expect standard linear regression to perform poorly.

a) Build a standard linear regression model, remember to include some preprocessing

```
from sklearn.linear_model import LinearRegression
from sklearn.preprocessing import StandardScaler
from sklearn.pipeline import Pipeline

model = Pipeline([
  ('prep', StandardScaler()),
  ('model', LinearRegression())
])

model.fit(X, y)
```

```
## Pipeline(memory=None,
##          steps=[('prep',
##                  StandardScaler(copy=True, with_mean=True, with_std=True)),
##                 ('model',
##                  LinearRegression(copy_X=True, fit_intercept=True, n_jobs=None,
##                                   normalize=False))],
##          verbose=False)
```

```
linear_coefs = model.named_steps['model'].coef_
```

b) Use 10 fold cross validation to estimate mean squared error

```
from sklearn.model_selection import cross_validate
from sklearn.metrics import mean_squared_error, make_scorer

score_fn = make_scorer(mean_squared_error)

linear_model = cross_validate(
  model, X, y,
  scoring=score_fn, cv=10,
  return_train_score=False
)

linear_model['test_score'].mean()
```

```
## 0.30027559717499475
```

c) In a similar fashion to the notes on Lasso, can you find a lasso regression model that performs better than this linear regression

```
from sklearn.linear_model import LassoCV, Lasso

model = Pipeline([
  ('prep', StandardScaler()),
  ('model', LassoCV(cv=10))
])

model.fit(X, y)
```

```
## Pipeline(memory=None,
##          steps=[('prep',
##                  StandardScaler(copy=True, with_mean=True, with_std=True)),
##                 ('model',
##                  LassoCV(alphas=None, copy_X=True, cv=10, eps=0.001,
##                          fit_intercept=True, max_iter=1000, n_alphas=100,
##                          n_jobs=None, normalize=False, positive=False,
##                          precompute='auto', random_state=None,
##                          selection='cyclic', tol=0.0001, verbose=False))],
##          verbose=False)
```

```
param_alpha = model.named_steps['model'].alpha_

lasso_coef = model.named_steps['model'].coef_

lasso_pipe = Pipeline([
 ('prep', StandardScaler()),
```

```
  ('model', Lasso(alpha=param_alpha))
])

lasso_model = cross_validate(
  lasso_pipe, X, y,
  scoring = score_fn,
  cv = 10
)

lasso_model['test_score'].mean()

# performance is much better
# percentage reduction in average error
```

```
## 0.06858787160554028
```

```
(100*(lasso_model['test_score'].mean() - linear_model['test_score'].mean())/
  linear_model['test_score'].mean())
```

```
## -77.15835976988545
```

d) How do the two sets of coefficients compare?

```
linear_coefs[:5]
```

```
## array([0.09034486, 0.03881794, 0.05171983, 0.34117376, 0.18255434])
```

```
lasso_coef[:5]

# many coefficients are set to zero.
```

```
## array([-0.        , -0.01006167,  0.01686888,  0.23801812, -0.01664323])
```

```
import numpy as np
# percentage coefficients set to zero
100* np.sum(lasso_coef ==0)/1000

# huge reduction in dimension as well as improved predictive performance
```

```
## 68.9
```

e) How do ridge and elastic net compare?