Assignment 3

1 The VC-dimension (50 points)

1.

By definition $d_{\mathrm{VC}}(\mathcal{H})$ satisfies $m_{\mathcal{H}}(d_{\mathrm{VC}}(\mathcal{H})) = 2^{d_{\mathrm{VC}}(\mathcal{H})}$. As $|\mathcal{H}| = M$ we know from assignment 1, that $m_{\mathcal{H}}(n) \leq M$ for any n. Combining this, yields

$$2^{d_{\mathrm{VC}}(\mathcal{H})} \leq M \Leftrightarrow d_{\mathrm{VC}}(\mathcal{H}) \leq \log_2(M)$$

Which is our upper bound.

2.

Per assignment 1, we know that since $|\mathcal{H}| = 2$ we have $m_{\mathcal{H}}(n) = 2$. The maximum n that satisfies $m_{\mathcal{H}}(n) = 2 = 2^n$, is clearly $n = 1$. It follows that $2^{d_{\mathrm{VC}}(\mathcal{H})} = 1$.

5.

For the bound in theorem 3.16 to be non-vacuous, the square root term has to be less than 1, as we know that the binary loss is bounded by 1. As we are doing an order of magnitude calculation, we drop constants. We then have that,

$$\sqrt{\frac{\log(n^{d_{\mathrm{VC}}})}{n}} \leq 1 \Leftrightarrow d_{\mathrm{VC}} \leq \frac{n}{\log(n)}$$

Which should (in orders of magnitude) be the relationship between d_{VC}, to ensure that the bound is non-trivial.

6.

We compare the square root terms from theorem 3.2 and theorem 3.16. We can remove the square roots, as $x \mapsto \sqrt{x}$ is monotonic. This leads us to compared

$$\frac{\ln M/\delta}{2n} \quad \text{v.s.} \quad \frac{8\ln\left(2\left((2n)^{d_{\mathrm{VC}}}+1\right)/\delta\right)}{n}$$

We have from exercise 1 that $d_{\mathrm{VC}} \leq \log_2(M)$. We can thus lower-bound the theorem 3.16 expression by,

$$\frac{8\ln\left(2\left((2n)^{d_{\mathrm{VC}}}+1\right)/\delta\right)}{n} \geq \frac{\ln\left(\left((2n)^{\log_2(M)}\right)/\delta\right)}{2n}$$

Where we have also divided by 16, divided by 2 inside the logarithm, and subtracted 1 - operations which all make the expression smaller. As the logarithm has base 2,

$$\frac{\ln\left(\left((2n)^{\log_2(M)}\right)/\delta\right)}{2n} = \frac{\ln\left(\left(2^{\log_2(M)}n^{\log_2(M)}\right)/\delta\right)}{2n} = \frac{\ln\left(\left(Mn^{\log_2(M)}\right)/\delta\right)}{2n}$$

Now notice that $n^{\log_2(M)} \geq 1$ whenever $M \geq 1$, which we can safely assume (otherwise $\mathcal{H} = \emptyset$). Therefore,

$$\frac{\ln\left(\left(Mn^{\log_2(M)}\right)/\delta\right)}{2n} \geq \frac{\ln(M/\delta)}{2n}$$

We have therefore shown that,

$$\frac{\ln M/\delta}{2n} \leq \frac{8\ln\left(2\left((2n)^{d_{\mathrm{VC}}}+1\right)/\delta\right)}{n}$$

For any reasonable choice of M and n. This shows that the theorem 3.2 bound is always as least as good, and probably a fair bit better, since we are using preatty coarse inequalities in our reasoning, than the theorem 3.16 bound. This is not overly suprising, as theorem 3.2 is tailored specifically to finite hypothesis classes, whereas theorem 3.16 is more general.

7.

From theorem 3.16, we have that with probability at least $1-\delta$,

$$L(h) - \hat{L}(h,S) \leq \frac{8\ln\left(2\left((2n)^{d_{\mathrm{VC}}}+1\right)/\delta\right)}{n}$$

We also know that the VC-dimension of linear classifiers in $\mathbb{R}^{10}$ is 11. Pluggging this into the above inequality, as well as $\delta = 0.01$, we have that with 99% confidence,

$$L(h) - \hat{L}(h,S) \leq \frac{8\ln\left(2\left((2n)^{11}+1\right)/0.01\right)}{n}$$

We are interested in ensuring that the right hand side in the above inequality is less than 0.01, as the empirical loss will not underestimate the true loss with more than 0.01 with at least 99% confidence in this case. The bound is decreasing in n, so we can write a short while-loop, that terminates when the right hand side is less than 0.01. This yields that we need $n = 15609626$ samples.

8.

We want to show that we can come up with a configuration of 3 points in $\mathbb{R}^2$, such that we can label these in all 8 ways using positive circles. Let $x_1 = (-1,0), x_2 = (0,0)$ and $x_3 = (1,0)$. Labelling everything as positive, choose the circle with $c = (0,0)$ and $r = 2$. Labelling everything as negative, choose $c = (10,10)$ and $r = 1$. For labelling 1 point as positive and 2 as negative, choose $x_1, x_2, x_3 = c$, respectively and choose $r = 0.5$. For labelling x_i, x_j as positive, $i \neq j$, and the last point as negative, choose $c = \frac{x_i+x_j}{2}$ and $r = 1$. This generates all $2^3 = 8$ possible labellings of x_1, x_2, x_3, and we thus have $d_{\mathrm{VC}}(\mathcal{H}_+) \geq 3$.

9.

Consider adding a point x_4 to the above configuration. For each of the circles described above, the point is either inside or outside the circle. As we have 8 labellings above, this generates 8 labellings for x_1, x_2, x_3, x_4. Now, consider that we can "invert" these 8 labellings by changing each circle from a positive to a negative circle. This generates 8 new labellings, such that we have all $2^4 = 16$ possible labellings. This shows that $d_{\text{VC}}(\mathcal{H}_+ \cup \mathcal{H}_+) \geq 4$.

11.

We show that $d_{\text{VC}}(\mathcal{H}_d) = d + 1$ by induction after d.

Induction start: Assume $d = 0$. Then we have a single leaf node as our tree. This can label exactly 0+1 = 1 point as 1 or −1.

Induction step: Assume that $d_{\text{VC}}(\mathcal{H}_d) = d + 1$. Consider adding a point, and one layer of depth to the tree. The 2^{d+1} possible original labellings increase to $2^{d+2} = 2 \cdot 2^{d+1}$ possible labellings while the number of leaf nodes double. Per the induction hypothesis, $\mathcal{H}_d$ shatters $d + 1$ points. As the number of leaf nodes have doubled, the set of trees of depth $d + 1$ can now shatter one more point, showing that $d_{\text{VC}}(\mathcal{H}_{d+1}) = d + 2$. By the principle of simple induction, $d_{\text{VC}}(\mathcal{H}_d) = d + 1$.

12.

We have that $\mathcal{H} = \cup_{i=1}^{\infty} \mathcal{H}_i$. Given n points, we can choose the subset $\mathcal{H}_{n-1} \in \mathcal{H}$ that shatters n points. Since this holds for any n, we have that $d_{\text{VC}}(\mathcal{H}) = \infty$.

13.

Since our input space is the unit ball, and our hypothesis space is the space of linear seperators, we can apply theorem 3.22. Since we have achieved a margin of $\gamma = 0.1$, we have $||w|| \leq 10$. By theorem 3.22, we then have for our hypothesis h, that with probability at least 0.99,

$$L(h) \leq L_{\text{FAT}}(h) \leq \hat{L}_{\text{FAT}}(h, S) + \sqrt{8 \frac{\log(2((2 \cdot 100000)^{1+100}) + 1)(100 + 1) \cdot 100/0.01}{100000}} = 0.01 + 0.3158901005 = 0.3258901005$$

14.

Define the probability distribution of (X, Y) such that $\mathbb{P}(X = 1, Y = 1) = p$, $\mathbb{P}(X = 1, Y = -1) = 1 - p$, for $p \in [0, 1]$. That is, X takes the value 1 with probability 1 and Y is Bernoulli distributed with bias p. Let $\mathcal{H}$ be the set of 1-nearest neighbors classifiers on $\mathbb{R}$, where ties are broken at random. We have seen at lectures that $d_{\text{VC}}(\mathcal{H}) = \infty$. Let S be a sample of size n consisting of $(X_i, Y_i)_{i=1 \cdots n}$ sampled independently from the distribution generating (X, Y).

Notice that because the value of X_i is 1 with probability 1, when given a new sample (X', Y'), h will choose a random X_i, and return Y_i. The probability that Y_i is classified wrongly is then the probability of observing two i.i.d. Bernoulli variables being different. That is, we can compute the expected loss of any h as,

$$L(h) = \mathbb{P}(Y \neq Y') = \mathbb{P}(Y = 1, Y' = -1) + \mathbb{P}(Y = -1, Y' = 1) = 2p(1 - p)$$

Denote $q = 2p(1-p)$. Notice that the empirical loss of h on S, is actually unbiased as h is independent of S. Notice also, that the empirical loss in this special case of 1 nearest neighbors is actually not zero, because given X_i, the model does not necesarrily predict Y_i. The empirical loss on S is then binomially distributed with probability q and size parameter n. By Chebyshevs inequality we have that,

$$\mathbb{P}(|X - \mathbb{E}(X)| \geq \epsilon) \leq \frac{\mathbb{V}(X)}{\epsilon^2}$$

Or by setting the RHS equal to δ,

$$\mathbb{P}(\mathbb{E}(X) \geq \sqrt{\frac{\mathbb{V}(X)}{\delta}} + X) \leq \mathbb{P}(|X - \mathbb{E}(X)| \geq \sqrt{\frac{\mathbb{V}(X)}{\delta}}) \leq \delta$$

Setting $L(h) = E(X)$ and $X = \hat{L}(h, S)$, we have that with probability at least $1 - \delta$,

$$L(h) \leq \hat{L}(h, S) + \sqrt{\frac{\mathbb{V}(\hat{L}(h, S))}{\delta}}$$

From the distribution of the empirical loss, we have that

$$\mathbb{V}(\hat{L}(h, S)) = \frac{1}{n^2}(n \cdot q(1 - q)) = \frac{q(1 - q)}{n}$$

We then have that, with probability $1 - \delta$,

$$L(h) \leq \hat{L}(h, S) + \sqrt{\frac{q(1 - q)}{n\delta}}$$

plugging in $n = 100$ and $\delta = 0.05$, we want to ensure that the square root term is less than 0.01 by choosing q, that is we want to solve

$$\sqrt{\frac{q(1 - q)}{5}} \leq \frac{1}{100}$$

This is achived by setting $q \leq 0.0005005005$, which is in turn achieved by setting $p \geq 0.999499$. This clearly also holds when n gets larger than 100, as the variance is decreasing in n. By setting $p = 0.9995$ in the above described distribution, we therefore have that with probability at least 0.95,

$$L(h) \leq \hat{L}(h, S) + 0.01$$

for all $h \in \mathcal{H}$. (I suspect this is not really how the question is supposed to be solved - can it be done simpler?)

2 Variable Stars (50 points)

2.1 Data understanding and preprocessing (8 points)

The number of training examples is 150 and the number of test examples is 164. In the training examples, the frequency of class −1 is 0.453, and the frequency of class 1 is 0.545.

We normalize the data to zero mean and unit variance by computing the mean and standard deviation (more precisely, the in-sample standard deviation that does not use Bessel-correction) of each feature. We then from each feature subtract the mean and divide by the standard deviation of that feature.

Letting μ denote the vector of means of the features (in the training data), and D the diagonal matrix with the standard deviation of the features (in the training data) along the diagonal. Then the function f_{norm} that outputs the normalised matrix, given μ and D is.

$$f_{\text{norm}}(X) = XD - \mu D$$

We implement this function below

```
mu = np.mean(X_train, axis=0) # Compute mean of each feature
sd = np.std(X_train, axis=0, ddof = 0) # Compute (in-sample) standard deviation of each feature
def normalize(X,mean,sd): # Function to normalize array given mean and standard deviation
     D = np.diag(1/sd) # diagonalize std deviation vector
     return X @ D - mu @ D

X_train_norm = normalize(X_train, mu, sd) # Normalized training data
X_test_norm = normalize(X_test, mu, sd) # Normalized testing data

vars = np.var(X_test_norm, axis=0,ddof=0) # variance of test data
means = np.mean(X_test_norm, axis=0) # mean of test data

mean_var = np.array([means, vars]).T # Combine to one array
np.savetxt("mean_var.csv", mean_var, delimiter=",") # Send to csv for later use
```

This gives us the following means and variances of the transformed training and test data,

Table 1: Mean and variance of normalized test data

Mean	Variance	Feature
0.0904	1.9290	0
0.1658	7.2771	1
-0.0632	0.7855	2
-0.0802	0.7411	3
-0.0379	0.8555	4
-0.1084	0.9803	5
-0.1047	1.0681	6
-0.2116	2.8828	7
0.2688	2.9717	8
0.0805	1.4777	9
0.0141	1.0939	10
0.0575	1.1381	11
0.0139	1.1177	12
0.0035	1.2449	13
0.1325	1.2656	14
0.0234	1.0073	15
0.1302	1.1311	16
0.1306	3.8939	17
0.0252	5.7140	18
0.1006	5.0024	19
0.4756	54.2813	20
0.1081	1.4396	21
0.0482	1.0228	22
-0.1166	0.9728	23
0.1106	0.8535	24
0.0175	1.1609	25
-0.1017	0.5946	26
-0.1267	0.8026	27
-0.1828	0.4041	28
0.0070	1.2242	29

Mean	Variance	Feature
-0.0334	1.0256	30
0.0050	0.9960	31
0.2002	1.0306	32
-0.0096	1.0695	33
-0.0777	0.8185	34
0.1720	4.9093	35
0.3010	11.0053	36
0.1774	0.9737	37
0.0475	0.8089	38
-0.0235	0.8894	39
0.0818	2.4354	40
0.2213	2.2149	41
0.0405	1.5090	42
-0.1162	0.8938	43
-0.0298	1.3206	44
0.1043	0.8156	45
0.1151	1.1993	46
0.0960	2.2325	47
-0.0689	1.2203	48
-0.0502	0.9333	49
-0.1321	1.1945	50
0.0353	1.3139	51
-0.0011	1.3930	52
0.0083	0.8647	53
0.2277	1.9402	54
-0.0358	1.0322	55
0.1374	1.0682	56
0.1366	1.2131	57
0.0415	1.7374	58
-0.0109	1.8710	59
-0.0553	1.0111	60

2.2 Model selection using grid-search (21 points)

We use the `SVC` from `sklearn.svm` as our implementation of the SVM. Note that this uses a ℓ_1 penalty on the regularization term according to the documentation on https://scikit-learn.org/stable/modules/svm.html#svm-classification section 1.4.7.1. We set the `kernel` paramter to 'rbf', for using a Gaussian kernel. We keep the rest of the parameters at their default values - we will use grid search to determine the `gamma` and `C` parameters.

For doing the grid-search we do 5-fold cross-validation using the `GridSearchCV` class from `sklearn.model_selection`, with the `cv` parameter set to 5. The `estimator` parameter is set to the instance of the `SVC` class described above. As candidates for C and γ we use $C = \{10^{-2}, \cdots, 10^{4}\}$ and $\gamma = \{10^{-4}, \cdots, 10^{2}\}$. We set the `param_grid` paramter equal to a dictionary containg C and γ. For `scoring` we use accuracy, as we are dealing with 0-1 loss. We set `refit` equal to True, such that the model returned by the `fit` method is trained on all the datapoints in the training data.

We extract the hyperparamters with the lowest cross-validation error by the `best_estimator_` attribute, and see that these are given by $C = 1000$ and $\gamma = 0.0001$. We calculate the errors on the training and test data, by computing 1 minus the `score` method applied to the training and test data respectively. We get the training loss: 0.0133333 and the test loss 0.335366.

2.3

Recall that the slack variables ξ_i measures the classification error of the soft-margin SVM on x_i, and that $\xi_i = 0$ iff x_i is classified correctly and has a distance at least $1/||w||$ to the seperating hyperplane. Now, if we let C get very small, the optimization problem can allow for very large slack, that is a large number of misclassifications, and instead focus on maximizing the margin.

According to Christian Igel's Kernel Based Method notes p. 37, the KKT complementary conditions from 1-norm soft margin SVM, imply that if x_i violates the margin, we have that $\alpha_i = C$. The combination of the penalty being small for misclassification, such that the algorithm optimizes for a large margin, leading to many misclassifications, combined with the fact that misclassifications imply $\alpha_i = C$, we see that the number of bounded support vectors will increase as C decreases.

We can verify this claim experimentally by fixing our kernel and kernel coefficient at $\gamma = 0.0001$, and letting C decrease by a factor of 10 (from $C = 1000$ which we found in 2.2), and for each of these values, counting the number of bounded and free support vectors. We do this for $C \in \{-10^{-3}, \cdots, 1000\}$. Practically we do this by fitting an SVM to the training data, for each value of C. We then extract the dual coefficient α for each support vector by the `dual_coef_` attribute of the `SVC` class. We then count how many of these have an absolute value equal to C, which gives us the number of bounded support vectors. We can then get the number of free support vectors by subtracting the number of bounded vectors from the number of support vectors. This is implemented in the `count_support_vectors` function below.

```python
# eval: false
def count_support_vectors(C, X, y, gamma):
    svm = SVC(C=C,kernel="rbf", gamma=gamma) # Initialize SVM
    svm.fit(X,y) # Fit svm
    alphas = svm.dual_coef_ # Get (signed) alphas
    bounded = np.sum(abs(alphas)>=C) #Check whether support vector is bounded
    free = alphas.size-bounded #Rest is free
    # return array containing C, #bounded support vectors and #free support vectors
    return np.reshape(np.array([C, bounded, free]),(1,3))
```

We get the following results,

Table 2: Number of bounded/free support vectors for different values of C and $\gamma = 0.0001$

C	Bounded SV's	Free SV's
1e-03	136	0
1e-02	136	0
1e-01	135	3
1e+00	121	12
1e+01	113	16
1e+02	67	37
1e+03	12	72

Which shows exactly what we argued.