University of Applied Sciences and Arts Northwestern Switzerland
School of Business

DS-05 Resampling Lab: Applying VSA and CV

Dr. Gwendolin Wilke

Task 1: Apply VSA to the linear models you trained in Lab 1

- In Lab 1, you trained linear regression models for the `Auto` data set (which is contained in the `ISLR` library).
- You evaluated and compared your models based the *training* MSE.
- By now we now that the training error is not a good metric to asses the generalization capabilities of a model.
- To avoid overfitting, it is necessary to look at the *test* error instead (VSA).

1. **Load the ISLR library.**
2. **Do the necessary preprocessing of the Auto dataset (cf. Lab 1), namely:**

```
Auto1 <- Auto
Auto1$origin <- as.factor(Auto1$origin)
```

3. **Split the data in training and test set using a 30% - 70% split**

   Note: Do not set a seed here, so that you get different results in every run. At the end of this lab we will compare the different runs to see how stable / unstable the results are.

```
train = sample(nrow(Auto1),floor(nrow(Auto1)*0.7))
```

   Here, we generate indices of a training data set by random sampling 70% of the number of rows of our data set.
   The result is a 70% / 30% split in training / test data.
   Notice that the function floor() rounds down the result of the division.
   If you need help, look at help(sample).

```
Auto1Train <- Auto1[train,] # training data
Auto1Test <- Auto1[-train,] # test data
```

4. **Fit the different regression models you explored in Lab 1, but this time fit it only to the training data.**
5. **Calculate the *training* MSEs of your models (cf. Lab 1).**
6. **Calculate the *test* MSEs of your models.**

   To do that, predict the output variable on the test set using your model, and compare it with the true output values on the test set.
   E.g., for a model called lm.fit3, you get the predictions on the test set with `predict(lm.fit3, Auto1Test)`, and you get the true output values with `Auto1Test$mpg`.
   Then you can calculate the test MSE with `mean((Auto1Test$mpg - predict(lm.fit3, Auto1Test))^2)`.

4. **Compare the training MSEs and the test MSEs of your models.**

   The test MSE should be bigger than the training MSE (in the majority of cases).

Task 2: Apply CV to the linear models you trained in Lab 1

- Even better than applying VSA is to compare the cross-validation errors (CV) of your models.
- The CV error does not depend on the random choice of training/test split, and is therefore more reliable.

Calculate the k-fold cross validation error for your regression models. To do that, do the following:

1. Install and load the library `boot`.

2. Fit your linear models again, but this time…
   - … instead of fitting them only to the training data set, **fit them to the whole data set**.
     The reason is that cross-validation splits the data set into training and test set in each of it's folds anyways – we don't need to do it.
   - … instead of using the `lm()` function, **use the** `glm()` **function**[*].
     `glm()` allows us to use the function `cv.glm()` of the library `boot`, which conveniently provides us with the cross-validation error of our model.

3. Calculate the **5-fold cross-validation errors** of your models using the function `cv.glm()` from the library `boot`.
   E.g., for a model called `glm.fit3`, you would do:

   ```
   cv.error.fit3 = cv.glm(<Subset of Auto1>, glm.fit3, K=5)$delta[2]
   ```

   - Here, `<Subset of Auto1>` contains only the features of `Auto1` that you used to train your model. E.g., assume you used only the attributes `weight, year` and `origin` to train your model, you get `(<Subset of Auto1>` with `select(Auto1,-c(cylinders, horsepower, displacement, name))`
   - `K=5` applies 5-fold cross validation.
   - The output of `cv.glm` is a data frame that contains a feature `delta. delta` again holds two values, the second of which is the cross validation error.
   - If you want to know more about the function, consult `help(cv.glm)`.

[*] *Remark:* It that it makes no difference whether we use `lm()` or `glm()` for fitting the models, the results should be exactly the same.
- `glm()` stands for *generalized linear models*. It allows to fit, e.g., a logistic regression model or a polynomial regression model to your data.
- `glm()` has a parameter `family`. If `family` is not specified, `glm()` defaults to `lm()`.
- You can try it out yourself with one of your models. E.g., for a linear model named `glm.fit3`, you could do:

```
lm.fit3  <- lm(mpg ~ weight+year+origin, data=Auto1)
glm.fit3 <- glm(mpg ~ weight+year+origin, data=Auto1)
```

4. Now compare the training error, the test errors and the cv errors of all your models.
5. Rerun your code several times from beginning to end. Since you did not set a seed, …
   - the randomizer for sampling the training and test data in VSA chooses a different training set every time. The training and test errors will thus be different in every run.
   - The randomizer used for calculating the cv folds will choose different folds ever time. The cv-error will thus be different every time.

   What you should see is that the cv error is much more stable than the test error: The test error “jumps around” much more than the cv error.