Assignment 3

Aleksandr Salo

Due April 9, 2015

1 Uniform convergence (20 points)

1. We proved that for any finite set of hypotheses $H = \{h_1, ..., h_k\}$, if we pick the hypothesis $\hat{h}$ that minimizes the training error on a set of m examples, then with probability at least $1 - \delta$:

$$\epsilon(\hat{h}) \leq (\min_i \epsilon(h_i) + 2\sqrt{\frac{1}{2m} log \frac{2k}{\delta}})$$

Consider a learning algorithm which, after looking at m training examples, chooses some hypothesis $\hat{h} \in H$ that makes zero mistakes on this training data. This process may yield a hypothesis that makes no mistakes in training, but still has non-zero generalization error. Show that with probability $(1 - \delta)$:

$$\epsilon(\hat{h}) \leq \frac{1}{m} log \frac{k}{\delta}$$

Proof:

(a) Let $\hat{h}$ be a bad classifier, that doesn't generalize very well:

$$\epsilon(\hat{h}) > \gamma$$

(b) We may choose this classifier, if for given training set S_m it has zero training error. The probability, that chosen classifier would **not** make an error on the one particular example is (by definition):

$$P(\text{Correct decision}) \leq 1 - \hat{\epsilon}(\hat{h}))$$

(c) Then the probability that the classifier would **not** fail for all the examples in training set is:

$$\begin{aligned} P(\text{All correct}) &\leq (1 - \hat{\epsilon}(\hat{h}))^m \\ &\leq (1 - \gamma)^m \\ &\leq e^{-\gamma m} \end{aligned}$$

(d) Now let us think about all the such bad classifiers out there and apply union bound. The probability that all (k) of them would **not** fail on any training example is (by union bound):

$$\begin{aligned} P &= k(1 - \gamma)^m \\ &\leq k e^{-\gamma m} \end{aligned}$$

We want that probability to be equal to δ, so:

$$k e^{-\gamma m} = \delta$$

(e) Let solve for γ:

$$\begin{aligned} ke^{-\gamma m} &= \delta \\ e^{-\gamma m} &= \frac{\delta}{k} \\ -\gamma m &= \log \frac{\delta}{k} \\ \gamma &= -\frac{1}{m} \log \frac{\delta}{k} \\ \gamma &= \frac{1}{m} \log \frac{k}{\delta} \end{aligned}$$

(f) Thus we showed that with probability $(1 - \delta)$:

$$\epsilon(\hat{h}) \leq \frac{1}{m} \log \frac{k}{\delta}$$

2. Let us take us mid stage formula from part 1 for a spin: $ke^{-\gamma m} = \delta$ and write the above bound in form of sample complexity bound:

$$\begin{aligned} -\gamma m &= \log \frac{\delta}{k} \\ m &= -\frac{1}{\gamma} \frac{\delta}{k} \\ m &= \frac{1}{\gamma} \log \frac{k}{\delta} \end{aligned}$$

Thus, for m bigger than this, $\epsilon(\hat{h}) \leq \gamma$ will hold the probability at least $1 - \delta$.

2 Mistake bounds (20 points)

Consider learning by selecting a hypothesis $h : X \to \{0, 1\}$ from a finite class H. Suppose we make two rather strong assumptions:

1. We assume that all our training data were correct (i.e. there is no noise). Thus, during training, if a hypothesis $h \in H$ ever misclassified an example, we know h must be wrong.
2. We assume that the correct hypothesis exists in H.

Consider now the following algorithm which observes a sequence of examples from a sample S, one at a time. For each example, it makes a prediction using a majority vote of the remaining hypotheses. It then discards incorrect hypotheses based on their individual correctness. Note that this algorithm makes predictions 'in real time', one example at a time, rather than examining all the examples first.

1. Prove that the following inequality holds:

$$numMistakes \leq log_2(|H|)$$

Proof

(a) Let us consider the extreme case: on the first iteration all the hypothesis guessed incorrectly. Then we have $numMistakes = 1$ and all the hypothesis eliminated - that is the algorithm finished.

(b) Thus the maximum number of mistakes would be reached when hypothesis predictions spitted almost equally. To be precise: $|h_{incorrect}| = |h_{correct}| + 1 + 1\{|H|iseven\}$.

(c) In that case, on each step we would do one mistake and decrease overall number of hypothesis by half. The total number the equals to $\log_2(|H|)$.

(d) Since we can do only one mistake at a step, the maximum number of mistakes is bound by the number of steps:

$$numMistakes \leq \log_2(|H|)$$

Note, that if we have small number of examples, then we finish before we found a correct hypothesis. In that case numMistakes would be even smaller.

2. What is the least number of mistakes that the algorithm might make before getting down to $|H| = 1$?
Proof

(a) Let us take arbitrary large number of hypothesis, say $|H| = 64$.

(b) Then on the first step of the algorithm it is possible, in the best case, that exactly half of our hypothesis guess correctly and another half - incorrectly. Then, we meet the condition $\{s \geq \frac{|H|}{2}\}$ of getting correct overall prediction by majority vote. On the next step eliminate exactly half of the hypothesis that guessed incorrectly.

(c) Now we have $|H| = 32$ and $numMistakes = 0$. Obviously we can continue this process until only one best hypothesis left.

(d) Due to the assumptions (that all the data samples are correct and that the correct hypothesis exists in H), after we have only one h it would work perfectly on all the other training data and would do no mistakes

(e) Thus the answer is: the algorithm might make **no** mistakes before it finishes.

3 Experiments with uniform convergence and mistake bounds (20 points)

First we generate the data (N = 500):

```
X_0 = pd.DataFrame([(x, -(N/2 - x) ** 1/2 * (random.random()+0.5) + 50, 0) for x in xrange(N/2)],
    columns=['x1','x2','y'])
X_1 = pd.DataFrame([(x, x ** 1/2 * (random.random()+0.5), 1) for x in xrange(N/2)],
    columns=['x1','x2','y'])
X = X_0.append(X_1,ignore_index = True);
```

And find correct hypotheses $\hat{h}_\theta = -9 + 0.25x_1 < x_2$

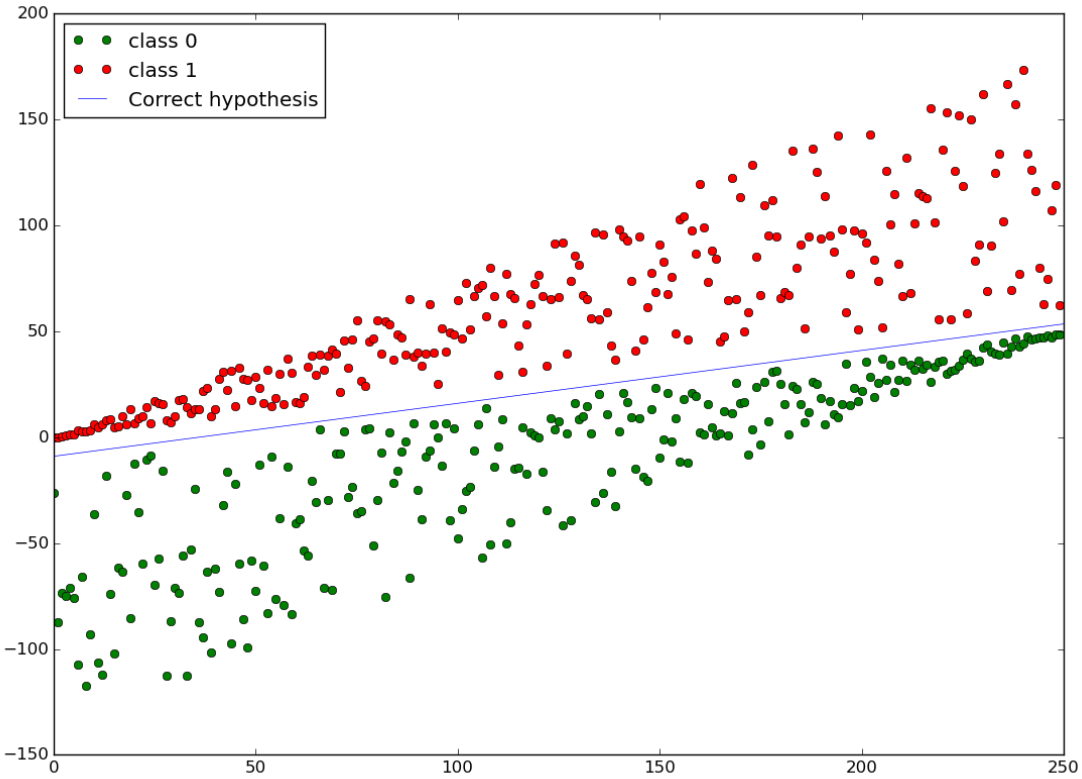

Figure 1: Generated data

Then we define a function, that generates random hypotheses in such a fashion that some of them might and would work for some test samples:

```
def generateH(num):
    H = [[-9 * (random.random() + 0.5)*5, 0.25*(3*random.random() - 0.5)] for i in xrange(num)]
    H.append([-9, 0.25]) #add correct hypothesis
    return H
```

Now using Halving algorithm, which defined as following:

```
def Halving(H, S, verbose=True):
    numMistakes = 0
    for i in S:
        votes = 0
        for h in H:
            votes +=  int(X.x2[i] > h[0] + X.x1[i] * h[1])
        #Derive majority vote
        p = int(votes >= len(H)/2.0)
        numMistakes += int(p <> X.y[i])
        #delete incorrect hypothesis
        [H.remove(h) for h in H if int(X.x2[i] > h[0] + X.x1[i] * h[1]) <> X.y[i]]
    if verbose:
        print 'h_cap:', H, 'numMistakes: ', numMistakes
    return numMistakes
```

We test our mistake bound $numMistakes \leq log_2(|H|)$ by running the test routing for random test samples (n=0.3 * N) derived from the same distribution (N=500):

```
def Test(N_RUN, verbose=True):
    mistakes = []

    #make 100 runs
    for i in xrange(N_RUN):
        #Define test sample space
        test_i = random.sample(xrange(N), N * 3 / 10)
        H_copy = [h for h in H]
        #Run Halving algorithm on test sample space and keep numMistakes
        mistakes.append(Halving(H_copy, test_i, False))

    if verbose:
        print 'Mistakes: ', mistakes
        print "Mean mistake: %0.2f (sd = %0.2f); Mistake bound: %d; Empirical Risk Bound: %0.2f"
            % (np.mean(mistakes), np.std(mistakes), math.ceil(np.log2(len(H))),
                np.log2(len(H)/delta)/150)
        return mistakes, np.mean(mistakes), np.std(mistakes)
    return mistakes, np.mean(mistakes), np.std(mistakes)
```

To see a dynamic changes in mistake bound we run this routine several times for various set of hypotheses (with various number of them) and take the average:

```
runs_h = [1, 5, 10, 30, 50, 80]
numMistake_means = []; numMistake_sds = []; mistake_bound = []; emp_risk_bound = []
for i, h_num in enumerate(runs_h):
    H = generateH(h_num)
    mistakes, mean, sd = Test(N_RUN=10)

    numMistake_means.append(mean)
    numMistake_sds.append(sd)
    mistake_bound.append(math.ceil(np.log2(len(H))))
    emp_risk_bound.append(np.log2(len(H)/delta)/150)

    plot() #plot the data
```

On the figure we can see how good the hypotheses work on average on random samples derived from the same distribution:

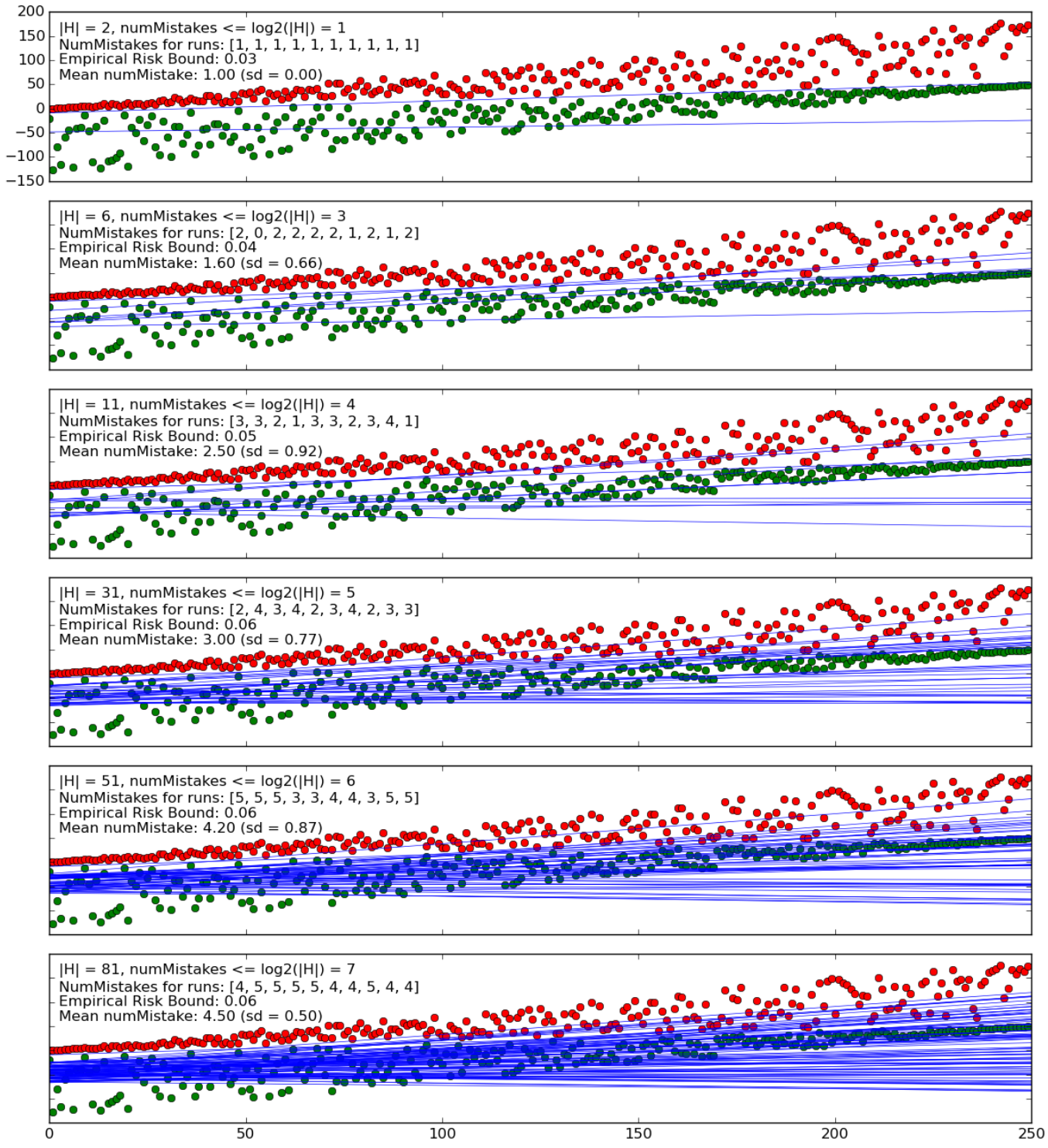

Figure 2: Test of "Halving" algorithm for various set of hypotheses

Now we can analyze how well our theoretical bound on the number of mistakes $numMistakes \leq log_2(|H|)$ correlates with the practical results:

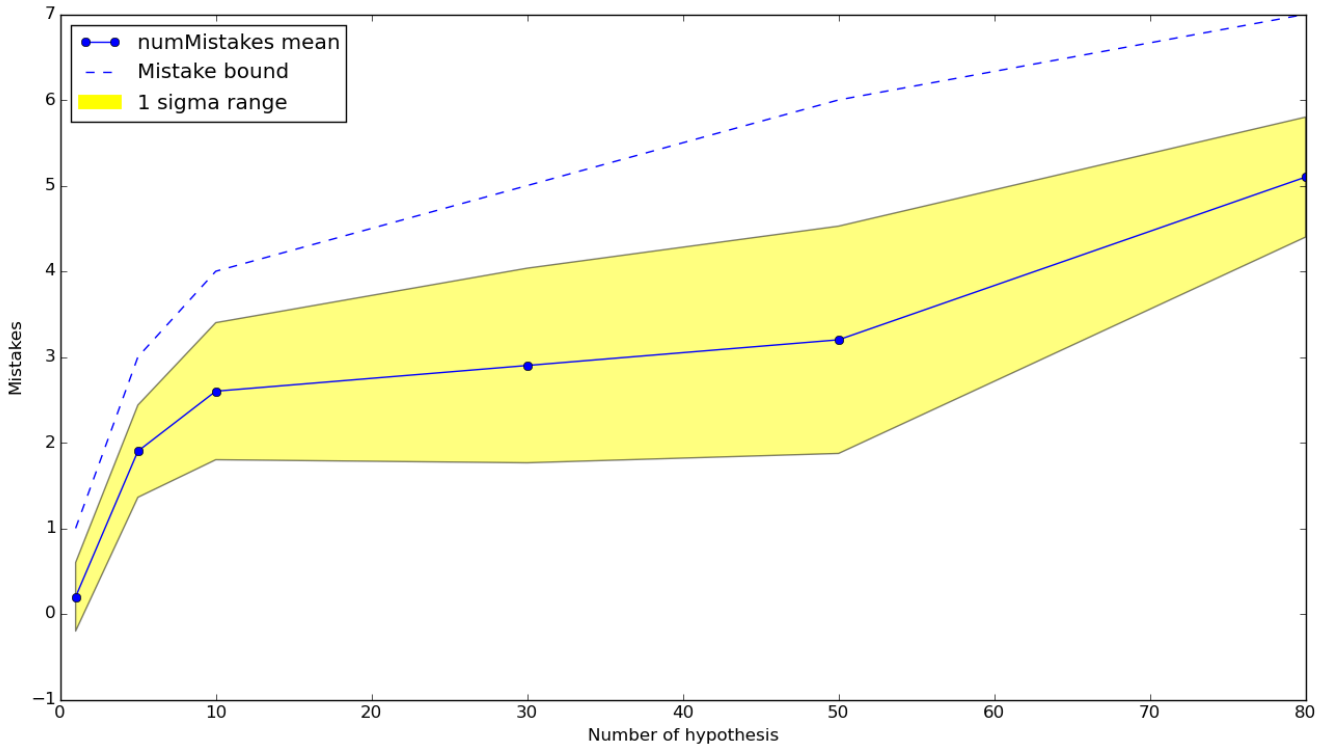

Figure 3: Average number of mistakes as a function of $|H|$

From figure 3 we can conclude the following:

1. Number of mistakes grows with the increase of the set of hypotheses, which is logical.
   However, and this is important caveat, **it does not necessarily does so**. For example, if we generate a million of totally wrong hypotheses - all of them would be rejected at the first step of the Halving algorithm. Thus this increase is only possible, when wrong hypothesis (in a sense, that they do some mistakes on all the examples of the distribution) can perform without mistakes on some subset of examples derived from that distribution.

2. As we proven in the section above, number of mistakes could not be more that $log_2(|H|)$, since each time we make a mistake we end up deleting at least a half of the hypotheses.
   Thus our test confirms the analytical bound.

We can also plot the empirical risk bound (line is derived as $N * \epsilon(\hat{h})$)

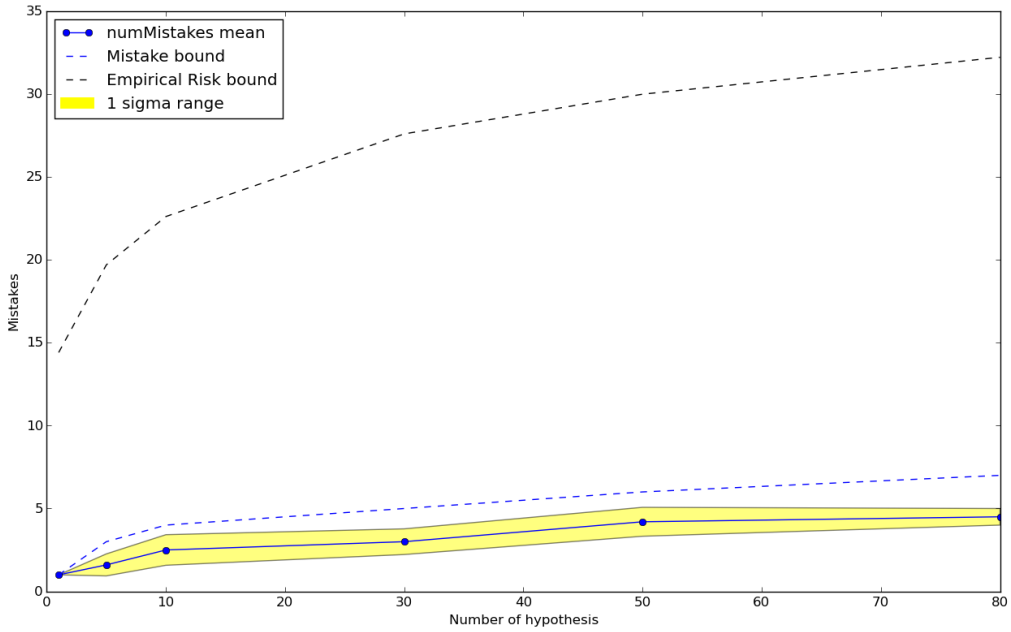

Figure 4: Empirical risk bound $\epsilon(\hat{h}) \leq \frac{1}{m} log \frac{k}{\delta}$ for $\delta = 0.1$

We can clearly see that empirical risk bound is less tight than bound for Halving algorithm.

Now, what if instead of finite hypotheses set and choosing, we **did** actually learn? Let us try to learn on subsets with various sizes of all the examples of the given distribution and keep track of how well our optimal (on training data) hypothesis would perform on the entire set of data.

```
random.seed(123)
RUNS = 10
max_sample_size = 20
sample_start = 1
results = np.zeros([RUNS, max_sample_size - sample_start + 1])
for run in xrange(RUNS):
run_results = []
for size in xrange(sample_start, max_sample_size+1):
    test_i = random.sample(xrange(N/2), size) + random.sample(xrange(N/2, N), size)
    test_X = X.ix[test_i]

    #learn the best hypothesis for sample data
    clf = linear_model.LogisticRegression()
    clf.fit(test_X[['x1','x2']], test_X[['y']])

    num_mistakes = 0
    #test chosen hypothesis on all the data
    for i in X.index:
    if int(sigmoid(np.dot([X.x1[i], X.x2[i]], clf.coef_.T) + clf.intercept_ ) > 0.5) <> X.y[i]:
        num_mistakes += 1
        run_results.append(num_mistakes)
        results[run] = run_results
```

Again, we see how the empirical risk bound is coherent with our data experiments. That also holds true for

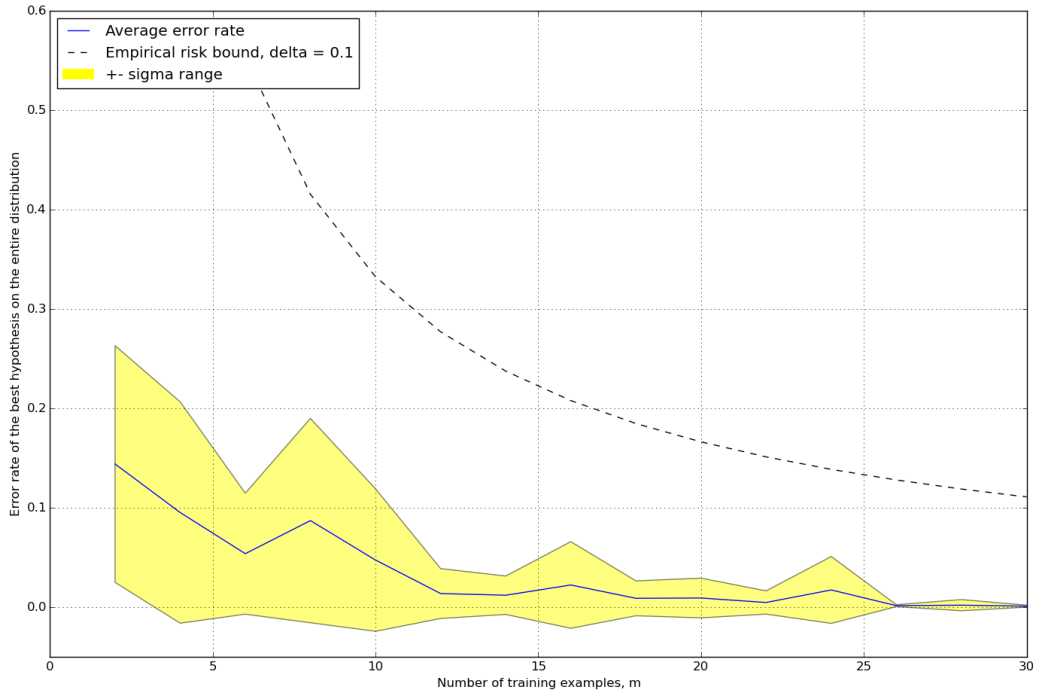

Figure 5: Empirical risk bound $\epsilon(\hat{h}) \leq \frac{1}{m} log \frac{k}{\delta}$ $(\delta = 0.1)$ for Linear Regression Classifier

Halving algorithm:

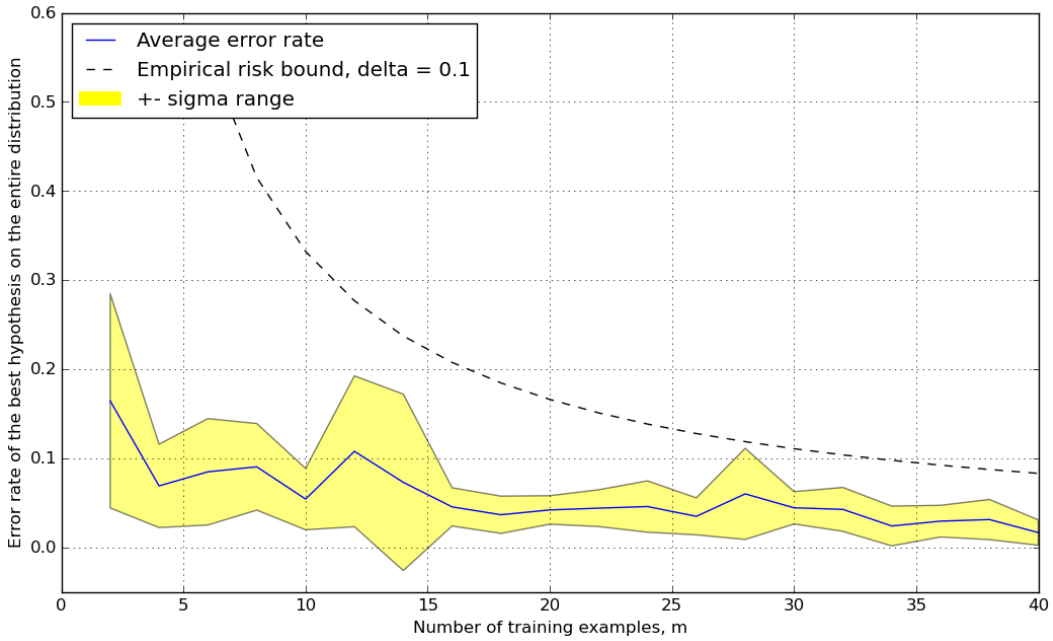

Figure 6: Empirical risk bound $\epsilon(\hat{h}) \leq \frac{1}{m} log \frac{k}{\delta}$ $(\delta = 0.1)$ for Halving algorithm

4 ℓ_2 norm soft margin SVMs (20 points)

New algorithm is given by the following optimization problem (notice that the slack penalties are now squared):

$$\min_{w,b,\xi} \frac{1}{2}||w||^2 + \frac{C}{2}\sum_{i=1}^{m}\xi_i^2$$
$$\text{s.t. } y^{(i)}(w^T x^{(i)} + b) \geq 1 - \xi_i, i = 1, ..., m$$

(a) Notice that we have dropped the $\xi_i \geq 0$ constraint in the ℓ_2 problem. Show that these non-negativity constraints can be removed. That is, show that the optimal value of the objective will be the same whether or not these constraints are present.
Proof:

1. Obviously, the quadratic term of the minimization is indifferent to the negativity of ξ since $(-a)^2 = a^2$.
2. If $\xi_i \leq 0$ then we can set $\xi_i = 0$ to satisfy easier the margin constraint $y^{(i)}(w^T x^{(i)} + b) \geq 1 - \xi_i$ which in turn makes the objective better. Thus, the optimal solution would never get $\xi < 0$ in the first place.
Explanation: consider related Lagrangian term: $-\sum_{i=1}^{m}\alpha_i(y^{(i)}(x^T w + b) - 1 + \xi_i)$. In order to maximize it (given non-positive ξ) we need to use $\xi = 0$.

(b) What is the Lagrangian of the ℓ_2 soft margin SVM optimization problem?
By definition:

$$\mathcal{L}(w, b, \xi, \alpha, r) = \frac{1}{2}||w||^2 + \frac{C}{2}\sum_{i=1}^{m}\xi_i^2 - \sum_{i=1}^{m}\alpha_i(y^{(i)}(x^T w + b) - 1 + \xi_i)$$

Note, that there is no longer $\sum_{i=1}^{m}\xi_i$ term since we eliminated that constraint.

(c) Find the critical (minimal) points of the Lagrangian with respect to w, b, and ξ by taking the following gradients: $\nabla_w \mathcal{L}$, $\frac{\partial \mathcal{L}}{\partial b}$, $\nabla_\xi \mathcal{L}$, and setting them equal to 0. Here $\xi = [\xi_1, ..., \xi_m]^T$.

$$\nabla_w \mathcal{L} = w - \sum_{i=1}^{m}\alpha_i y^{(i)} x^{(i)} = 0 \rightarrow w = \sum_{i=1}^{m}\alpha_i y^{(i)} x^{(i)}$$
$$\frac{\partial \mathcal{L}}{\partial b} = \sum_{i=1}^{m}\alpha_i y^{(i)} = 0$$
$$\nabla_\xi \mathcal{L} = C\xi_i - \alpha_i = 0 \rightarrow \xi_i = \alpha_i \frac{1}{C}$$

(d) What is the dual of the ℓ_2 soft margin SVM optimization problem? Let us now plug back derived on the previous step w, ξ, b:

$$\mathcal{L} = \frac{1}{2}\sum_{i=1}^{m}\frac{\alpha_i}{\xi_i}\xi_i^2 + \frac{1}{2}\sum_{i,j=1}^{m}(\alpha_i y^{(i)} x^{(i)})^T(\alpha_j y^{(j)} x^{(j)}) - \sum_{i=1}^{m}\alpha_i[y^{(i)}((\sum_{j=1}^{m}\alpha_j y^{(j)} x^{(j)})^T x^{(i)} + b) - 1 + \xi_i]$$
$$= \frac{1}{2}\sum_{i=1}^{m}\alpha_i\xi_i - \frac{1}{2}\sum_{i,j=1}^{m}\alpha_i\alpha_j y^{(i)} y^{(j)}(x^{(i)})^T x^{(j)} - (\sum_{i=1}^{m}\alpha_i y^{(i)})b + \sum_{i=1}^{m}\alpha_i - \sum_{i=1}^{m}\alpha_i\xi_i$$
$$= \sum_{i=1}^{m}\alpha_i - \frac{1}{2}\sum_{i,j=1}^{m}\alpha_i\alpha_j y^{(i)} y^{(j)}(x^{(i)})^T x^{(j)} - \frac{1}{2}\sum_{i=1}^{m}\frac{\alpha_i^2}{C}$$

Note that at the last step we used the facts (derived on the **c** step) that $\sum_{i=1}^{m} \alpha_i y^{(i)} = 0$ and $\xi_i = \alpha_i \frac{1}{C}$ Thus the dual form:

$$\max_{\alpha} \sum_{i=1}^{m} \alpha_i - \sum_{i,j=1}^{m} \alpha_i \alpha_k y^{(i)} y^{(j)} \langle x^{(i)}, x^{(j)} \rangle - \frac{1}{2} \sum_{i=1}^{m} \frac{\alpha_i^2}{C}$$

$$\text{s.t. } \alpha_i \geq 0, i = 1, ..., m$$

$$\sum_{i=1}^{m} \alpha_i y^{(i)} = 0$$

5 VC dimension (20 points)

Let the input domain of a learning problem be $\mathcal{X} = \mathbb{R}$. Give the VC dimension for each of the following classes of hypotheses. In each case, if you claim that the VC dimension is d, then you need to show that the hypothesis class can shatter d points, and explain why there are no sets of d + 1 points that it can shatter.

Note, that VC(h) is the size of the largest set shattered by h. H shatters set s if h can realize any labeling on s. In other words, VC dimension is a number of training points that can be classified *exactly*.

1. $h(x) = 1\{a < x\}$, with parameter $a \in \mathbb{R}$
   With this hypothesis we can shatter any 1 point, but for one particular case of 3 points we **can't**:

Figure 7: VC $h(x) = 1\{a < x\}$

   That is if we switch the positions of two points we misclassify. **Thus, VC(h) = 1**

2. $h(x) = 1\{a < x < b\}$, with parameters $a, b \in \mathbb{R}$
   With this hypothesis we can shatter any 2 points, even those we had troubles with previous h, but for one particular pattern of 3 points we **can't**:

Figure 8: VC $h(x) = 1\{a < x < b\}$

   That is if we for example, have $x_1 < x_2 < x_3$ and class labels are 1, 0, 1 respectively, then we can't capture both class-1 examples. **Thus, VC(h) = 2**

   Naturally, if we extrapolate the hypothesis to the extended version:

$$h(x) = \begin{cases} x < a_1 \\ a_1 < x < a_2 \\ a_2 < x < a_3 \\ \dots \\ a_d < x \end{cases}$$

   Then we can clearly see that this hypothesis would shatter any d training points.
   Thus, VC(h) = d

   That is, given any sample of points on the line, we can create a sufficient number of intervals to realize any labeling.

Obviously enough, there is no upper bound on number of points that could be shattered by sin wave. But the question of the VC dimension is the question of what is the number of examples that sin wave would be able to realize **any** labeling of them?

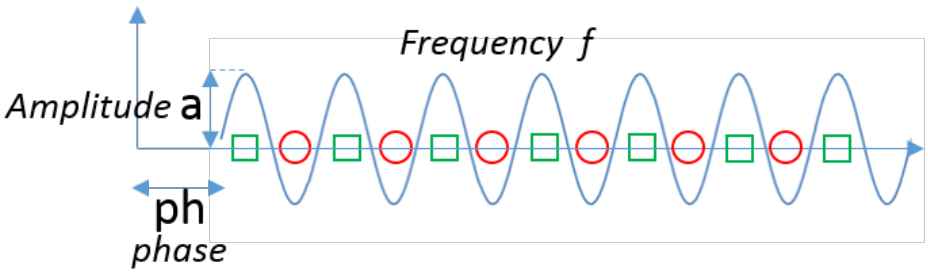

Figure 9: VC for sin weve

3. $h(x) = 1\{asin(x) > 0\}$, with parameter $a \in \mathbb{R}$
   Be as infinite as it is, sin wave with ability to change it **amplitude** is useless in shattering even two training points in this case:

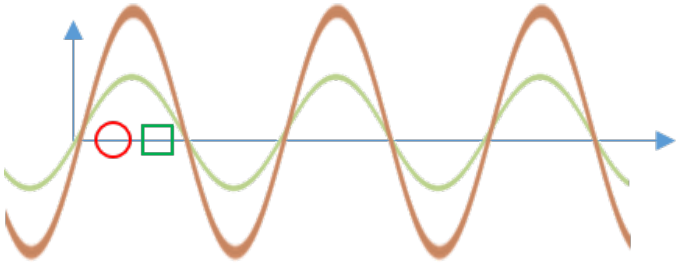

Figure 10: VC $h(x) = 1\{asin(x) > 0\}$

Thus, VC(h) = 1.

4. $h(x) = 1\{sin(a + x) > 0\}$, with parameters $a \in \mathbb{R}$
   What about the sin wave with ability to change it **phase**? It seem like we were able to shatter the previous case, yet more examples make this h stumble.

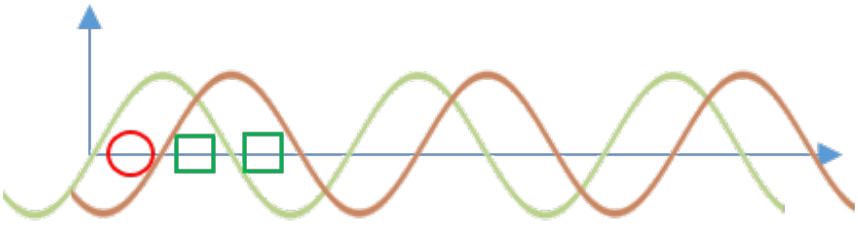

Figure 11: VC $h(x) = 1\{sin(a + x) > 0\}$

Thus, VC(h) = 2.

5. **(extra credit)** $h(x) = 1\{sin(ax + phase) > 0\}$, with parameters $a \in \mathbb{R}$
   What about the sin wave with ability to change it **frequency** and phase? It seems like given sample of points on the line, we can choose a sufficiently large value for frequency and appropriate phase to realize any labeling. That result is similar to the case of large number of intervals for the line.

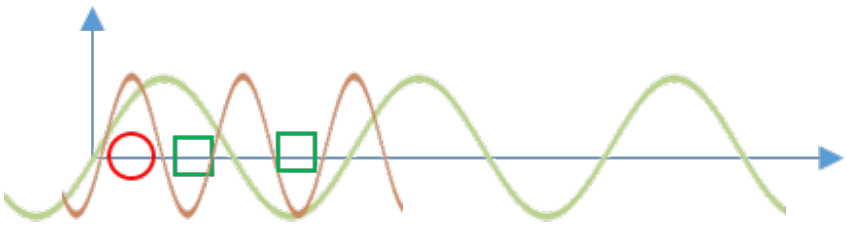

Figure 12: VC $h(x) = 1\{sin(ax + phase) > 0\}$

Thus, VC(h) = ∞