

In-Class Assignment 7 - Linear Model Selection

- What do I do with all of these variables?
- If I have p variables, how many combinations must I consider?
 - $2^p - 1024$ if $p = 10, 1,073,741,824$ if $p = 30!$
- What is happening to my model evaluation statistics as the number of predictors increases?
 - RSS ↓
 - $R^2 \uparrow$

$$RSS = \sum_{i=1}^n e_i^2 = \sum_{i=1}^n (y_i - \hat{y}_i)^2 \quad (1)$$

$$\hat{\sigma} = RSE = \sqrt{RSS/(n - p - 1)} \quad (2)$$

$$TSS = \sum_{i=1}^n (y_i - \bar{y})^2 \quad (3)$$

$$R^2 = \frac{TSS - RSS}{TSS} \quad (4)$$

$$F = \frac{(TSS - RSS) / p}{RSS / (n - p - 1)} \quad (5)$$

- As usual, our problem is the propensity to *overfit* with more flexibility in our model. Additional predictors give our methods more parameters to fit to the training data, but we want to perform well on the test data.
 - We can indirectly or directly measure test error:
 - Indirect measurements:
 - $AIC \sim C_p = \frac{1}{n} (RSS + 2p\hat{\sigma}^2)$
 - $BIC = \frac{1}{n} (RSS + \log(n)p\hat{\sigma}^2)$
 - BIC will select smaller models because of the $\log(n)$ penalty term
 - Adjusted $R^2 = 1 - \frac{RSS/(n-d-1)}{TSS/(n-1)}$
 - Recall R^2 above.
 - Adjusted version is attempting to capture something about test error, but C_p , AIC, and BIC are preferred for this use.
 - What about the logistic regression case?
 - We use the *deviance* metric: $-2 * LL$
 - Direct method - see cross-validation methods from Module 6!
- Can we systematically search the space of predictors?

- Yes, we can find the optimal number by finding the best subset of k predictors for each possible k , then selecting the best k
- We can also perform *stepwise* techniques which *guide* the search.
- *Forward Selection:*
 1. Let \mathcal{M}_0 be the null model with only the intercept.
 2. For $k = 0, 1, \dots, p - 1$
 - A. Fit all $p - k$ models that add one predictor to the current k predictors (model \mathcal{M}_k)
 - B. Pick the best model using a training error metric (RSS or R^2)
 3. Pick from among the p models using a test error method or metric (see above.)
- *Backward Selection:*
 1. Let \mathcal{M}_p be the full model with all considered predictors.
 2. For $k = p, p - 1, \dots, 1$
 - A. Fit all k models that remove one predictor to the current k predictors (model \mathcal{M}_k)
 - B. Pick the best model using a training error metric (RSS or R^2)
 3. Pick from among the p models using a test error method or metric (see above.)
- Of course we can create a hybrid of these two approaches.
 - One example: for each forward step perform a backwards step
- We can also add cross-validation to these procedures:
 - In Step 2. we will have a different model for each fold.
 - In Step 3, we average the validation error across all folds. When the best k is chosen, we can then refit the model using all data.

```
In [1]: for p in range(31):  
    print(p, 2**p)
```

```
0 1
1 2
2 4
3 8
4 16
5 32
6 64
7 128
8 256
9 512
10 1024
11 2048
12 4096
13 8192
14 16384
15 32768
16 65536
17 131072
18 262144
19 524288
20 1048576
21 2097152
22 4194304
23 8388608
24 16777216
25 33554432
26 67108864
27 134217728
28 268435456
29 536870912
30 1073741824
```

```
In [2]: import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import statsmodels.api as sm
from statsmodels.formula.api import ols, logit
from statsmodels.api import OLS
from sklearn.model_selection import train_test_split
from statsmodels.stats.outliers_influence import variance_inflation_factor
from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay
from sklearn.model_selection import ShuffleSplit
import scipy.stats as ss
import seaborn as sns
from itertools import product
from sklearn.metrics import roc_auc_score, roc_curve, accuracy_score
import warnings
import sklearn.model_selection as skm
from sklearn.preprocessing import StandardScaler
from copy import copy, deepcopy

# Turn off pandas warnings
warnings.filterwarnings("ignore", category=pd.errors.SettingWithCopyWarning)

mdf = pd.read_excel('../data/Case_Study_2.xlsx', sheet_name='Data')
```

```
mdf_train = mdf.loc[~mdf['Magazine Sales (Units)'].isna()]
mdf_train
```

Out[2]:

Index		Opponent	Magazine Sales (Units)	Week In Season	Opponent Preseason Rank	Preseason Ticket Sales	CSU Preseason Rank	Throw Jersey Yes
0	1	Cedar Falls University	4165.0	2	120	47420	21	
1	2	Oklahoma A&M	3746.0	3	58	47420	21	
2	3	Urbana College	4943.0	5	67	47420	21	
3	4	University of Bloomington	2366.0	9	83	47420	21	
4	5	Indiana A&M	1796.0	10	74	47420	21	
5	6	Minneapolis State University	1979.0	13	68	47420	21	
6	7	Mt Pleasant College	3866.0	1	109	46198	57	
7	8	University of Ames	5194.0	2	99	46198	57	
8	9	Ann Arbor University	1909.0	5	6	46198	57	
9	10	Evanston University	2523.0	6	55	46198	57	
10	11	Madison University	2734.0	8	17	46198	57	
11	12	Columbus University	2034.0	11	3	46198	57	
12	13	Lincoln University	6463.0	1	6	44211	73	
13	14	DeKalb College	3128.0	3	99	44211	73	
14	15	Pennsylvania A&M	2972.0	6	1	44211	73	
15	16	University of Bloomington	2158.0	8	68	44211	73	
16	17	Urbana College	2120.0	10	78	44211	73	
17	18	Minneapolis State University	1434.0	12	38	44211	73	

Index	Opponent	Magazine Sales (Units)	Week In Season	Opponent Preseason Rank	Preseason Ticket Sales	CSU Preseason Rank	Throw Jersey Yes
18	19 University of Kalamazoo	2691.0	2	83	37851	71	
19	20 University of Ames	3202.0	3	79	37851	71	
20	21 Michigan A&M	1669.0	6	24	37851	71	
21	22 Columbus University	2194.0	8	15	37851	71	
22	23 Madison University	1536.0	9	4	37851	71	
23	24 Evanston University	763.0	11	94	37851	71	
24	25 Ohio A&M	3059.0	1	109	40549	76	
25	26 Northern Cincinnati University	2390.0	2	79	40549	76	
26	27 Pennsylvania A&M	3056.0	5	39	40549	76	
27	28 University of Bloomington	2298.0	8	93	40549	76	
28	29 Ann Arbor University	2956.0	9	17	40549	76	
29	30 Minneapolis State University	2324.0	12	59	40549	76	
30	31 LBJ University	2885.0	1	84	41362	46	
31	32 University of Ames	3677.0	3	52	41362	46	
32	33 University of Logan	1911.0	4	114	41362	46	
33	34 Indiana A&M	2404.0	6	22	41362	46	
34	35 Michigan A&M	2113.0	7	23	41362	46	
35	36 Madison University	2345.0	10	32	41362	46	

Index	Opponent	Magazine Sales (Units)	Week In Season	Opponent Preseason Rank	Preseason Ticket Sales	CSU Preseason Rank	Throw Jersey Yes
36	37 Evanston University	1969.0	11	69	41362	46	
37	38 Northern Cincinnati University	3634.0	1	68	42843	31	
38	39 Western New York University	2500.0	2	117	42843	31	
39	40 University of Tempe	2810.0	4	29	42843	31	
40	41 Ann Arbor University	3961.0	6	16	42843	31	
41	42 Pennsylvania A&M	2440.0	9	44	42843	31	
42	43 Urbana College	2017.0	10	45	42843	31	
43	44 Minneapolis State University	2173.0	12	26	42843	31	
44	45 Ohio A&M	4382.0	1	95	47035	19	
45	46 University of Ames	4037.0	2	77	47035	19	
46	47 Michigan A&M	2930.0	5	37	47035	19	
47	48 Columbus University	2757.0	7	11	47035	19	
48	49 Indiana A&M	2678.0	10	32	47035	19	
49	50 Madison University	2445.0	12	15	47035	19	
50	51 Letterman University	2985.0	1	111	54584	19	
51	52 Cedar Falls University	2800.0	3	120	54584	11	

Index	Opponent	Magazine Sales (Units)	Week In Season	Opponent Preseason Rank	Preseason Ticket Sales	CSU Preseason Rank	Throw Jersey Yes
52	53	Urbana College	2910.0	5	69	54584	11
53	54	University of Bloomington	2500.0	7	86	54584	11
54	55	Ann Arbor University	3721.0	8	7	54584	11
55	56	Minneapolis State University	2500.0	12	23	54584	11

In [3]:

```

rename_dict = {
    'Year': 'Year',
    'Opponent': 'Opponent',
    'Magazine Sales (Units)': 'Sales',
    'Week In Season': 'Week_in_Season',
    'Opponent Preseason Rank': 'Opponent_Preseason_Rank',
    'Preseason Ticket Sales': 'Preseason_Ticket_Sales',
    'CSU Preseason Rank': 'CSU_Preseason_Rank',
    'Throwback Jersey (1 = Yes; 0 = No)': 'Throwback_Jersey',
    'Kickoff Temperature': 'Kickoff_Temperature',
    'Home Game Number': 'Home_Game_Number',
    'Conference Game (1 = Yes; 0 = No)': 'Conference_Game',
    'Homecoming (1 = Yes; 0 = No)': 'Homecoming',
    'Game Day Weather': 'Game_Day_Weather',
    "Opponent's Previous Season Number of Wins": "Opponent_Prev_Wins",
    "Opponent's Previous Season Number of Losses": "Opponent_Prev_Losses",
    "CSU's Previous Season Number of Wins": "CSU_Prev_Wins",
    "CSU's Previous Season Number of Losses": "CSU_Prev_Losses"
}
mdf_train.rename(rename_dict, axis=1, inplace=True)
mdf_train = mdf_train[[x for x in list(rename_dict.values())]]
mdf_train.dtypes

```

```
Out[3]: Year           int64
Opponent        object
Sales          float64
Week_in_Season   int64
Opponent_Presseason_Rank int64
Preseason_Ticket_Sales   int64
CSU_Presseason_Rank     int64
Throwback_Jersey       int64
Kickoff_Temperature    float64
Home_Game_Number      int64
Conference_Game       int64
Homecoming          int64
Game_Day_Weather      object
Opponent_Prev_Wins     int64
Opponent_Prev_Losses    int64
CSU_Prev_Wins         int64
CSU_Prev_Losses        int64
dtype: object
```

```
In [4]: mdf_train['Opponent'] = mdf_train['Opponent'].str.replace(' ', '').str.replace(mdf_train = pd.get_dummies(mdf_train, columns=['Opponent', 'Game_Day_Weather'])
mdf_train.head()
```

```
Out[4]:   Year  Sales  Week_in_Season  Opponent_Presseason_Rank  Preseason_Ticket_Sales
0      1  4165.0              2                  120            47420
1      1  3746.0              3                  58             47420
2      1  4943.0              5                  67             47420
3      1  2366.0              9                  83             47420
4      1  1796.0             10                  74             47420
```

5 rows × 40 columns

```
In [5]: mdf_train.dtypes
```

```
Out[5]: Year                      int64
Sales                     float64
Week_in_Season            int64
Opponent_Preseason_Rank   int64
Preseason_Ticket_Sales    int64
CSU_Preseason_Rank        int64
Throwback_Jersey          int64
Kickoff_Temperature       float64
Home_Game_Number          int64
Conference_Game           int64
Homecoming                int64
Opponent_Prev_Wins        int64
Opponent_Prev_Losses      int64
CSU_Prev_Wins             int64
CSU_Prev_Losses           int64
Opponent_CedarFallsUniversity int64
Opponent_ColumbusUniversity int64
Opponent_DeKalbCollege    int64
Opponent_EvanstonUniversity int64
Opponent_IndianaAM        int64
Opponent_LBJUniversity    int64
Opponent_LettermanUniversity int64
Opponent_LincolnUniversity int64
Opponent_MadisonUniversity int64
Opponent_MichiganAM       int64
Opponent_MinneapolisStateUniversity int64
Opponent_MtPleasantCollege int64
Opponent_NorthernCincinnatiUniversity int64
Opponent_OhioAM            int64
Opponent_OklahomaAM       int64
Opponent_PennsylvaniaAM   int64
Opponent_UniversityofAmes int64
Opponent_UniversityofBloomington int64
Opponent_UniversityofKalamazoo int64
Opponent_UniversityofLogan int64
Opponent_UniversityofTempe int64
Opponent_UrbanaCollege    int64
Opponent_WesternNewYorkUniversity int64
Game_Day_Weather_Rain     int64
Game_Day_Weather_Sunny     int64
dtype: object
```

```
In [6]: y_var = 'Sales'
x_vars = [x for x in mdf_train.columns if x != y_var]
```

```
In [7]: def get_results(x_vars, y_var='Sales', data=mdf_train):
    model_string = f'{y_var} ~ {x + ".join(x_vars)}'
    model = ols(model_string, data=data)
    results = model.fit()
    return results

results = get_results(x_vars)
results.summary()
```

Out[7]:

OLS Regression Results

Dep. Variable:	Sales	R-squared:	0.910			
Model:	OLS	Adj. R-squared:	0.708			
Method:	Least Squares	F-statistic:	4.507			
Date:	Wed, 02 Oct 2024	Prob (F-statistic):	0.000807			
Time:	11:39:04	Log-Likelihood:	-397.86			
No. Observations:	56	AIC:	873.7			
Df Residuals:	17	BIC:	952.7			
Df Model:	38					
Covariance Type:	nonrobust					
		coef	std err	t	P> t	[0.02
	Intercept	-568.2598	4500.296	-0.126	0.901	-1.01e+0
	Year	-129.7111	61.255	-2.118	0.049	-258.94
	Week_in_Season	-197.8443	161.767	-1.223	0.238	-539.14
	Opponent_Presseason_Rank	-8.5929	9.909	-0.867	0.398	-29.49
	Presseason_Ticket_Sales	-0.0224	0.042	-0.527	0.605	-0.11
	CSU_Presseason_Rank	-28.2341	16.900	-1.671	0.113	-63.89
	Throwback_Jersey	679.4031	858.371	0.792	0.440	-1131.60
	Kickoff_Temperature	12.6243	11.537	1.094	0.289	-11.71
	Home_Game_Number	142.5779	346.851	0.411	0.686	-589.21
	Conference_Game	783.8839	885.088	0.886	0.388	-1083.48
	Homecoming	-102.2743	316.844	-0.323	0.751	-770.75
	Opponent_Prev_Wins	196.1953	249.033	0.788	0.442	-329.21
	Opponent_Prev_Losses	349.8304	294.334	1.189	0.251	-271.15
	CSU_Prev_Wins	221.2822	341.366	0.648	0.525	-498.93
	CSU_Prev_Losses	339.6795	436.956	0.777	0.448	-582.21
	Opponent_CedarFallsUniversity	425.2742	856.567	0.496	0.626	-1381.92
	Opponent_ColumbusUniversity	-1028.0521	540.699	-1.901	0.074	-2168.82
	Opponent_DeKalbCollege	33.2951	644.224	0.052	0.959	-1325.90
	Opponent_EvanstonUniversity	-1242.8776	637.964	-1.948	0.068	-2588.86
	Opponent_IndianaAM	-1292.0586	543.153	-2.379	0.029	-2438.01
	Opponent_LBJUniversity	-368.5468	616.381	-0.598	0.558	-1668.99

Opponent_LettermanUniversity	-335.4292	726.299	-0.462	0.650	-1867.78
Opponent_LincolnUniversity	2540.3362	888.617	2.859	0.011	665.51
Opponent_MadisonUniversity	-980.2745	532.721	-1.840	0.083	-2104.21
Opponent_MichiganAM	-1083.2691	512.716	-2.113	0.050	-2165.00
Opponent_MinneapolisStateUniversity	-568.7728	553.770	-1.027	0.319	-1737.12
Opponent_MtPleasantCollege	150.5184	670.976	0.224	0.825	-1265.11
Opponent_NorthernCincinnatiUniversity	13.7161	570.625	0.024	0.981	-1190.19
Opponent_OhioAM	-320.3756	748.364	-0.428	0.674	-1899.28
Opponent_OklahomaAM	434.6451	810.943	0.536	0.599	-1276.29
Opponent_PennsylvaniaAM	-403.4642	547.951	-0.736	0.472	-1559.54
Opponent_UniversityofAmes	336.4036	455.025	0.739	0.470	-623.61
Opponent_UniversityofBloomington	-634.1231	664.025	-0.955	0.353	-2035.09
Opponent_UniversityofKalamazoo	-572.3051	657.405	-0.871	0.396	-1959.30
Opponent_UniversityofLogan	-869.0397	666.713	-1.303	0.210	-2275.68
Opponent_UniversityofTempe	-1287.3982	819.568	-1.571	0.135	-3016.53
Opponent_UrbanaCollege	-818.9018	590.020	-1.388	0.183	-2063.73
Opponent_WesternNewYorkUniversity	-1533.2378	698.074	-2.196	0.042	-3006.04
Game_Day_Weather_Rain	-531.8545	506.662	-1.050	0.309	-1600.81
Game_Day_Weather_Sunny	354.4038	266.097	1.332	0.200	-207.01
Omnibus:	4.109	Durbin-Watson:	1.723		
Prob(Omnibus):	0.128	Jarque-Bera (JB):	4.527		
Skew:	0.048	Prob(JB):	0.104		
Kurtosis:	4.390	Cond. No.	3.35e+15		

Notes:

- [1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
- [2] The smallest eigenvalue is 1e-20. This might indicate that there are strong multicollinearity problems or that the design matrix is singular.

Question 1

- What problems might there be with this model, just thinking logically?

- Answer: We have a very high condition number that suggests that there is multicollinearity. It means some of the predictor variables are highly correlated with each other like for example Opponent_Prev_Wins and Opponent_Prev_Losses are likely to be negatively correlated. Hence it is difficult to ignore their individual effects on the dependent variable. Therefore the model is potentially unstable and hard to interpret.

```
In [8]: training_validation_sets = []
for fold in range(10):
    training_set, validation_set = train_test_split(mdf_train, test_size=.1,
    training_validation_sets.append((training_set, validation_set))
```

Forward Selection

```
In [12]: def forward_selection(x_vars, training_validation_sets, init_current_predictors):
    M_k = {
        i:
        {
            "predictors": [],
            "rsquared": [],
            "BIC": []
        }
        for i in range(len(x_vars))
    }

    for (training_set, validation_set) in training_validation_sets:
        if len(init_current_predictors) == 0:
            results = get_results(['1'], data=training_set)
            M_k[0]['predictors'].append(['1'])
            M_k[0]['rsquared'].append(results.rsquared)

            validation_results = get_results(['1'], data=validation_set)
            M_k[0]['BIC'].append(validation_results.bic)

        current_predictors = copy(init_current_predictors)
        for i in range(len(init_current_predictors) + 1, len(x_vars)):
            possible_predictors = [x for x in x_vars if x not in current_predictors]
            best_rsquared = 0.0
            best_x_var = ''
            for x_var in possible_predictors:
                results = get_results(current_predictors + [x_var], data=training_set)
                if results.rsquared > best_rsquared:
                    best_rsquared = copy(results.rsquared)
                    best_x_var = copy(x_var)

            current_predictors += [best_x_var]
            validation_results = get_results(current_predictors, data=validation_set)

            M_k[i]['predictors'].append(copy(current_predictors))
            M_k[i]['rsquared'].append(copy(best_rsquared))
```

```
M_k[i]['BIC'].append(copy(validation_results.bic))
return M_k
M_k = forward_selection(x_vars, training_validation_sets)
```

```
In [46]: def plot_M_k(M_k, metric='BIC'):
    ks = []
    metrics = []
    for k, v in M_k.items():
        ks.append(k)
        metrics.append(np.average(v[metric]))

    plt.scatter(x=ks, y=metrics)

plot_M_k(M_k)
```

KeyError Traceback (most recent call last)
Cell In[46], line 10
6 metrics.append(np.average(v[metric]))
8 plt.scatter(x=ks, y=metrics)
----> 10 plot_M_k(M_k)

Cell In[46], line 6, in plot_M_k(M_k, metric)
4 for k, v in M_k.items():
5 ks.append(k)
----> 6 metrics.append(np.average(v[metric]))
8 plt.scatter(x=ks, y=metrics)

KeyError: 'BIC'

```
In [15]: k = 11
best_rsquared = 0.0
best_predictors = []
for predictors in M_k[k]['predictors']:
    results = get_results(predictors)
    if results.rsquared > best_rsquared:
        best_rsquared = copy(results.rsquared)
        best_predictors = copy(predictors)

results = get_results(best_predictors)
results.summary()
```

Out[15]:

OLS Regression Results

Dep. Variable:	Sales	R-squared:	0.780			
Model:	OLS	Adj. R-squared:	0.725			
Method:	Least Squares	F-statistic:	14.17			
Date:	Wed, 02 Oct 2024	Prob (F-statistic):	4.24e-11			
Time:	11:41:08	Log-Likelihood:	-422.82			
No. Observations:	56	AIC:	869.6			
Df Residuals:	44	BIC:	893.9			
Df Model:	11					
Covariance Type:	nonrobust					
	coef	std err	t	P> t	[0.025	0.975]
Intercept	3526.4100	678.862	5.195	0.000	2158.253	4894.567
Kickoff_Temperature	17.6299	7.863	2.242	0.030	1.784	33.476
Opponent_LincolnUniversity	2756.0700	598.162	4.608	0.000	1550.554	3961.586
CSU_Preseason_Rank	-17.7102	3.578	-4.950	0.000	-24.921	-10.500
Home_Game_Number	-147.1215	66.554	-2.211	0.032	-281.251	-12.992
Opponent_Preseason_Rank	-6.6021	2.506	-2.634	0.012	-11.653	-1.551
Year	-104.2873	33.276	-3.134	0.003	-171.351	-37.223
Opponent_OhioAM	789.9541	404.684	1.952	0.057	-25.632	1605.541
Opponent_UniversityofAmes	969.0878	295.168	3.283	0.002	374.215	1563.960
Game_Day_Weather_Sunny	386.2703	183.703	2.103	0.041	16.041	756.499
Opponent_PennsylvaniaAM	476.8246	328.889	1.450	0.154	-186.008	1139.657
Opponent_IndianaAM	-387.6487	325.760	-1.190	0.240	-1044.175	268.878
Omnibus:	2.950	Durbin-Watson:	1.731			
Prob(Omnibus):	0.229	Jarque-Bera (JB):	2.840			
Skew:	0.525	Prob(JB):	0.242			
Kurtosis:	2.663	Cond. No.	984.			

Notes:

[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.

```
In [20]: temp = results.pvalues.reset_index().sort_values(0)
current_predictors = [x for x in temp.loc[temp[0] < .05]['index'].tolist() if x]
current_predictors
```

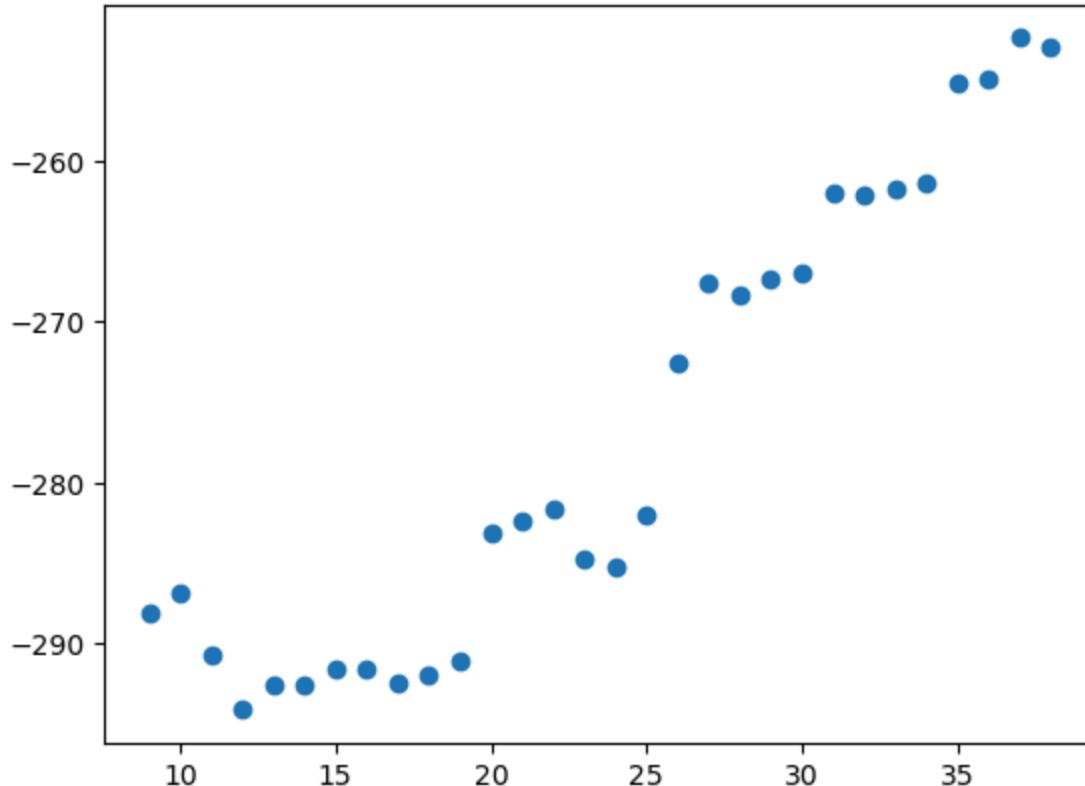
```
Out[20]: ['CSU_Preseason_Rank',
          'Opponent_LincolnUniversity',
          'Opponent_UniversityofAmes',
          'Year',
          'Opponent_Preseason_Rank',
          'Kickoff_Temperature',
          'Home_Game_Number',
          'Game_Day_Weather_Sunny']
```

```
In [21]: M_k = forward_selection(x_vars, training_validation_sets, init_current_predi
```

```
In [22]: plot_M_k(M_k)
```

```
/opt/anaconda3/lib/python3.12/site-packages/numpy/lib/function_base.py:520:
RuntimeWarning: Mean of empty slice.
    avg = a.mean(axis, **keepdims_kw)
/opt/anaconda3/lib/python3.12/site-packages/numpy/core/_methods.py:129: RuntimeWarning: invalid value encountered in scalar divide
    ret = ret.dtype.type(ret / rcount)
```

```
Out[22]: <matplotlib.collections.PathCollection at 0x30c8c4bf0>
```



```
In [23]: k = 12
best_rsquared = 0.0
best_predictors = []
for predictors in M_k[k]['predictors']:
    results = get_results(predictors)
```

```
if results.rsquared > best_rsquared:  
    best_rsquared = copy(results.rsquared)  
    best_predictors = copy(predictors)  
  
results = get_results(best_predictors)  
results.summary()
```

Out[23]:

OLS Regression Results

Dep. Variable:	Sales	R-squared:	0.794			
Model:	OLS	Adj. R-squared:	0.737			
Method:	Least Squares	F-statistic:	13.83			
Date:	Wed, 02 Oct 2024	Prob (F-statistic):	4.24e-11			
Time:	11:43:53	Log-Likelihood:	-420.93			
No. Observations:	56	AIC:	867.9			
Df Residuals:	43	BIC:	894.2			
Df Model:	12					
Covariance Type:	nonrobust					
		coef	std err	t	P> t	[0.025]
	Intercept	-2190.7723	2477.819	-0.884	0.382	-7187.770
	CSU_Preseason_Rank	-13.7709	9.533	-1.445	0.156	-32.996
	Opponent_LincolnUniversity	2957.8079	589.787	5.015	0.000	1768.390
	Opponent_UniversityofAmes	799.6692	287.184	2.785	0.008	220.508
	Year	-111.4989	36.991	-3.014	0.004	-186.098
	Opponent_Preseason_Rank	-4.3286	2.553	-1.695	0.097	-9.478
	Kickoff_Temperature	15.4184	7.824	1.971	0.055	-0.360
	Home_Game_Number	-182.6265	64.184	-2.845	0.007	-312.066
	Game_Day_Weather_Sunny	447.5209	177.410	2.523	0.015	89.739
	CSU_Prev_Wins	457.6159	185.367	2.469	0.018	83.788
	CSU_Prev_Losses	504.5384	247.548	2.038	0.048	5.310
	Opponent_WesternNewYorkUniversity	-876.8883	552.988	-1.586	0.120	-1992.094
	Opponent_UniversityofLogan	-776.4066	533.695	-1.455	0.153	-1852.705
Omnibus:	2.433	Durbin-Watson:	1.899			
Prob(Omnibus):	0.296	Jarque-Bera (JB):	1.853			
Skew:	0.442	Prob(JB):	0.396			
Kurtosis:	3.109	Cond. No.	3.68e+03			

Notes:

[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.

[2] The condition number is large, 3.68e+03. This might indicate that there are strong multicollinearity or other numerical problems.

Logistic Forward Selection

```
In [34]: from ISLP import load_data
Smarket = load_data('Smarket')
Smarket = pd.get_dummies(Smarket, columns=['Direction'], drop_first=True)
Smarket['Direction_Up'] = Smarket['Direction_Up'].astype(int)

train = Smarket.loc[Smarket['Year'] <= 2004]
test = Smarket.loc[Smarket['Year'] == 2005]

y_var = 'Direction_Up'
x_vars = [f'Lag{x}' for x in range(1, 6)]

training_validation_sets = []
for fold in range(10):
    training_set, validation_set = train_test_split(train, test_size=.2, shuffle=True)
    training_validation_sets.append((training_set, validation_set))

Smarket.head()
```

	Year	Lag1	Lag2	Lag3	Lag4	Lag5	Volume	Today	Direction_Up
0	2001	0.381	-0.192	-2.624	-1.055	5.010	1.1913	0.959	1
1	2001	0.959	0.381	-0.192	-2.624	-1.055	1.2965	1.032	1
2	2001	1.032	0.959	0.381	-0.192	-2.624	1.4112	-0.623	0
3	2001	-0.623	1.032	0.959	0.381	-0.192	1.2760	0.614	1
4	2001	0.614	-0.623	1.032	0.959	0.381	1.2057	0.213	1

```
In [78]: def get_metrics(results, validation_set, threshold=.5):
    validation_set['score'] = results.predict(validation_set)
    validation_set['pred'] = (validation_set['score'] > threshold).astype(int)
    accuracy = accuracy_score(validation_set[y_var], validation_set['pred'])

    fpr = ((validation_set['pred'] == 1) & (validation_set[y_var] == 0)).mean()
    tpr = ((validation_set['pred'] == 1) & (validation_set[y_var] == 1)).mean()
    fnr = ((validation_set['pred'] == 0) & (validation_set[y_var] == 1)).mean()
    tnr = ((validation_set['pred'] == 0) & (validation_set[y_var] == 0)).mean()

    auc_logistic = roc_auc_score(validation_set[y_var], validation_set['score'])

    return accuracy, fpr, tpr, fnr, tnr, auc_logistic

def get_results_logit(x_vars, y_var='Direction_Up', data=train):
    model_string = f'{y_var} ~ {x_vars}'.join(x_vars)
    model = logit(model_string, data=data)
    results = model.fit(disp=False)
```

```

    return results

def forward_selection_logit(x_vars, training_validation_sets, init_current_predictors):
    M_k = {
        i:
        {
            "predictors": [],
            "deviance": [],
            "validation_auc": []
        }
        for i in range(len(x_vars) + 1)
    }

    for (training_set, validation_set) in training_validation_sets:
        if len(init_current_predictors) == 0:
            results = get_results_logit(['1'], data=training_set)
            M_k[0]['predictors'].append(['1'])
            M_k[0]['deviance'].append(-2 * results.llf)

            validation_results = get_results_logit(['1'], data=validation_set)
            accuracy, fpr, tpr, fnr, tnr, auc_logistic = get_metrics(validation_results)
            M_k[0]['validation_auc'].append(auc_logistic)

        current_predictors = copy(init_current_predictors)

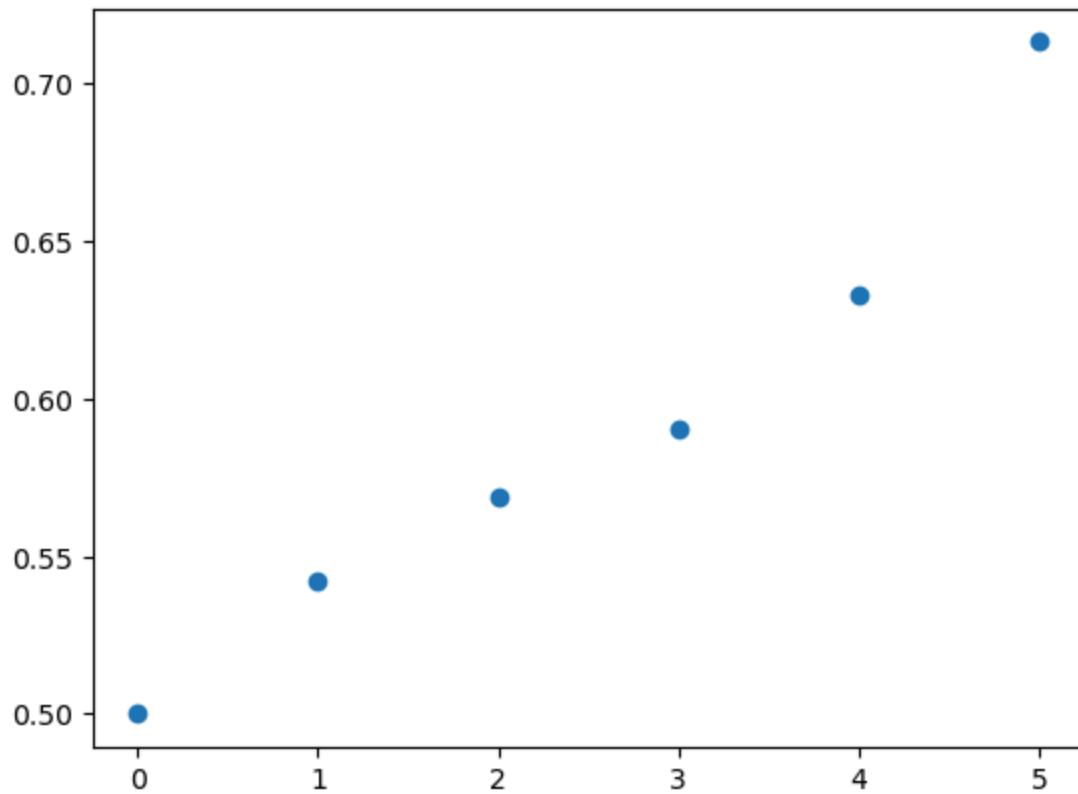
        for i in range(len(init_current_predictors) + 1, len(x_vars) + 1):
            possible_predictors = [x for x in x_vars if x not in current_predictors]
            best_deviance = np.inf
            best_x_var = ''
            for x_var in possible_predictors:
                results = get_results_logit(current_predictors + [x_var], data=training_set)
                if -2 * results.llf < best_deviance:
                    best_deviance = -2 * results.llf
                    best_x_var = copy(x_var)

            M_k[i]['deviance'].append(best_deviance)
            current_predictors += [best_x_var]
            validation_results = get_results_logit(current_predictors, data=validation_set)
            accuracy, fpr, tpr, fnr, tnr, auc_logistic = get_metrics(validation_results)
            M_k[i]['predictors'].append(copy(current_predictors))
            M_k[i]['validation_auc'].append(auc_logistic)

    return M_k
M_k = forward_selection_logit(x_vars, training_validation_sets)

```

In [79]: plot_M_k(M_k, metric='validation_auc')



```
In [80]: M_k[5]
```

```
Out[80]: {'predictors': [['Lag1', 'Lag3', 'Lag4', 'Lag5', 'Lag2'],  
    ['Lag4', 'Lag2', 'Lag3', 'Lag5', 'Lag1'],  
    ['Lag1', 'Lag3', 'Lag5', 'Lag2', 'Lag4'],  
    ['Lag1', 'Lag5', 'Lag4', 'Lag3', 'Lag2'],  
    ['Lag5', 'Lag1', 'Lag2', 'Lag3', 'Lag4'],  
    ['Lag3', 'Lag1', 'Lag5', 'Lag2', 'Lag4'],  
    ['Lag4', 'Lag1', 'Lag3', 'Lag5', 'Lag2'],  
    ['Lag1', 'Lag5', 'Lag2', 'Lag3', 'Lag4'],  
    ['Lag1', 'Lag5', 'Lag3', 'Lag4', 'Lag2']],  
    'deviance': [1067.8488807826457,  
    1081.2477749310501,  
    1064.638026313591,  
    1073.9629727619053,  
    1051.4475892270434,  
    1071.3474474898005,  
    1075.0904083018577,  
    1069.0201979654498,  
    1065.8379143305938,  
    1064.8038126809918],  
    'validation_auc': [0.6729824561403508,  
    0.7452631578947367,  
    0.7153884711779448,  
    0.7083833533413366,  
    0.6804549114331723,  
    0.6996193910256411,  
    0.7514005602240896,  
    0.682406015037594,  
    0.7052525252525252,  
    0.7710939845861275]}
```

```
In [81]: k = 5  
best_auc = 0.0  
best_predictors = []  
for predictors in M_k[k]['predictors']:  
    validation_results = get_results_logit(predictors, data=train)  
    accuracy, fpr, tpr, fnr, tnr, auc_logistic = get_metrics(validation_results)  
  
    if auc_logistic > best_auc:  
        best_auc = auc_logistic  
        best_predictors = copy(predictors)  
  
results = get_results_logit(best_predictors)  
results.summary()
```

Out[81]:

Logit Regression Results

Dep. Variable:	Direction_Up	No. Observations:	998				
Model:	Logit	Df Residuals:	966				
Method:	MLE	Df Model:	31				
Date:	Wed, 02 Oct 2024	Pseudo R-squ.:	0.02659				
Time:	13:54:10	Log-Likelihood:	-673.24				
converged:	True	LL-Null:	-691.63				
Covariance Type:	nonrobust	LLR p-value:	0.2187				
		coef	std err	z	P> z	[0.025	0.975]
Intercept	0.0251	0.065	0.386	0.700	-0.102	0.152	
Lag1	-0.0594	0.059	-1.007	0.314	-0.175	0.056	
Lag3	0.0038	0.057	0.066	0.947	-0.109	0.116	
Lag1:Lag3	0.0721	0.045	1.586	0.113	-0.017	0.161	
Lag4	0.0141	0.057	0.246	0.806	-0.098	0.127	
Lag1:Lag4	-0.0235	0.039	-0.595	0.552	-0.101	0.054	
Lag3:Lag4	-0.0181	0.044	-0.408	0.684	-0.105	0.069	
Lag1:Lag3:Lag4	-0.0041	0.029	-0.141	0.888	-0.061	0.053	
Lag5	-0.0149	0.056	-0.265	0.791	-0.125	0.095	
Lag1:Lag5	-0.0907	0.043	-2.089	0.037	-0.176	-0.006	
Lag3:Lag5	0.0466	0.040	1.178	0.239	-0.031	0.124	
Lag1:Lag3:Lag5	0.0285	0.031	0.929	0.353	-0.032	0.089	
Lag4:Lag5	0.0157	0.046	0.342	0.732	-0.074	0.105	
Lag1:Lag4:Lag5	0.0005	0.028	0.017	0.986	-0.055	0.056	
Lag3:Lag4:Lag5	0.0178	0.031	0.579	0.563	-0.043	0.078	
Lag1:Lag3:Lag4:Lag5	0.0437	0.021	2.068	0.039	0.002	0.085	
Lag2	-0.0787	0.058	-1.346	0.178	-0.193	0.036	
Lag1:Lag2	-0.0117	0.048	-0.244	0.808	-0.105	0.082	
Lag3:Lag2	0.0377	0.046	0.821	0.412	-0.052	0.128	
Lag1:Lag3:Lag2	0.0148	0.032	0.462	0.644	-0.048	0.078	
Lag4:Lag2	-0.0215	0.043	-0.504	0.614	-0.105	0.062	
Lag1:Lag4:Lag2	0.0188	0.028	0.670	0.503	-0.036	0.074	
Lag3:Lag4:Lag2	-0.0404	0.029	-1.385	0.166	-0.097	0.017	

Lag1:Lag3:Lag4:Lag2	0.0091	0.018	0.518	0.604	-0.025	0.044
Lag5:Lag2	0.0029	0.038	0.078	0.938	-0.071	0.076
Lag1:Lag5:Lag2	-0.0309	0.027	-1.145	0.252	-0.084	0.022
Lag3:Lag5:Lag2	0.0365	0.025	1.483	0.138	-0.012	0.085
Lag1:Lag3:Lag5:Lag2	0.0038	0.017	0.223	0.824	-0.029	0.037
Lag4:Lag5:Lag2	0.0350	0.027	1.304	0.192	-0.018	0.088
Lag1:Lag4:Lag5:Lag2	-0.0052	0.017	-0.304	0.761	-0.039	0.028
Lag3:Lag4:Lag5:Lag2	-0.0057	0.016	-0.370	0.712	-0.036	0.025
Lag1:Lag3:Lag4:Lag5:Lag2	-0.0079	0.010	-0.757	0.449	-0.028	0.013

```
In [83]: temp = results.pvalues.reset_index().sort_values(0)
current_predictors = [x for x in temp.loc[temp[0] < .05]['index'].tolist() if x]
current_predictors

results = get_results_logit(current_predictors)
results.summary()
```

Out[83]: Logit Regression Results

Dep. Variable:	Direction_Up	No. Observations:	998			
Model:	Logit	Df Residuals:	995			
Method:	MLE	Df Model:	2			
Date:	Wed, 02 Oct 2024	Pseudo R-squ.:	0.009653			
Time:	13:55:12	Log-Likelihood:	-684.96			
converged:	True	LL-Null:	-691.63			
Covariance Type:	nonrobust	LLR p-value:	0.001261			
	coef	std err	z	P> z	[0.025	0.975]
Intercept	0.0388	0.064	0.609	0.542	-0.086	0.164
Lag1:Lag5	-0.0741	0.037	-1.984	0.047	-0.147	-0.001
Lag1:Lag3:Lag4:Lag5	0.0303	0.014	2.203	0.028	0.003	0.057

```
In [89]: expanded_current_predictors = []
for x in current_predictors:
    temp = x.split(":")
    expanded_current_predictors.append(" * ".join(temp))

expanded_current_predictors
results = get_results_logit(expanded_current_predictors)
results.summary()
```

Out[89]:

Logit Regression Results

Dep. Variable:	Direction_Up	No. Observations:	998			
Model:	Logit	Df Residuals:	982			
Method:	MLE	Df Model:	15			
Date:	Wed, 02 Oct 2024	Pseudo R-squ.:	0.01509			
Time:	13:59:18	Log-Likelihood:	-681.19			
converged:	True	LL-Null:	-691.63			
Covariance Type:	nonrobust	LLR p-value:	0.1408			
	coef	std err	z	P> z	[0.025	0.975]
Intercept	0.0390	0.064	0.608	0.543	-0.087	0.165
Lag1	-0.0503	0.057	-0.886	0.376	-0.162	0.061
Lag5	-0.0096	0.054	-0.178	0.859	-0.115	0.096
Lag1:Lag5	-0.0701	0.039	-1.791	0.073	-0.147	0.007
Lag3	-0.0024	0.055	-0.044	0.965	-0.111	0.106
Lag1:Lag3	0.0549	0.042	1.321	0.187	-0.027	0.136
Lag5:Lag3	0.0354	0.037	0.969	0.332	-0.036	0.107
Lag1:Lag5:Lag3	0.0308	0.026	1.170	0.242	-0.021	0.082
Lag4	0.0062	0.055	0.112	0.911	-0.101	0.114
Lag1:Lag4	-0.0360	0.037	-0.977	0.329	-0.108	0.036
Lag5:Lag4	0.0211	0.039	0.540	0.589	-0.055	0.098
Lag1:Lag5:Lag4	-0.0081	0.022	-0.371	0.711	-0.051	0.035
Lag3:Lag4	-0.0115	0.040	-0.289	0.772	-0.090	0.067
Lag1:Lag3:Lag4	-0.0089	0.026	-0.341	0.733	-0.060	0.042
Lag5:Lag3:Lag4	0.0043	0.025	0.174	0.862	-0.044	0.052
Lag1:Lag5:Lag3:Lag4	0.0349	0.017	2.109	0.035	0.002	0.067