Machine Learning B
Home Assignment 5

Jakob Schauser, pwn274

January 2021

1 PAC-Bayesian Aggregation

As the choice of programming language was free, I of course went with python. As it took quite a lot of code to solve this part of the assignment the main result is just shown here. I have used the bounds from the paper:

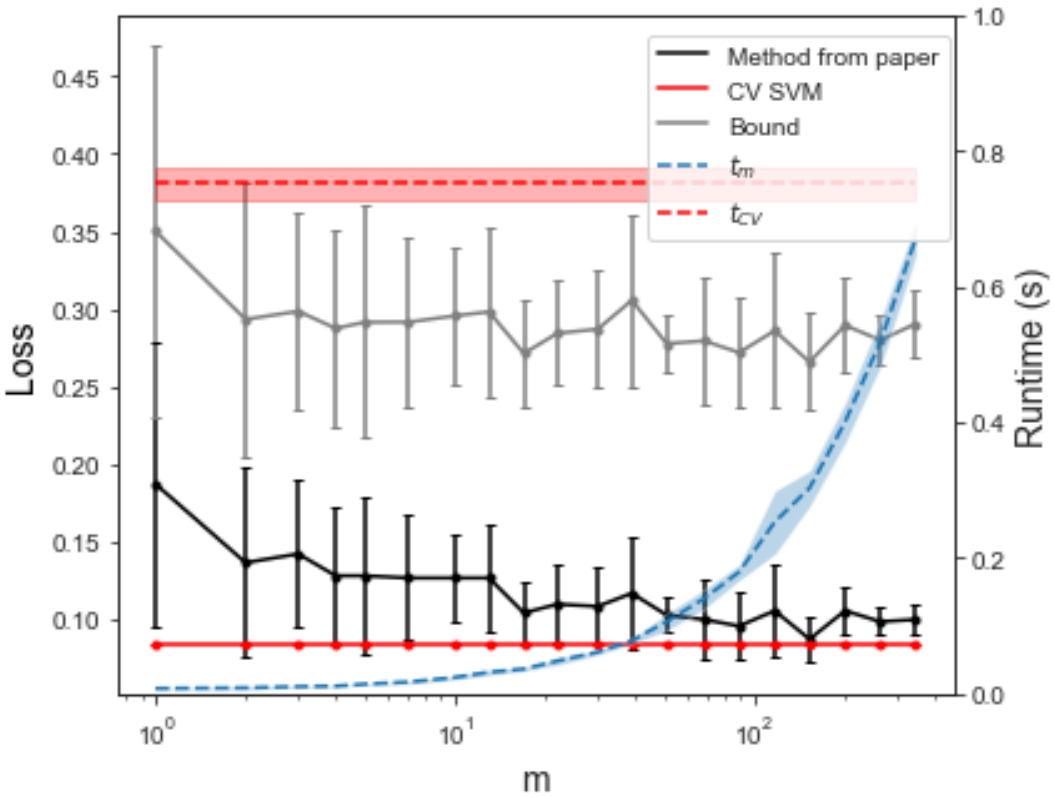

Figure 1: The main result/crown jewel of the home assignment

As I have no idea which snippets you would like to see, I have chosen the most interesting here. Feel free to skip and/or just look in the actual notebook:

Importing the data:

```
with open("ionosphere.data") as dat:
    dat = np.genfromtxt(dat,delimiter = ",",dtype=str)
    y = (dat[:,-1]=='g')*2-1
    X = dat[:,:-1].astype(float)
```

The bound:

```
def bound(lambd,loss,rho,n,r,delta):
    a1 = np.sum(rho*loss.T)/(1-lambd/2)
    a2 = (KL(rho,1/m*np.ones(len(rho))) + np.log(2*np.sqrt(n-r)/delta))
    a3 = lambd*(1-lambd/2)*(n-r)

    res = a1 + a2/a3

    return res
```

Finding the baseline (this was done multiple times and averaged):

```
params = {'gamma': np.logspace(-3,3,7), "C": np.logspace(-4,4,9)}
clf = svm.SVC()

GS = GridSearchCV(clf,params)

GS.fit(X_test, y_test)

baseline = svm.SVC(C=GS.best_params_['C'],gamma=GS.best_params_['gamma'])

baseline.fit(X,y)

base_loss = baseline.predict(X_test)
```

The majority vote:

```
scaled_guesses = np.array([model.predict(X_test) for model in models])
guess = np.sign(np.sum(rhos*scaled_guesses.T,axis = 1))
```

The

2 AdaBoost

I had no time to answer this question. But, looking at the bright side, this also means it takes less time for you to grade this part of my assignment.

This means, that you with good conscience can go watch a 4-5-minute YouTube video before continuing.

You're welcome!

3 XGBoost

3.1

Loading the data, extracting the targets and splitting into test are done by the following the lines:

```
data = np.genfromtxt("quasars.csv",dtype = float,delimiter = ",")

X, y = data[:,:-1], data[:,-1]

Xtrain, Xtest, ytrain, ytest = train_test_split(X,y, test_size = 0.2)
```

3.2

As I wanted continous evaluation on 10 percent of the training set, the training loop was done using this code snippet:

```
model.fit(X90, y90,
        eval_set=[(X10, y10)],
        eval_metric='rmse')
```

The training looked as follows:

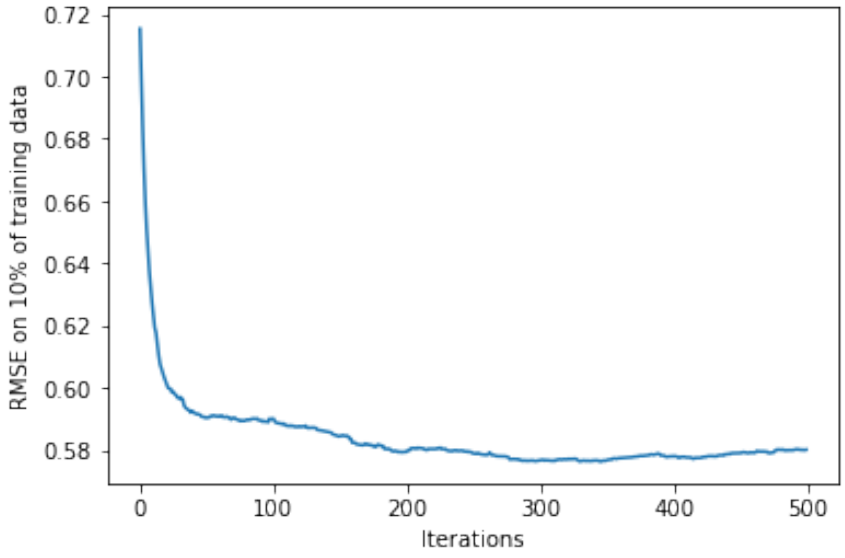

Figure 2: The training of the XGB

The final RMSE for the test set was 0.5422

3.3

The grid was defined as follows:

```
params = {
    'colsample_bytree':np.linspace(0.1,1.1,grain),
    'learning_rate':np.linspace(0.1,1.1,grain),
    'max_depth':np.linspace(1,8,grain).astype(int),
    'reg_lambda':np.linspace(0,1,grain),
    'n_estimators':np.linspace(100,500,grain).astype(int)}
```

A grid search was conducted, using these three lines:

```
search = GridSearchCV(model,params,verbose = 0)
search.fit(Xtrain,ytrain)
best = xgb.XGBRegressor(**search.best_params_)
```

This gave the following parameters:

```
search.best_params_ =
{'colsample_bytree': 0.6,
 'learning_rate': 0.1,
 'max_depth': 8,
 'n_estimators': 100,
 'reg_lambda': 1.0}
```

After training and predicting, the new models RMSE is 0.530

This is only slightly better, which is dissapointing as the grid search was quite slow.

4 Illustration of Bernstein's Inequality for Bernoulli

4.1

The Hoeffding I am using is the one that we should all know by heart:

$$e^{-2n\varepsilon^2} \tag{1}$$

The Bernstein I am using is the one that was uploaded on the weekly schedule:

$$e^{-\frac{n\varepsilon^2}{2\sigma^2+2c\varepsilon/3}} \tag{2}$$

Where c is set to 0.5 as we are looking at Bernoulli.

The plot:

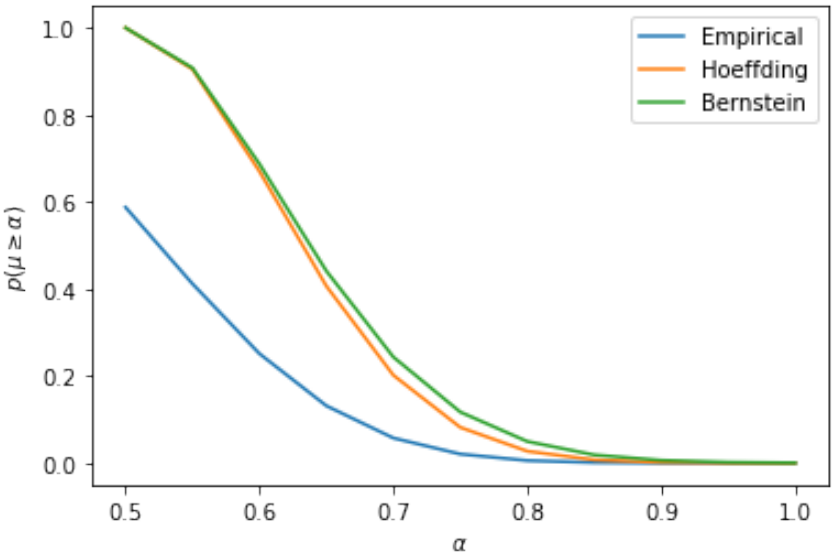

Figure 3: Plot for $\mu = 0.5$

For the more likely results, (i.e. the 50% chance of being above 0.5), both bounds are quite loose. In the more improbable end of the spectrum, the bounds are more useful, with Hoeffding actually being most tight.
When I calculate i get the exact probabilities I use the following logic:
$p = 1/20^{20}$ for $\alpha = 1$ as it is only when all "coins" are flipped to 1
$p = 21/20^{20}$ for $\alpha = 0.95$. Here we allow for 1 of the coins to be flipped to 0. This can happen in an additional 20 ways.

4.2

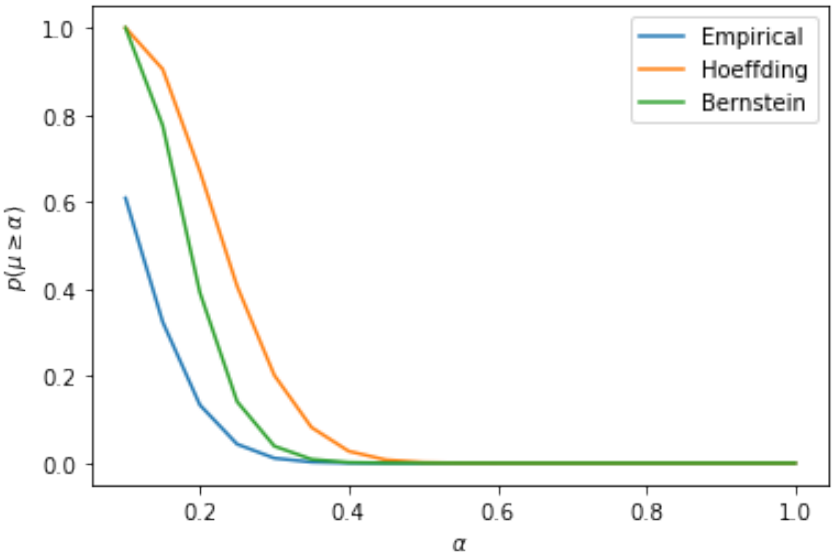

Figure 4: Plot for $\mu = 0.1$

4.3

In the second plot, at the leftmost point (where the chance should be 50% but I am not getting for some reason), the two bounds are about the same in the two cases last time. But for $\mu = 0.1$ it does not take long for both bounds to become more tight. Here we also sees Bernstein's bound shine, as it is consistently tighter than Hoeffding's.

4.4

I had no time to solve this :(