Assignment 4 answer

June 22, 2021

```
In [2]: import numpy as np
        import pandas as pd
        import matplotlib.pyplot as plt
```

0.1 22.1a

```
In [3]: np.random.seed(17)
        n = 500
        k = 100
        x = np.random.normal(size=(n, k))
        y = np.random.normal(size=(n, 1))
```

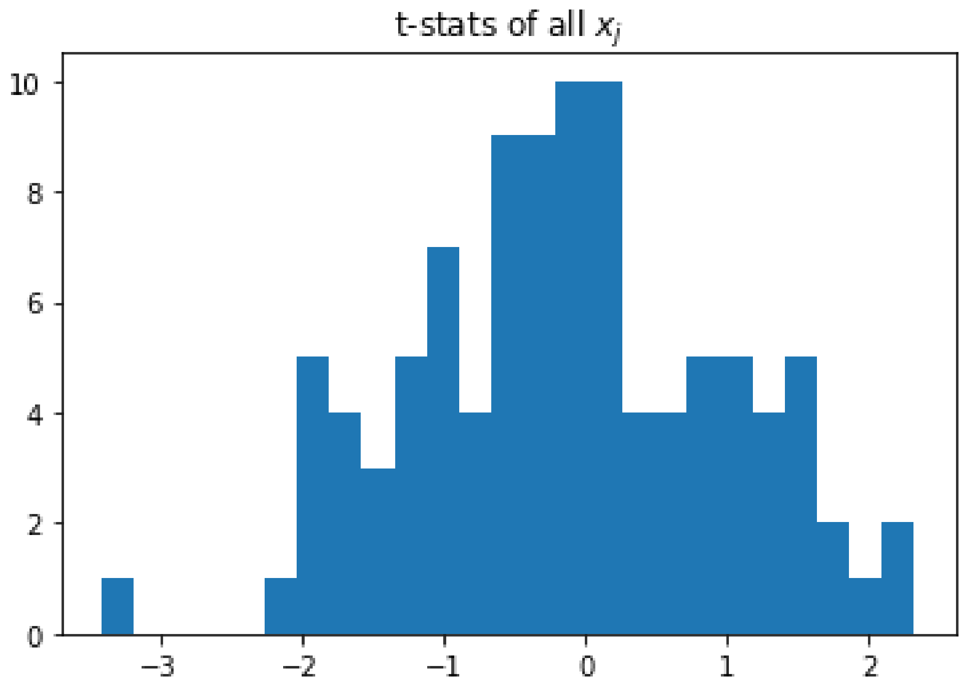
```
In [4]: def regress(y, X):
            B = np.linalg.inv(X.T@X) @ (X.T@y)
            yhat = X@B
            eps = y-yhat
            RSS = (eps**2).sum()
            RegSS = ((yhat-y.mean())**2).sum()
            TSS = ( (y-y.mean())**2 ).sum()
            R2 = RegSS / TSS
            n, k = X.shape
            F = (RegSS / k) / (RSS/(n-k-1))
            return B, R2, RSS, F, RegSS
```

```
In [5]: def calc_tstats(y, x, B, RSS):
            t_stats = []
            n, k = x.shape
            all_k = list(range(k))
            for j in all_k:
                xj = x[:,j]
                not_j = list(all_k)
                not_j.remove(j)
                not_xj = x[:,not_j]
                B_xj, R2_xj, RSS_xj, _, _ = regress(xj, not_xj)
                VIF = 1/(1-R2_xj)
                spread_xj = ((xj - xj.mean())**2).sum()
                VAR_B_xj = VIF * (RSS_xj/(n-k-1)) / spread_xj
                SE_B_xj = np.sqrt(VAR_B_xj)
```

```
            t_stats.append(B[j] / SE_B_xj)
        return np.array(t_stats)
```

In [6]:
```
B, R2, RSS, F, RegSS = regress(y, x)
t_stats = calc_tstats(y, x, B, RSS)
```

In [7]:
```
i_stat_sig = np.where(np.abs(t_stats) > 1.96)[0]
plt.hist(t_stats,25);
plt.title('t-stats of all $x_j$');
```



In [8]:
```
print (f"F-stat = {F:.2f} is about 1, not significant, but maybe large enough to encou
print (f"The statistically significant regressors are {i_stat_sig}")
```

```
F-stat = 1.01 is about 1, not significant, but maybe large enough to encourage us to look for a
The statistically significant regressors are [ 4  9 22 25 58 79]
```

All of the true betas a zero. We always expect 5% of the betas to be statistically significant at the 5% level. With seed=17 we found 6 (6% of 100) statistically significant betas. Other seeds will show other numbers of regressors as statistically significant, but there will always be around 5%.

0.2 22.1b

In [233]:
```
i_b3 = np.argsort(-np.abs(t_stats).flatten())[:3]
print (i_stat_sig)
print (t_stats[i_b3])
```

```
[ 4  9 22 25 58 79]
[[-3.40448037]
 [ 2.318819  ]
 [ 2.29023479]]
```

```
In [333]: x_b3 = x[:, i_b3]
          B_b3, R2_b3, RSS_b3, F_b3, _ = regress(y, x_b3)
          t_stats_b3 = calc_tstats(y, x_b3, B_b3, RSS_b3)
```

```
In [239]: print (F_b3)
          print (t_stats_b3)
```

```
6.660222989856333
[[-3.65274909]
 [ 2.22876063]
 [ 2.38516563]]
```

The F statistic (F_b3) is larger, indicating a better-fitting model with just these regressors. The t statistics for the betas are similar in this fit to their values in the full 100-regressor regression.

0.3 22.1c

```
In [312]: def best_F_regressor(y, x, j_so_far):
              n, k = x.shape
              F_max = 0
              j_max = None
              if len(j_so_far) > 0:
                  RegSS0 = regress(y, x[:,j_so_far])[-1]
              else:
                  RegSS0 = 0
              q = 1  # checking 1 regressor at a time
              for j in range(x.shape[1]):
                  if j in j_so_far:
                      continue
                  j_use = j_so_far + [j]
                  B, R2, RSS, F, RegSS = regress(y, x[:,j_use])
                  F = (RegSS - RegSS0)/q / (RSS/(n-k-1))
                  if F > F_max:
                      F_max = F
                      j_max = j
                      R2_adj = 1 - (n-1)*RSS / (n-len(j_use)) / (RSS+RegSS)
              return j_max, F_max, R2_adj

          def forward_stepwise(y, x, max_k):
              n, k = x.shape
              j_kept = []
              F_base = 0
```

```
        for j in range(k):
            j_max, F_max, R2_adj = best_F_regressor(y, x, j_kept)
            if len(j_kept) == max_k:
                break
            j_kept += [j_max]
            F_base = F_max
        return j_kept, R2_adj
```

In [313]:
```
j_top3, _ = forward_stepwise(y, x, max_k=3)
```

In [332]:
```
B_top3, R2, RSS, F, RegSS = regress(y, x[:,j_top3])
t_stats = calc_tstats(y, x[:,j_top3], B, RSS)
print(f"F = {F:.2f}")
print(f"t_stats = {t_stats}")
```

```
F = 7.18
t_stats = [[-3.90026502]
 [ 2.01601123]
 [ 2.25602031]]
```

The F statistic is not strongly significant and suggests we keep the three regressors chosen by the forward stepwise procedure. The t statistics are high and comparable to the best regressors from part (a).

0.4 22.1d

In [324]:
```
def forward_stepwise_best(y, x):
    n, k = x.shape
    j_kept = []
    R2_adj_last = 0
    for j in range(k):
        j_max, F_max, R2_adj = best_F_regressor(y, x, j_kept)
        if R2_adj <= R2_adj_last:
            break
        R2_adj_last = R2_adj
        j_kept += [j_max]

    return j_kept, R2_adj_last
```

In [350]:
```
j_r2_adj, R2_adj_last = forward_stepwise_best(y,x)
print (len(j_r2_adj))
```

```
34
```

In [351]:
```
B_r2_adj, R2, RSS, F, RegSS = regress(y, x[:,j_r2_adj])
t_stats = calc_tstats(y, x[:,j_kept], B, RSS)
print(f"F = {F:.2f}")
print(f"t_stats = {t_stats}")
```

```
F = 2.67
t_stats = [[-3.77628592]
 [ 1.95295005]
 [ 2.18526096]
 [-1.68947975]
 [-2.0835613 ]
 [ 2.32649287]
 [-2.25862687]
 [-2.22376471]
 [ 1.78893931]
 [-1.17926446]
 [-1.53051012]
 [-1.65812825]
 [ 1.37075733]
 [ 1.79643864]
 [-1.79871819]
 [-1.79578572]
 [ 1.77956642]
 [-1.57654455]
 [-1.60233174]
 [ 1.35014903]
 [ 1.63796439]
 [ 1.27146355]
 [ 1.28790167]
 [-1.41386298]
 [ 1.37239095]
 [ 1.38955949]
 [ 1.27722197]
 [-1.70751694]
 [ 1.29860346]
 [-1.41485975]
 [-1.2654294 ]
 [ 1.27039077]
 [-1.31506008]
 [-1.00292759]]
```

Selecting the model based purely on adjusted R^2 produces a model with lowest (absolute) t statistics for B, and a lower F statistic than for the top-3 model of 22.1c.

0.5 22.1e

```
In [329]: np.random.seed(19)
          x_val = np.random.normal(size=(n, k))
          y_val = np.random.normal(size=(n, 1))
```

```
In [346]: def validate(y_val, x_val, B):
              y_hat = x_val@B
```

```
E = (y-y_hat)
SSE = (E**2).sum()
TSS = ((y-y.mean())**2).sum()
corr = np.corrcoef(y_hat.flatten(), y.flatten())[0,1]
R2 = 1 - SSE/TSS
print (f"SSE = {SSE:.4f} corr = {corr:.2f} R2 = {R2:.4f}")
```

```
In [347]: validate(y_val, x_val[:,i_b3], B_b3)
```

```
SSE = 582.0677 corr = 0.01 R2 = -0.0406
```

```
In [349]: validate(y_val, x_val[:,j_top3], B_top3)
```

```
SSE = 577.3831 corr = 0.03 R2 = -0.0322
```

```
In [352]: validate(y_val, x_val[:,j_r2_adj], B_r2_adj)
```

```
SSE = 676.5546 corr = 0.01 R2 = -0.2095
```

None of these models work out of sample (in the validation set) because the data generating process was pure noise. There is no correlation between y and the x's. All of the true betas are 0.

0.6 22.1f

This notebook can be run with different seeds. The results are qualitatively similar each time:

- There is no structure in this data. The true betas are all zero, but about 5% of the regressors will look statistically significant at the p=.05 level. If we had asked for p=.10 for significance, then we would have seen 10% of the regressors being significant. The p value is the fraction of false positives we're willing to let through when the null hypothesis holds.
- When we distill the model down to its best-in-the-fit-data regressors by any F- or t- based variable-selection method (or AIC or BIC, as some people explored) we get a better F statistic and R2, too. The refined (few best regressors) model is a good fit to the in-sample data.
- If we test on a validation set (out-of-sample) we see that the model fails. It is a good idea to validate a model on out-of-sample data.