Supervised Machine Learning HW 2

Group 3: Maja Nordfeldt, Jakob Rauch, Timo Schenk, Konstantin Sommer

16-1-2020

Introduction

Can demographic and socioeconomic characteristics of a neighbour hood predict turnover of grocery stores in that neighborhood? A high turnover is an important factor when supermarkets choose the location of a new store Especially in urban areas, data on a large number of socioeconomic and demographic factors is available. From an intuitive perspective, most of the variables might play a role in determining sales. However, when predicting the turnover from data, it is often better to rely only on a small subset of the variables. Using data from supermarkets in the Chicago area, we estimate an elastic net and find 8 relevant predictors out of 44 observed demographic factors. We conclude that a prediction of turnovers in a certain location should rely on the share of small households, average age, share of high income households and unemployment measures.

Data

The data is store-level scanner data collected at 77 supermarkets in the Chicago area during 1996, in a collaboration between Chicago Booth School of Business and Dominick's Finer Foods[1]. Store data is matched with demographic census data of the metropolitan Chicago area, originally obtained from the US Government in 1990.

The dependent variable *grocery_sum* is the total turnover in one year of groceries, measured in $. The predictor variables are demographics and socioeconomic characteristics of a given stores' neighborhood - *age9* and *age60*, for instance represent the percent of population under age 9 and 60 respectively. *ethic* presents percent of blacks and hispanics, *nwhite* indicates the non-white share of the population, *educ* the share of college graduates, *nocar* the share of residents without automobiles, *income* the log of the median income, *poverty* indicates the share of population with income below $15,000, *incsigma* the approximated standard deviation of the income distribution, *hsizeavg* the average household size, *hsizex* and *hhxplus* variables denote percent of households containing x persons, *hhsingle* measures percentage of detached houses, *workwom* the percentage of working women with full time jobs, *sinhouse* the percentage of one-person households, *density* the trading area in square miles per capita, *hvalx* the percent of households with value over x thousands of dollars, *hvalmean* the approximated mean household value, *single* the percentage of singles, *retired* the percentage of retired and *unemp* the percentage of unemployed. *wrk* variables denotes percent of women working, with a *ch* ending denoting having children, *nch* no children and *nwrk* not working, and the number indicating age of children (5 means children younger than 5 years old and 17 means children are between 6 - 17 years old). *telephn* and *mortgage* indicates percent of households having a telephone and mortgage respectively. Further, *shopx* denotes percent of type x of shopper - including constrained, hurried, avid, unfetters, birds and stranges. Finally, *shopindx* present ability to shop, meaning having a car and being a single family house.

Note that from the original dataset we transform all variables to z-scores, removing their unit of measure. After this transformation, each variables value can be interpreted as standard deviations from its mean. The importance of this will be explained in the next section.

[1] The dataset was retrieved from https://www.chicagobooth.edu/research/kilts/datasets/dominicks

Methods

Since we are dealing with a dataset with about 44 predictors and only 77 observations we use a methodology that induces both shrinkage of parameter estimates and variable selection. This is because a situation in which the ratio of observations to predictor variables is low, there is a danger of what is called “overfitting”. In such a situation we might be able to find unbiased estimates of a coefficient for each of the variables using the standard multiple regression framework, but since there are only few observations available per parameter, the model will do very poorly in terms of out of sample prediction. We therefore use a combination of Ridge and Lasso regressions called elastic net. Both of these methods induce shrinkage of the parameter estimates towards zero, while Lasso also also induces variable selection. For both methods, the loss function - which in OLS is only the residual sum of squares - is extended by a penalty term, that increases with the size of the coefficients. So when minimizing the loss function, one can also think about the penalty term as a constraint on the size of the coefficients. Since both of these methods have advantages and disadvantages, which we will describe in the following, we use a combination of the two, the elastic net method.

The loss function for ridge regression is $(y - X\beta)^T(y - X\beta) + \lambda\beta^T\beta$. The first term is the sum of squared residuals, so the loss function of the multiple regression setting. As one can see we are here not only interested in minimizing this but in addition we also add the penalty term $\lambda\beta^T\beta$, where λ is the penalty strength. One can see that the larger the chosen value of λ the lower will be the optimal coefficients. The penalty term increases here quadratically in β, which is the key difference to the LASSO method in which the penalty term is $\lambda||\beta||_1$, where $||\beta||_1$ refers to the L1 norm so the sum of the absolute values of the vector. So the penalty term in LASSO increases with the absolute value of the coefficients and one can again state that the larger the chosen penalty term the smaller will be the resulting coefficient values.

The main difference between the two penalty terms is that while Ridge drives the coefficient values close to zero but mostly not exactly to zero, LASSO will assign an exact zero to many of the coefficients. Why this is the case becomes a bit clearer when looking at how the penalty term translate into a constraint on minimizing the RSS. While for Ridge this is $\beta^T\beta < \gamma$, for LASSO this is $||\beta||_1 < \gamma$. Both of these constraints are equivalent to minimizing the respective loss function and both span a space around the origin in which the the optimal coefficient values must be. While in Ridge this is a hyperbole, in LASSO this is a space with straight edges. One can see that when analysing the two dimensional case that it is mostly the case that for LASSO one of the coefficients will be zero and the other one will be relatively large, for Ridge, both coefficients are driven towards zero but not to exactly zero.

Driving some of the coefficients exactly to zero has the added advantage of ending up with a simpler model, since these coefficients will then not be used in prediction anymore. On the other hand, LASSO might also drive coefficients towards zero that are actually not zero, and that are then excluded eventhough they might actually explain some of the variation of the endogenous variable. So to use the advantages of both methods we will combine the two in an elastic net approach in which the penalty term is set up as a combination of the two previous ones: $\lambda(\alpha||\beta||_1 + (1 - \alpha\beta^T\beta)$, where λ still determines the penalty strength and α determines the importance of the LASSO penalty relative to the Ridge penalty.

Any deviation from the OLS loss function will result in biased estimates for the coefficients, however, a penalty term leads to the resulting coefficients exhibiting smaller variances. Additionally, OLS does not work in the presence of exact or strong multicollinearity since the X matrix of predictors will become singular or close to singular, making the inversion of the $X'X$ matrix impossible or very inexact (due to many rounding errors). Adding a penalty term, however, means that the matrix that needs to be inverted always has full rank. Moreover, a sufficiently high penalty prevents overfitting and, in the presence of many predictors and few observations, will outperform a multiple regression estimation in predicting out of sample values. One therfore faces what is known as a bias variance trade of, determined by the penalty strength λ.

Using the elastic net method, we need to decide for values of the two hyperparameters: the penalty λ and α, the weight of LASSO relative to Ridge regression. There is an optimal combination of values for these hyperparameters, in the sense of delivering the best out-of-sample performance. To find this optimal combination, we use a method called k-fold cross-validation. In this procedure, we first arrange our dataset into random order (as the order of observations might be correlated with some features of the data, e.g. when neighborhoods are sampled in a spatial order). We then split the data into k bins of equal size. We use the last (k-1) bins as a training sample (i.e. we fit our model to this data and end up with parameter estimates) and use the first bin as test sample, where we calculate the out-of-sample mean squared error, a measure of the distance between predicted values given our parameter estimates and the observed values. We repeat this k times, using in the second/third/... iteration the second/third/... bin for testing and the remaining bins for training. We finally take the square root of the average of these k mean squared errors, which will be our measure of model performance given the hyperparameters. To find the optimal values, we do a grid search - calculating the model performance for each combination of α and λ in the grid. The best values will lead to the lowest average mean squared error. These values lead to a model with good predictive performance.

As stated earlier, we standardize all our variables by substracting the mean and dividing through their variance. This turns all variables into mean zero, variance one variables that are now de facto unitless. In a classical OLS setting this only implies that the size of the coefficients can be compared in the sense that larger coefficients would imply larger effects in terms of standard deviations. But in a Ridge or Lasso setting it actually becomes necessary to do, for a very intuitive reason. In both cases if one would not standardize the scale of the variables would matter for how much they would be shrinked, because the penalization terms try to minimize the size of the coefficients. If one would now measure something in meter than instead of in kilometers the coefficient would trivially be much larger and thus more penalized. Standardizing all values will therefore guarantee that all values are treated equally.

Results

Table 1: Non-zero point estimates for (lambda,alpha) = (0.04,0.6)

AGE9	HSIZE2	HSIZE34	SINHOUSE	HVAL150	HVAL200	UNEMP	SHPHURR
0.3759	0.02787	0.1159	0.1104	0.1978	0.01314	0.254	0.1544

We applied the method to the Chicago data to select the relevant predictors out of 44 observed demographic factors. Table 1 shows the estimates of the 8 coefficients that the elastic net has not shrunken to zero. All estimates are positive, meaning they contibute to an increased store turnover. The dominating predictor is the percent of kids under 9 in the population (increasing this percentage with 1 std, holding everything else constant, increases yearly turnover with approximately 0.38 std), next followed by the percent of unemployed. Least explanatory power, but yet retained in the model has the characterstic household of a value over 200 000 dollar, and household size of two has around double its value. Household size of 3-4 and single household demonstrate similar estimates, yet roughly half of the contribution of household value over 150 000 dollar.

The optimal shrinkage parameter was obtained by K-fold crossvalidation, splitting the data into $k = 7$ parts to get equally large bins for 77 observations. Our manually programmed results are close to those from the R-package `glmnet()`. Yet the estimates are not exactly the same, as both `glmnet()` and our manual function have non-deterministic results because data is randomly rearranged. Best values are $\alpha = 0.6$ and $\lambda = 0.04$ for a grid search with α ranging between 0 and 1, with stepsize 0.1, and λ ranging between 0.01 and 1 with stepsize 0.01. The optimal α translates to only a slightly larger weight on the LASSO-part. LASSO works well when only a few β_j are not around 0, why the moderate weight on the ridge-part may indicate that this is not the case here. The quite small optimal λ means low shrinkage strength, allowing more β_j to be non-zero then a higher value would. This would indicate that a relatively small portion of variance has been traded for bias.

Conclusion & Discussion

In this study we aimed to identify what demographic characteristics that are linked to higher supermarket store turnover. Using an elastic-net strategy, we identified a few key predictors.

Firstly, it appears that the household size is relevant - in particular single and three to four-person households have explanatory power. Even more important appears the share of high-income household values to be, in particular a household value over 150 000 dollar. The return on targeting a neighbourhood with a higher share of value over 200 000 dollar is however not as large. Somewhat contradictory to the income predictors is the share of unemployed, being the second highest predictor. Further, hurried shoppers appear to be a profitable group to target. The most important factor seems to be share of population under age of 9, speaking to the value of households with young children For future stores, planners might want to pay extra attention to candidate neighbourhoods employment, income, household size and kids demographics. However, recommendations should be considered together with studies based on more recent data, as consumers shopping habits may have changed with e.g. the last decades technology development. For such reasons, future studies may want to use more recent data, in addition to consider the online shopping channels to plan store location efficiently. Conclusions may also not generalize to other locations, and there could be important demographic variables not included.

Considering our chosen methodology, a risk perhaps worth considering is the risk of overfitting, having optimized two parameters based on our sample, and true values may be different in the population. Further, although our elastic net strategy provides benefits of both LASSO and ridge regressions, it is computationally heavier. Additionally, we do not know with certainty that the variables assigned 0-value estimates are indeed 0. A future study could replicate the work with a larger observation set, such that standard regression could appropriately be applied and estimates identified without the loss of precision.

Code

```r
# load necessary packages
library("pander") # for tables
library("glmnet")

# Load supermarket data from github
githubURL = "https://github.com/jakob-ra/Supervised-Machine-Learning/raw/master/HW_2/supermarket1996.RDa
load(url(githubURL))
df = subset(supermarket1996, select = -c(STORE, CITY, ZIP, GROCCOUP_sum, SHPINDX) )
attach(df)

# create summary statistics
pander(summary(supermarket1996) , caption = 'Summary statistics')

# Create vector y (turnover)
y = GROCERY_sum

# Create matrix X of predictor variables
X = subset(df, select = -GROCERY_sum)

# Standardize variables
X = scale(X)
y = scale(y)

# Loss function for MM-algorithm
loss = function(y,X,alpha,lambda,beta,epsilon){
```

```r
  # Loss function of MM-algorithm given the data, parameters beta, hyperparameters lambda and alpha,
  # and convergence threshold epsilon. Returns both the value of the loss function and the D matrix

  n = length(y) # Number of observations
  p = length(beta) # Number of parameters

  D = matrix(0,p,p) # Initialize p times p matrix of 0s
  beta_abs_list = rep(0,p) # Vector to hold absolute values of beta (will need them later)

  for (j in seq(0,p,1)){
    beta_abs_list[j] = abs(beta[j])
    D[j,j] = 1/max(c(beta_abs_list[j],epsilon))
  }

  c = 1/(2*n)*t(y)%*%y + 1/2*lambda*alpha*sum(beta_abs_list)
  l = 1/2*t(beta)%*%(1/n*t(X)%*%X + lambda*(1-alpha)*diag(p) +
                     lambda*alpha*D)%*%beta - 1/n*t(beta)%*%t(X)%*%y + c

  return(list(l,D))
}

# MM-algorithm for minimizing the elastic net loss function
elastic_net_MM = function(y,X,alpha,lambda,beta_0=rep(0,(dim(X)[2])),epsilon=10^(-8)){
  # Fits an elastic net model via MM-algorithm. Returns vector beta_hat for given data,
  # hyperparameters lambda and alpha, intitial parameter guess beta_0 (default is a vector of 0s)
  # and convergence threshold epsilon (default is 10^(-8)).

  n = length(y) # Number of observations
  p = (dim(X)[2]) # Number of parameters

  # Set values for the first iteration
  beta = beta_0
  k = 1
  # Set l_new and l_old so that (l_old-l_new)/l_old > epsilon is true in the first iteration
  l_new = 0
  l_old = 1

  # Iterate the approximation until convergence
  while ((l_old-l_new)/l_old > epsilon){
    l_and_D = loss(y,X,alpha,lambda,beta,epsilon) # Loss returns both the value of the loss function an
    l_old = l_and_D[[1]]  # Gets value l of loss function
    D = l_and_D[[2]] # Gets matrix D, which we already computed in the loss function
    A = 1/n*t(X)%*%X + lambda*(1-alpha)*diag(p) + lambda*alpha*D
    beta = solve(A, 1/n*t(X)%*%y)
    l_new = loss(y,X,alpha,lambda,beta,epsilon)[[1]]

   # print(k)
    k = k + 1
    #print(l_old-l_new)
  }

  return(beta)
}
```

```
k_fold_crossval = function(y,X,k,alpha,lambda){
  # Function for k-fold crossvalidation, returns RMSE for given alpha and lambda
  n = length(y) # Number of observations
  p = (dim(X)[2]) # Number of parameters

  # Reshuffle data and split into k groups of equal size
  reshuffled_indices = sample(seq(1,n,1), n, replace=FALSE) # Shuffle indices of our n observations
  n_test = floor(n/k) # n=77 and k=10 would give 7 obs. in the test set the rest of the observations
  # in the training set. For n=77, using k=11 or k=7 is advisable to get evenly sized groups.

  MSE=array(0,k) # Vector to hold the MSEs for each fold

  # Loop over the k folds and save MSEs
  for (i in seq(1,k,1)){
    # Divide data into test and training
    y_test = y[c(reshuffled_indices[(1+(i-1)*n_test):(i*n_test)])]
    y_train = y[-c(reshuffled_indices[(1+(i-1)*n_test):(i*n_test)])]
    X_test = X[c(reshuffled_indices[(1+(i-1)*n_test):(i*n_test)]),]
    X_train = X[-c(reshuffled_indices[(1+(i-1)*n_test):(i*n_test)]),]

    beta = elastic_net_MM(y_train,X_train,alpha,lambda) # get beta estimates for training data from MM
    fitted_val = (X_test)%*%beta # get y_hat for test data
    MSE[i]=1/n_test*t(y_test-fitted_val)%*%(y_test-fitted_val) # save MSE values for test data
  }

  RMSE = sqrt(sum(MSE)/k)

  return(RMSE)
}


min_RMSE = function(y,X,k=7,alpha_values=seq(0,1,length.out=3),lambda_values=10^seq(-3, 4, length.out =
  # Tunes hyperparameters alpha and lambda via k-fold crossvalidation.
  # Returns optimal combination of alpha and lambda
  # as well as the resulting RMSE.

  RMSE = matrix(0,length(alpha_values),length(lambda_values)) # create matrix to hold RMSE for each
                                                               # hyperparamter combination

  # Loop over hyperparameter combinations
  for (i in 1:length(alpha_values)){
    for (j in 1:length(lambda_values)){
      RMSE[i,j] = k_fold_crossval(y,X,k,alpha_values[i],lambda_values[j]) # Fill the matrix with RMSE
    }
  }

  min_index = arrayInd(which.min(RMSE), dim(RMSE)) # Find index of lowest RMSE
  alpha_min = alpha_values[min_index[1]] # alpha value at lowest RMSE
  lambda_min = lambda_values[min_index[2]] # lambda value at lowest RMSE
  RMSE_min = RMSE[min_index] # lowest RMSE

  return(list(alpha_min,lambda_min,RMSE_min))
```

```r
}

## We want to compare our result with glmnet function, which cannot cross-validate alpha
## -> compare for fixed alpha
alpha = 0.5

# Find best cross validated lambda from glmnet, letting the function choose
# the sequence of lambdas to try
result.cv <- cv.glmnet(X, y, alpha=alpha, folds=7)
print(result.cv$lambda.min)
# Note that the result here varies quite a bit but is frequently equal or close to 0.05

# Find best cross validated lambda from our own function,
# with a fine sequence of lambdas between 0 and 1
print(min_RMSE(y,X,alpha_values=alpha,lambda_values=seq(0.01,1,0.01))[[2]])
# --> Results are equal or very close


## Knowing that everything works as intended, find the best combination of alpha and lambda
result = min_RMSE(y,X,alpha_values=seq(0,1,0.1),lambda_values=seq(0.01,1,0.01)) # This takes a while
print(result)
# Get point estimates on full sample for the optimal values of alpha and lambda
beta_hat = elastic_net_MM(y,X,0.6,0.04)

# set very low point estimates (rounding errors) to 0
zero_ind = arrayInd(which(beta_hat < 10^(-5)), dim(beta_hat))
beta_hat[zero_ind] = 0
pander(t(beta_hat[beta_hat[,1]!=0,]) ,
       caption = "Non-zero point estimates for (lambda,alpha) = (0.04,0.6)")
```