Estimating distribution mean and variance based on underlying parameters

Macmillan Jacobson

April 2023

1 Abstract

In the analysis described, below, we will train two kinds of models to predict the means and variances of unknown distributions based on their two underlying parameters. We define our target MSEs as ≤ 120 for our mean predictions and ≤ 550 for our variance predictions. By using a training and testing split and a 5-fold cross validation scheme, we find that the best predictor of each distributions mean is a LOESS model with tuned hyperparameters. This results in a training MSE of $MSE = 1.20287$. By using the same analysis scheme as our mean predictions, we find that the best predictor of variance of each distribution is an SVR model with tuned hyperparameters. The resulting MSE is $MSE_{\mu} = 531.6501$. Our models achieve our targets and we will use them to predict mean and variance, respectively, on our testing set.

2 Introduction

Suppose that we have a distribution based on two underlying independent parameters. We have a distribution $Y = Y(X_1, X_2)$ where Y is assumed to be a random variable whose distribution depends on X_1 and X_2, which are independent. Our task is, given 100 X_1s, 100 X_2s and 200 samples from Y for each (X_1, X_2) pair, to create a model that can estimate a distribution Y's mean and variance given its underlying parameters, X_1 and X_2. We will do this by employing several learning algorithms and measuring their efficacy in prediction. Once we have developed a sufficiently accurate model, we will use it to predict the mean and variance of a Ys distribution based on given X_1, X_2 values. Our results are as follows.

3 Problem Statement

It is often difficult to find an exact distribution to match a given set of data. However, given some underlying parameters of the distribution of a data set and realization of values from that data set, we can build a model that should be able to estimate the distribution's mean and variance, among other statistical measures. In this paper, we will focus on estimating just mean and variance with target MSEs on our testing data set of ≤ 120 for our mean predictions and ≤ 550 for our variance predictions.

4 Description of Data

The training set that we are given contains 100 X_1 values, 100 X_2 values and 200 samples from each distribution defined by that distributions parameters, X_1 and X_2. Specifically, since we have 100 X_1s and X_2s, we have $100 * 100 = 10^4$ rows in our data set, each corresponding to a different Y distribution. In addition to our X_1 and X_2s, we also have 200 samples from each distribution. Thus, our training data set is a 10,000 X 202 table, each row corresponding to a distribution, each rows first two columns corresponding to X_1 and X_2 values and the remaining 200 columns contain realization from each distribution.

5 Proposed Methodology

Our first task is to develop dependent variables for our model. This is easily accomplished by simply taking the mean and variance of each tuple's 200 samples. This results in two separate data sets to train on. The first consists of 10,000 rows, each corresponding to a different distribution and 3 columns, the first two being X_1 and X_2 respectively and the final column being the mean of the 200 samples from our original data set. The second data set we will train on is exactly the same, except that the third column is the variance of the 200 samples from our original data set.

Since we do not know the distribution that Y is based on, and only its parameters, it makes sense to take a non-parametric approach to solving this problem. Non-parametric methods are using to analyze data that does not come from an assumed distribution. Though we do know that the data does come from some distribution, we have no way of determining which distribution it is without an exhaustive search and since there are thousands of potential distributions, it makes sense to relax that assumption and attempt to build a model that does not make any assumptions.

Upon deciding to use a non-parametric approach, several options are available. We will start with 2 potential models and expand the search if the results from these two models are deemed to be insufficient in accuracy. The chosen models are support vector regression (SVR) and locally estimated scatterplot smoothing (LOESS). Each of these models will be trained on a training set and measured against a testing set. The metric we will use to measure a model's accuracy is mean squared error, computed as follows:

$$MSE_\mu = \frac{1}{J}\sum_{i=1}^{I}\sum_{j=1}^{J}(\hat{\mu}(x_{1i}, x_{2j}) - \mu^*_{true}(x_{1i}, x_{2j}))^2$$

$$MSE_V = \frac{1}{IJ}\sum_{i=1}^{I}\sum_{j=1}^{J}(\hat{V}(x_{1i}, x_{2j}) - V^*_{true}(x_{1i}, x_{2j}))^2$$

Where I is the number of X_1s, J is the number of X_2s, $\hat{\mu}(x_{1i}, x_{2j})$ and $\hat{V}(x_{1i}, x_{2j})$ are our respective mean and variance estimates, and $\mu^*_{true}(x_{1i}, x_{2j})$ and $V^*_{true}(x_{1i}, x_{2j})$ are the true mean and variance values.

An 80/20 train/test split will be used in the analysis top measure each model's efficacy. Additionally, each of these models' hyperparameters will be tuned using 5-fold cross validation and an exhaustive search of parameters. In the case of SVR, we will also scale our data in order to use SVR properly, then before calculating the MSE, we will invert that scaling for our predictions to get an accurate measurement of MSE.

Once we have decided which tuned models to use, we will retrain the model using our entire training data set. Then we will use these new models to predict on the testing set provided to us.

6 Analysis and Results

In our analysis of the SVR model, we will use Python's SKLearn package for scaling, train/test split and model training. We first train our SVR using the cross validation scheme defined above. We first train on our data set containing the mean measurement. Our cross validation scheme resulted in using an RBF kernel with a kernel coefficient of $\frac{1}{nfeatures} = \frac{1}{2}$ and a regularization parameter of $C = 1000$. Our resulting MSE from the above SVR on our testing data is: $MSE_\mu = 1.2654$. This is quite good, but we will continue with the LOESS model to compare.

Next, we use the SVR model to predict variance. Again, using the cross validation scheme defined above, we find that the best hyperparameters are again and RBF kernel with a kernel coefficient of $\frac{1}{nfeatures} = \frac{1}{2}$ and a regularization parameter of $C = 110$. The resulting MSE in the case of variance on our testing set is: $MSE_\mu = 531.6501$.

We will continue with our analysis by training a LOESS model to do the same predictions as above. For our LOESS model we will use R's e1071 package for our training model and the caret package for our fold creation for cross validation. Our hyperparameter tuning resulting from our cross validation scheme in the case of predicting the mean are a degree 2 polynomial with a span of 0.06414. This model resulted in a testing MSE of $MSE = 1.20287$. This is quite good and likely at the limit of the best results we can achieve.

In the case of predicting variance, our hyperparameter tuning results in, again a degree 2 polynomial, but with a span of 0.1035. The resulting MSE is $MSE = 542.333$. A bit worse than our SVR model.

The results above are extremely good. We will not continue to explore better models as any improvements in our MSEs would likely be marginal, is any. We will use our LOESS model to predict both our means and an SVR model to predict our variances.

7 Conclusion

In the conclusion of our analysis, we found that using a combination of a LOESS model and an SVR model to predict our mean and variance, respectively, results in the lowest MSE. We will use the models defined above, trained on our entire complete data sets, to predict the means and variances of distributions defined by the parameters given in the training set. If our models result in MSEs similar to those found in the training data, we should achieve $MSE_\mu \leq 120$ and $MSE \leq 550$, which are our targets.

8 Code For This Report

8.1 SVR

```
import pandas as pd
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.metrics import mean_squared_error
from scipy.stats import randint
from sklearn.preprocessing import StandardScaler
from sklearn.svm import SVR

df = pd.read_csv('7406train.csv', header=None)

df_t = df.set_index([0,1])
mean = df_t.mean(axis=1)
var = df_t.var(axis=1)
df_t['mean'] = mean
df_t['var'] = var
df_t = df_t.loc[:, 'mean':].reset_index().rename(columns={0:'x1', 1:'x2', 'mean'
df_t

X = df_t.drop(['Ym', 'Yv'], axis=1)
Ym = df_t['Ym']
Yv = df_t['Yv']

sc_X = StandardScaler()
sc_ym = StandardScaler()
sc_yv = StandardScaler()
Xs = sc_X.fit_transform(X)
ysm = sc_ym.fit_transform(Ym.values.reshape(-1, 1))
ysv = sc_yv.fit_transform(Yv.values.reshape(-1, 1))

Xsm_train, Xsm_test, ysm_train, ysm_test = train_test_split(Xs, ysm, test_size=0
Xsv_train, Xsv_test, ysv_train, ysv_test = train_test_split(Xs, ysv, test_size=0

rand_search = GridSearchCV(
    SVR(),
    param_grid = {
        'kernel': ['rbf'],
        'gamma': ['auto'],
        'C': [1000,2000,500]
    },
    cv=5
)

rand_search.fit(Xsm_train, ysm_train.ravel())
```

```
print('Best hyperparameters:', rand_search.best_params_)

ysm_pred = rand_search.best_estimator_.predict(Xsm_test)

print(
    'MSE-Mean: ' +
    str(mean_squared_error(
        sc_ym.inverse_transform(ysm_pred.reshape(-1,1)),
        sc_ym.inverse_transform(ysm_test))
       )
)

rand_search = GridSearchCV(
    SVR(),
    param_grid = {
        'kernel': ['rbf'],
        'gamma': ['auto'],
        'C': [90,100,110]
    },
    cv=5
)

rand_search.fit(Xsv_train, ysv_train.ravel())

print('Best hyperparameters:', rand_search.best_params_)

ysv_pred = rand_search.best_estimator_.predict(Xsv_test)

print(
    'MSE-Mean: ' +
    str(mean_squared_error(
        sc_yv.inverse_transform(ysv_pred.reshape(-1,1)),
        sc_yv.inverse_transform(ysv_test))
       )
)

reg = SVR(kernel='rbf', gamma='auto', C=110)
reg.fit(Xs, ysv.ravel())
Xtest = pd.read_csv('7406test.csv', header=None)
Xtests = sc_X.transform(Xtest)

ypredVar = reg.predict(Xtests)
yPredVar = sc_yv.inverse_transform(ypredVar.reshape(-1,1))
```

8.2 LOESS

```
traindata <- read.table(file = "./7406train.csv", sep=",");
df = cbind(traindata[c(1:2)], rowMeans(traindata[-c(1:2)]))
colnames(df) = c('x1', 'x2', 'Y')
set.seed(1)
sample <- sample(c(TRUE, FALSE), nrow(df), replace=TRUE, prob=c(0.8,0.2))
train  <- df[sample, ]
test   <- df[!sample, ]

library(e1071)
library(caret)
folds <- createFolds(train$Y, k = 5)
mses = c()
ss = c()
for (s in seq(0.05, 0.75, len=100)){
  results = c()
  for (i in 1:5){
    tr <- train[-folds[[i]],]
    te <- train[folds[[i]],]
    loess <- loess(Y ~ ., data=tr, span=s, degree=2, surface='direct')
    yPred = predict(loess, newdata = te[c(1:2)])
    k = (te[c(3)] - yPred)^2
    result = sum(k)/dim(k)[1]
    results <- c(results, result)
  }
  mses = c(mses, mean(results))
  ss = c(ss, s)
}
mses[which.min(mses)]
ss[which.min(mses)]

loess <- loess(Y ~ ., data=train, span=ss[which.min(mses)], degree=2, surface="d
yPred = predict(loess, newdata = test[c(1:2)])
k = (test[c(3)] - yPred)^2
sum(k)/dim(k)[1]

library(matrixStats)
dfv = cbind(traindata[c(1:2)], rowVars(as.matrix(traindata[-c(1:2)])))
colnames(dfv) = c('x1', 'x2', 'Y')
sample <- sample(c(TRUE, FALSE), nrow(dfv), replace=TRUE, prob=c(0.8,0.2))
trainv <- dfv[sample, ]
testv <- dfv[!sample, ]

folds <- createFolds(trainv$Y, k = 5)
msesv = c()
```

```
ssv = c()
for (s in seq(0.05, 0.15, len=100)){
  results = c()
  for (i in 1:5){
    tr <- trainv[-folds[[i]],]
    te <- trainv[folds[[i]],]
    loess <- loess(Y ~ ., data=tr, span=s, degree=2, surface='direct')
    yPred = predict(loess, newdata = te[c(1:2)])
    k = (te[c(3)] - yPred)^2
    result = sum(k)/dim(k)[1]
    results <- c(results, result)
  }
  msesv = c(msesv, mean(results))
  ssv = c(ssv, s)
}
msesv[which.min(msesv)]
ssv[which.min(msesv)]

loess <- loess(Y ~ ., data=trainv, span=ssv[which.min(msesv)], degree=2, surface
yPred = predict(loess, newdata = testv[c(1:2)])
k = (testv[c(3)] - yPred)^2
sum(k)/dim(k)[1]
```