Machine Learning B
Home Assignment 2

Jakob Schauser, pwn274

November 2021

1 The airline question

1.1

The odds are binomial so this the probability can be calculated precisely:

$$f(k,n,p) = \frac{n!}{k!(n-k)!}p^k(1-p)^{n-k} \tag{1}$$

$$p = f(100, 100, 1-0.05) = (1-0.05)^{100} = 0.0059 \tag{2}$$

1.2

I use Hoeffding:

$$\mathbb{P}\left(\frac{1}{n}\sum_{i=1}^{n} X_i - \mu \geq \varepsilon\right) \leq e^{-2n\varepsilon^2} \tag{3}$$

$$\mathbb{P}\left(\frac{1}{n}\sum_{i=1}^{n} X_i \geq \varepsilon + \mu\right) \leq e^{-2n\varepsilon^2} \leftrightarrow \mathbb{P}\left(\frac{1}{n}\sum_{i=1}^{n} X_i \geq p\right) \leq e^{-2n(p-\mu)^2} \tag{4}$$

Using this for both and multiplying gives:

$$\mathbb{P}\left(\frac{1}{n}\sum_{i=1}^{n} X_i \geq p\right)\mathbb{P}\left(\frac{1}{n}\sum_{i=1}^{n} X_i \geq p\right) \leq e^{-2\cdot 10k(p-0.05)^2}e^{-2\cdot 100(p-0.00)^2} = e^{-2(10k\cdot(p-0.05)^2+100\cdot p^2)} \tag{5}$$

I then differentiate and set equal to zero for finding the maximal p:

$$0 = -2(100 + 10k2(p-0.05)) \cdot e^{-2(10k\cdot(p-0.05)^2+100\cdot p^2)} \tag{6}$$

Max is at $p = 5/101$ I get the 'bad' bound of 0.61.
This can of course also be done like the first part (purely multiplying binomials).

$$p(100, 100, 0.95)p(1000, 1000, 0.95) \tag{7}$$

This gives a (much better) bound of: 0.00011 at $p = 0.95$

The combination of a Hoeffding and binomials gives the "official" best as written in the assignment, but I have no idea why this combination of things would be preferred.

1.3

2 The growth function

2.1

The growth function is defined as the maximal number of dichotomies. On n points, this is either be seen purely by combinatorics as 2^n or (if we have fewer hypotheses than data points) the number of hypotheses M. As both of these are upper bounds, it is easy to see that

2.2

As the minimal number of ways to classify any number of data points given two hypotheses is at least 2, we have a lower bound: $m_{\mathcal{H}}(n) \geq 2$. We also know that $m_{\mathcal{H}}(n) \leq \min(M, 2^n) = \min(2, 2^n) = 2$ for all $n > 0$. So, given at least one data point, we get:

$$2 \leq m_{\mathcal{H}}(n) \leq 2 \tag{8}$$

This is of course the same as saying:

$$m_{\mathcal{H}}(n) = 2 \tag{9}$$

For two hypotheses.

2.3

We can rewrite $\min(M, 2^{2n}) = \min(M, 2^{n^2})^2$. It is trivial to see that $\min(a, b^2) \leq \min(a, b)^2$ is true, as long as a and b are larger or equal to one (which we can assume). This is clear, as the left hand side is the square of the right a is the lower, and the two sides are equal if b^2 is the larger. If a is between b and b^2 the left will evaluate to a and the right will choose b, evaluating to b^2. This is per definition less than b^2.

3 Support Vector Machines

3.1

Deliverables: number of training and test examples; code snippets for the nor- malization; mean and variance of features in the test data The split between the classes is:
Train: [45.33333333 54.66666667]
Test: [46.95121951 53.04878049]

I assume you mean the mean and variance of the transformed test data. You did not specify how you wanted the numbers, so here they are:

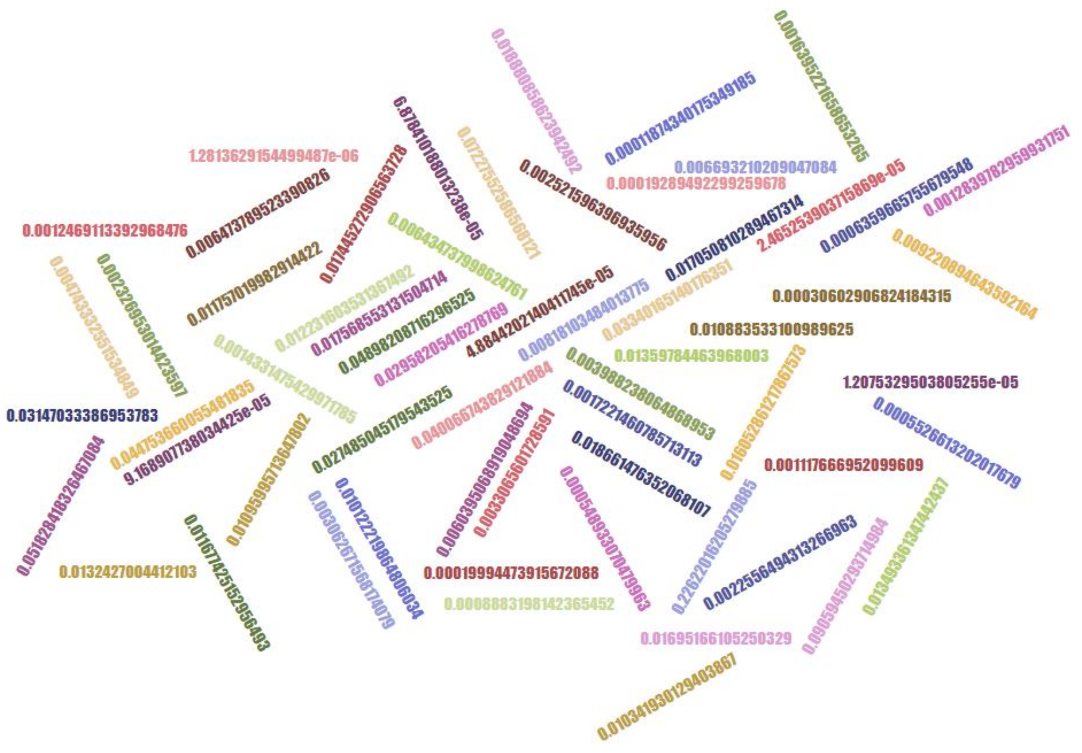

Figure 1: The means.

Most are around 0 (which is to be expected) but there are some that are off.

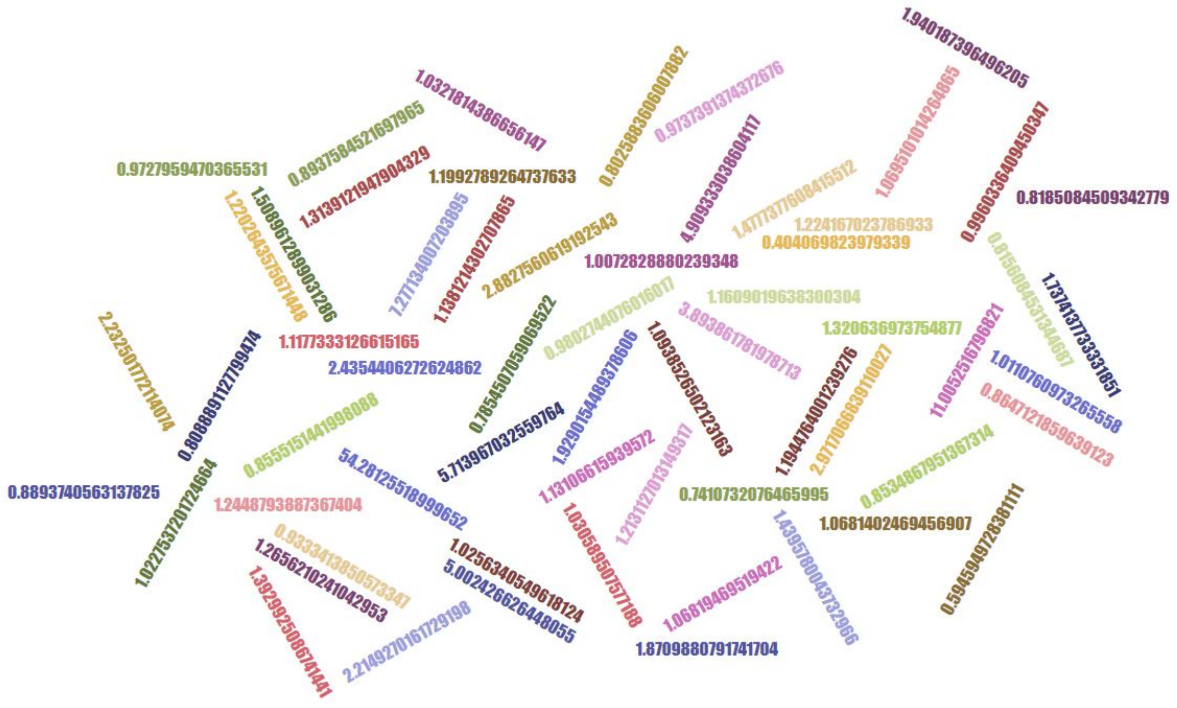

Figure 2: The variances.

Again, most are around 1 (which, again, is to be expected) but there are some clear outliers!

The scaling could have been done by hand, but I did it the Python-way and imported a function:

```
scale = StandardScaler()
X_train_scaled = scale.fit_transform(X_train)
```

3.2

The software carrying the code is sklearn as usual.

```
from sklearn.svm import SVC
from sklearn.model_selection import cross_val_score
```

The grid search was done by the following four lines:

```
Cs, Gs = np.logspace(-2,1,40), np.logspace(-3,0,40)
grid = {'gamma': Gs,'C': Cs}
gridsearch = GridSearchCV(model, grid, cv = 5)
gridsearch.fit(X_train_scaled,y_train)
```

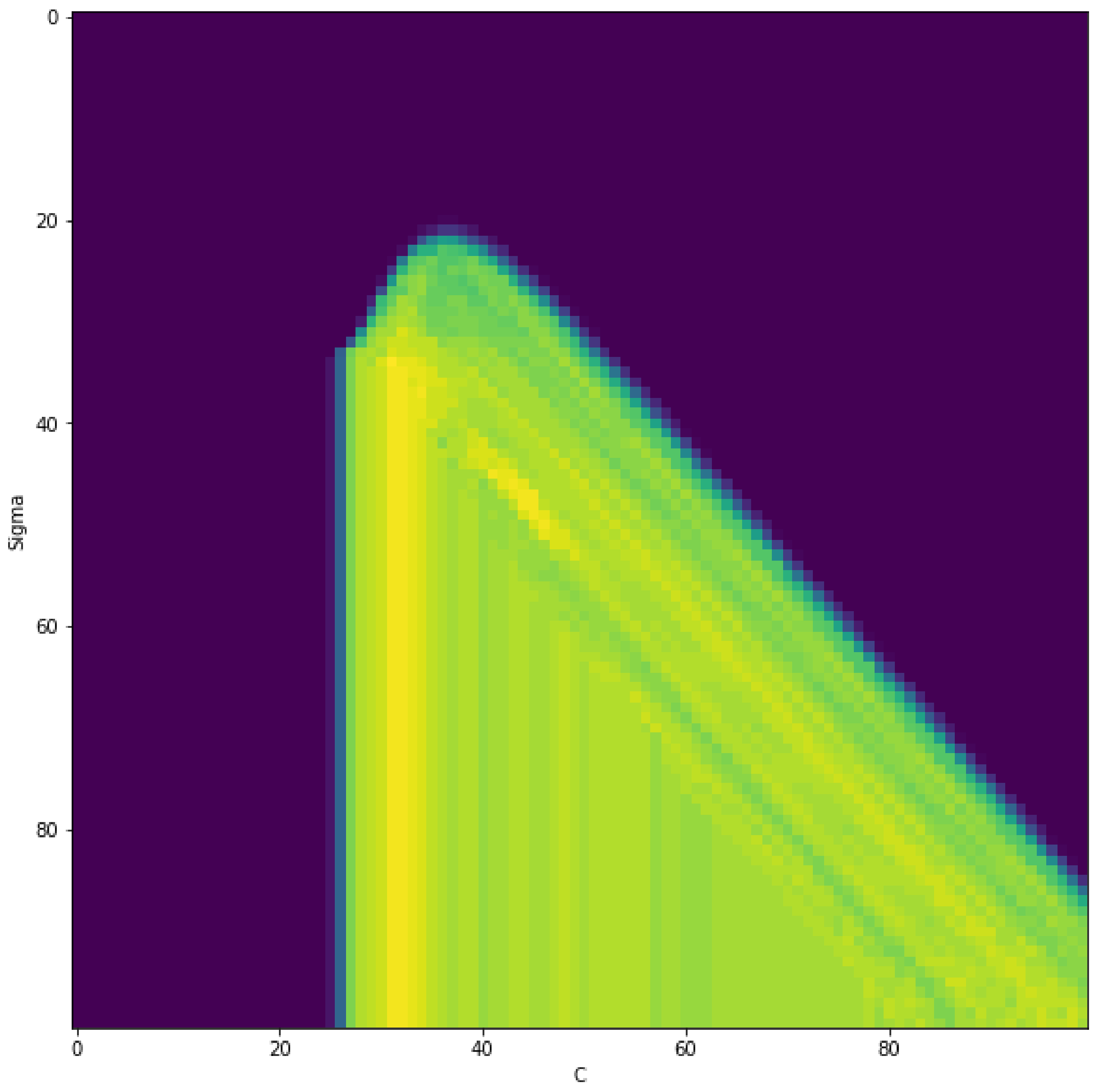

Figure 3: A rough visualization of the search space. Please don't look at the axes.

The optimal values as returned by the function were:
'C': 1.1937766417144369, 'gamma': 0.0345510729459222

This resulted in 0-1-losses of:
train 1.0
test 0.774390243902439

3.3

I expect the number of bounded support vectors to fall when C is increased (and vice versa). This will happen because giving the algorithm more 'slack' will allow for more support vectors to be correctly classified and lie close to the margin.

Only finding two more seemed boring, so here are some plots for 99 additional:

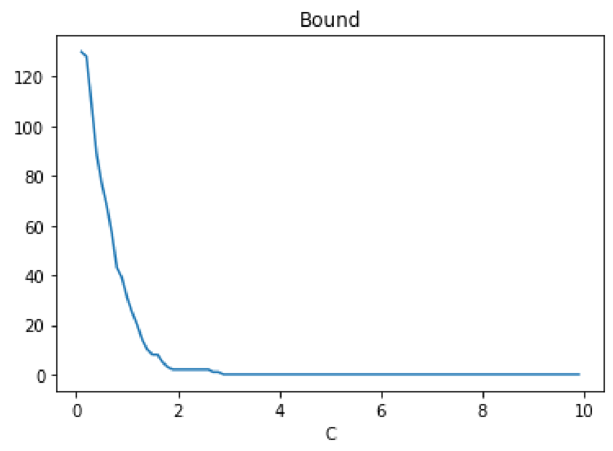

Figure 4: Bound

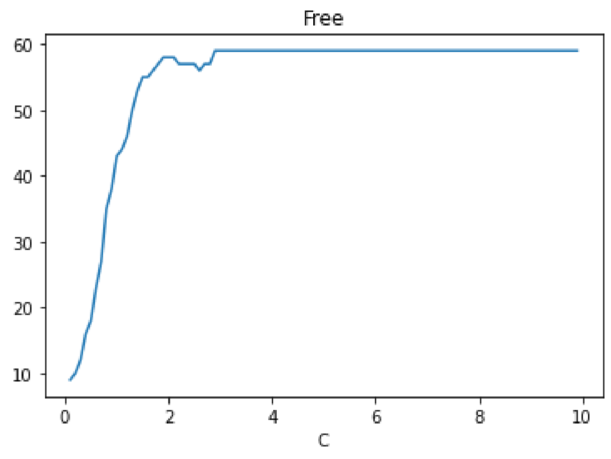

Figure 5: Free

```
free.append(sum(np.logical_and((np.array(c)<C)[0], (np.array(c)>0)[0])))
bounds.append(sum((np.abs(c)==C)[0]))
```