CSE 546 SPRING 2021: HW1

Name: Honda Li

Short Answer and “True or False” Conceptual Questions

A.0

(A.0.a)

The bias and variance is similar to the concepts of precision and accuracy in Experimental Physics. Under the context of machine learning, Bias refers to part of the learning errors caused by a model being too simple, in a way that it just cant represent the joint probability density function with its simplicity.

The variance refers to the variance of the random variable $\hat{f}(x)$, which depends on the samples we observed.

Bias-variance trade off relates 2 types of learning errors (Bias, Variance) with the model complexity.

(A.0.b)

Usually, higher the model complexity, higher the variance, lower the model complexity, higher the bias.

(A.0.c)

False. The bias decreases. Because the bias is: $\mathbb{E}\left[\left(f(x) - \mathbb{E}\left[\hat{f}(x)\right]\right)^2\right]$. The number of sample seems to be irrelevant to the amount of bias we have for the model.

(A.0.d)

True when we fix the model complexity. This is absolutely true when we consider a linear regression. If we take infinitely many data, the best line-fit will converge.

This is even more true when we just consider taking the average as a way of making prediction (Which is not a bad way to predict the output given $X = x$), then, this is literally the Central limit theorem, as we have more and more samples, the variance of the sample average gets smaller.

(A.0.e)

Yes, and this is exactly the idea behind regularization, where it tries to change the objective of the optimization problem a bit, allowing the models to use less features and prevent it from over-fitting.

(A.0.f)

We should use the test set to tune the hyper-parameters for the models. This is true because the test set is indicative of true performance of the model.
Consider hypertunning the polynomial regression using training set only, where p is the degree of polynomial. Then, for any given sample that is normalized, there is a $p = n$ that where n is the total number of samples and th polynomial will fit perfectly. This is obviously over-fitting. Hence we should hypertune using the test data set.

(A.0.g)

False. It’s an underestimate. This is a conclusion from the lecture.

Maximum Likelihood Estimator(MLE)

(A.1)

(A.1.a)

Objective: Find the expression for the maximum=likelihood estimate for the parameter λ for the poisson distribution, interns of the goal count. Assume idd rvs.

Here we will assume that observations obtained takes the form $x_1, x_2, \cdots x_N$, and then we derive the best estimator for λ in this much general context.

I will shut up and just show you the math:

$$
\begin{aligned}
&\prod_{n=1}^{N} \text{Poi}(x_n|\lambda) \qquad \text{(A.1.a.1)}\\
&\sum_{n=1}^{N} \log\left(\text{Poi}(x_n|\lambda)\right)\\
&\sum_{n=1}^{N} \left(-\lambda + x_n \ln(\lambda) + \log(x_n!)\right)
\end{aligned}
$$

Notice that, only some of the terms are relevant to the parameter λ, therefore, the optimization problem we are solving is:

$$
\lambda^+ = \underset{\lambda}{\text{argmax}} \left\{-N\lambda + \ln(\lambda)\sum_{i=1}^{N}(x_n)\right\} \qquad \text{(A.1.a.2)}
$$

To solve it, we just take the derivative, set it to zero and then solve for λ, because this function is a function that has a single local maximum.

$$
\begin{aligned}
\partial_\lambda \left[-N\lambda + \ln(\lambda)\sum_{i=1}^{N}(x_n)\right] =&0 \qquad \text{(A.1.a.3)}\\
-N + \frac{\sum_{n=1}^{N} x_n}{\lambda^+} =&0\\
\implies \lambda^+ =&\frac{\sum_{n=1}^{N} x_n}{N}
\end{aligned}
$$

Therefor, for this particular problem, the best estimator will be the average of all the observation, which is just:

$$
\frac{2+4+6+1}{5} = 2.6
$$

And that is the answer for the question.

(A.1.b)

The derivation of the best estimator in the general context is shown in A.1.a.

The numerical value for six observations is:

$$
\frac{2+4+6+1+3}{7} = 2.6666666666\cdots
$$

(A.1.c)

The numerical results for 5 observations has been shown in A.1.a and A.1.b respectively.

(A.2)

Objective: Find the MLE for the uniform distribution on $[o, \theta]$, where θ is the value we want to estimate.

Suppose that an observations has been made: $x_1, x_2, \cdots x_N$, and we assume that they are idd, and we want to find the likelihood of such an observation is generated using the Uniform distribution. And this will be given by the expression:

$$\prod_{n=1}^{N} \underbrace{\mathbb{P}\left(X = x_n\right)}_{\frac{1}{\theta}\mathbf{1}\{0 \le x_n \le \theta\}} \tag{A.2.1}$$

Observe that, if any of the observation is beyond the range $[0, \theta]$, we will have zero likelihood, so let's assume that $\theta \le \max_{1 \le i \le N}(x_i)$, then we will have this expression for the likelihood:

$$\prod_{n=1}^{N} \frac{1}{\theta} = \frac{1}{\theta^N} \tag{A.2.2}$$

And taking the log on that we have:

$$\log\left(\frac{1}{\theta^N}\right) = -N\log(\theta) \tag{A.2.3}$$

Observe that that function is monotonically decreasing as the value of θ get larger and larger, therefore, to maximize the likelihood, we need the value of θ to be as small as possible, and the smallest possible such θ that is not giving us zero likelihood is: $\max_{1 \le i \le N}(x_i)$, therefore, best estimate is given by:

$$\theta^+ = \max_{1 \le i \le N}(x_i)$$

Over-fitting

A.3

1. $S = \{(x_i, y_i)\}_{i=1}^N$ drawn from idd with underlying joint distribution $\mathcal{D}$.
2. The training set is break into $S_{\text{train}}, S_{\text{test}}$, and $S = S_{\text{Train}} \cup S_{\text{Test}}$, notice that overlapping is possible.
3. And the true least square error, given a predictive model f is: $\epsilon(f) = \mathbb{E}\left[(f(x)d - y)^2\right]$.

A.3.a

We want to show that that, the expected bias of the model f over the training set and the test set is the same as expected value of the bias over the distribution $\mathcal{D}$, hence it's unbiased.

Start by considering the expected value of $\epsilon_{\text{train}}(f)$:

$$\begin{aligned} & \mathbb{E}_{\text{train}}\left[\frac{1}{N_{\text{train}}}\sum_{(x,y)\in S_{\text{train}}}(f(x) - y)^2\right] \\ =& \frac{1}{N_{\text{train}}}\sum_{(x,y)\in S_{\text{train}}}\underbrace{\mathbb{E}_{\text{train}}\left[(f(x) - y)^2\right]}_{\epsilon(f)} \\ \hat{\epsilon}(f) =& \hat{\epsilon}_{\text{train}}(f) \end{aligned} \tag{A.3.a.1}$$

For each term $f(x) - y$, we assume that it's idd, and therefore, its expected value is going to be the same as if it's drawn from the distribution $\mathcal{D}$, because each sample of the train set is drawn in this way, therefore, we can conclude that $\mathbb{E}\left[(f(x)-y)^2\right]$ is going to give $\epsilon(f)$.

The proof for $\hat{\epsilon}_{\text{test}}(f)$ is going to be exact same because they are both drawn from the same idd joint distribution: $\mathcal{D}$. And hence we know that the bias estimator for any given model is going to be unbiased. Since both has the same variance we know that:

$$\mathbb{E}_{\text{train}}\left[\hat{\epsilon}_{\text{train}}(f)\right] = \mathbb{E}_{\text{test}}\left[\hat{\epsilon}_{\text{test}}(f)\right] = \hat{\epsilon}(f) \tag{A.3.a.2}$$

A.3.b

$\mathbb{E}_{\text{train}}\left[\hat{\epsilon}_{\text{train}}(\hat{f})\right] \neq \mathbb{E}\left[\epsilon(\hat{f})\right]$ Consider Linear regression where 2 the sample size is so small ($d-1$ samples where d is the number of features) that we have a perfect fit model $\hat{f}$ then the bias of the model is going to be zero under the training set, however, the bias over the real distribution is not, because $\mathcal{D}$ could be non-linear and there could be noises, which both will introduce some biased.

A.3.c

Some explanations are needed for this question. Consider the following expression:

$$\mathbb{E}_{\text{train}}\left[\hat{\epsilon}_{\text{train}}(\hat{f}_{\text{train}})\right] \le \mathbb{E}_{\text{train, test}}\;\left[\hat{\epsilon}_{\text{test}}(\hat{f}_{\text{train}})\right] \tag{A.3.c.1}$$

This is saying that, the expected value of the error on trained estimator evaluated using the test samples by enumerating over all the test samples is less than The expected value of the error when we train the model with train set and get the error with the test set. The estimator of the model is obtained via the following procedures:

Let $\mathcal{F} = \{f_1, f_2 \cdots\}$ be a collection of functions and $\hat{f}$ will minimizes the error given the training set. Now let's consider the error the expected value of the error on the test set of the estimator obtained from the train set:

$$\begin{aligned}
\mathbb{E}_{\text{train, test}}\;\left[\hat{\epsilon}_{\text{test}}(\hat{f}_{\text{train}})\right]\beta &= \sum_{f\in\mathcal{F}} \mathbb{E}_{\text{train, test}}\left[\hat{\epsilon}_{\text{test}}(f)\mathbf{1}\{\hat{f}_{\text{train}} = f\}\right] \\
&\underset{(1)}{=} \sum_{f\in\mathcal{F}} \mathbb{E}_{\text{test}}\left[\hat{\epsilon}(f)\right]\mathbb{E}_{\text{train}}\left[\epsilon_{\text{test}}\mathbf{1}\{\hat{f}_{\text{train}} = f\}\right] \\
&= \sum_{f\in\mathcal{F}} \mathbb{E}_{\text{test}}\left[\hat{\epsilon}_{\text{test}}(f)\right]\mathbb{P}\left(\hat{f}_{\text{train}} = f\right) \\
&= \mathbb{E}_{\text{test}}\left[\hat{\epsilon}_{\text{test}}(f)\right] \\
&\underset{(2)}{=} \mathbb{E}_{\text{train}}\left[\hat{\epsilon}_{\text{train}}(f)\right]
\end{aligned} \tag{A.3.c.2}$$

(1) This is by the independence between the test and the training set, therefore, 2 random variable based on each one them can have the expected value operator distributes over. (2) this is by results from A.3.a.2. Now, by the definition that $\hat{f}$ minimizes the error compare to all other models on the training setm then by definition, the expected value of it is going to keep the inequality, therefore:

$$\begin{aligned}
&\hat{f} = \min_{f\in\mathcal{F}} \hat{\epsilon}_{\text{train}}(f) \\
\implies &\mathbb{E}_{\text{train}}\left[\hat{\epsilon}_{\text{train}}(\hat{f}_{\text{train}})\right] \le \mathbb{E}_{\text{train}}\left[\hat{\epsilon}_{\text{train}}(f)\right] = \mathbb{E}_{\text{train, test}}\;\left[\hat{\epsilon}_{\text{test}}(\hat{f}_{\text{train}})\right]
\end{aligned} \tag{A.3.c.1}$$

And this is what we want to show for this problem.

Polynomial Regression

A.4

This part is too simple. See A.5.1

A.5

Code has been implemented. Here is the procedures for fitting the models:

1. Augmenting the features by raising the features to it's polynomials powers. Here we assume that the feature sare not interaction. Stack the initial feature columns horizontally to produce the Vandermonde matrix for the features.
2. Standardizing the data across the features (the columns) by it's standard deviation. This is just deviding each column by the sd of that column
3. Adding an additional rows of ones.
4. Creating the regularizer as a diagonal matrix, but the bottom right corner of the diagonal matrix is seet to zeero so we don't regularize on the offset for the regression model.

```
'''
    Template for polynomial regression
    AUTHOR Eric Eaton, Xiaoxiang Hu
'''

import numpy as np


#-----------------------------------------------------------------
#  Class PolynomialRegression
#-----------------------------------------------------------------

class PolynomialRegression:

    def __init__(this, degree=1, reg_lambda=1E-8):
        """
        Constructor
        """
        this.Degree = degree
        this.Lambda = reg_lambda


    def polyfeatures(self, X, degree):
        """
        Expands the given X into an n * d array of polynomial features of
            degree d.

        Returns:
            A n-by-d numpy array, with each row comprising of
            X, X * X, X ** 3, ... up to the dth power of X.
            Note that the returned matrix will not include the zero-th power.

        Arguments:
            X is an n-by-1 column numpy array
            degree is a positive integer
        """
        assert len(X.shape) == 2 and X.shape[1] == 1, "Wrong␣input␣shape␣for␣X␣for␣polyfeatures."
        return np.hstack([X**II for II in range(1, degree + 1)])


    def fit(this, X, y):
        """
            Trains the model
            Arguments:
                X is a n-by-1 array
                y is an n-by-1 array
            Returns:
                No return value
            Note:
                You need to apply polynomial expansion and scaling
                at first
        """

        X = this.polyfeatures(X, this.Degree)
        this.FeatureSTD = np.std(X, axis=0, keepdims=True)
        X = X / this.FeatureSTD
        XAug = np.zeros((X.shape[0], X.shape[1] + 1))
        XAug[:, :-1] = X
        XAug[:, -1] = np.ones(X.shape[0])
        X = XAug
        Regularizer = np.eye(X.shape[1])*this.Lambda
        Regularizer = this.Lambda*np.diag(np.ones(X.shape[1]))
        Regularizer[-1, -1] = 0
        this.ModelCoefficients = np.linalg.pinv(X.T@X + Regularizer)@X.T@y


    def predict(this, X):
        """
        Use the trained model to predict values for each instance in X
        Arguments:
            X is a n-by-1 numpy array
        Returns:
            an n-by-1 numpy array of the predictions
        """
        # Standardize input features using training data.
        X = this.polyfeatures(X, this.Degree)
        X = X / this.FeatureSTD
        XAug = np.zeros((X.shape[0], X.shape[1] + 1))
        XAug[:, :-1] = X
        XAug[:, -1] = np.ones(X.shape[0])
        PredictedY = XAug@this.ModelCoefficients
        return PredictedY



#-----------------------------------------------------------------
#  End of Class PolynomialRegression
#-----------------------------------------------------------------



def learningCurve(Xtrain, Ytrain, Xtest, Ytest, reg_lambda, degree):
    """
    Compute learning curve

    Arguments:
        Xtrain -- Training X, n-by-1 matrix
        Ytrain -- Training y, n-by-1 matrix
        Xtest -- Testing X, m-by-1 matrix
        Ytest -- Testing Y, m-by-1 matrix
        regLambda -- regularization factor
        degree -- polynomial degree

    Returns:
        errorTrain -- errorTrain[i] is the training accuracy using
        model trained by Xtrain[0:(i+1)]
        errorTest -- errorTrain[i] is the testing accuracy using
        model trained by Xtrain[0:(i+1)]

    Note:
        errorTrain[0:1] and errorTest[0:1] won’t actually matter, since we start displaying the
            learning curve at n = 2 (or higher)
    """

    n = len(Xtrain)
    errorTrain = np.zeros(n)
    errorTest = np.zeros(n)
    for II in range(2, n):
        TrainSetFeatures, TrainSetLabels = Xtrain[: II+ 1], Ytrain[:II + 1]
        Model = PolynomialRegression(degree=degree, reg_lambda=reg_lambda)
        Model.fit(TrainSetFeatures, TrainSetLabels)

        TrainPredicted = Model.predict(TrainSetFeatures)
        errorTrain[II] = np.mean((TrainPredicted - TrainSetLabels)**2)

        TestPredicted = Model.predict(Xtest)
        errorTest[II] = np.mean((Ytest- TestPredicted)**2)

    return errorTrain, errorTest
```

Ridge Regression on MNIST

A.6

For this problem we are looking at a linear model $W \in \mathbb{R}^{n\times d}$, and training labels $y_i \in \{0,1\}^{10}$. And the out put labels are just taking the index of the maximal element in the predicted vector. Note: $d = 784$, representing a all the features, each pixel of the flatten 28×28 image in the MNIST data set.
The model we want to train is a weight matrix $W = [w_i, w_2 \cdots w_d]$, and to predict the value for a given new data, let the estimator model to be $\hat{W}$, it's use like:

$$\underset{0\leq i\leq 10}{\operatorname{argmin}} \left\{ e_{j+1}^T \widehat{W}^T x_i \right\}$$

We take the output from the transposed weight matrix and look for the index of the entry with the maximal value to predict the lable digits. In addition, we also pack all the label vector vertically into a Matix $Y = [y_1, y_2, \cdots y_n]^T$, which will be $n \times 10$, and the data matrix $X = [x_1, x_2 \cdots x_n]^T$, to train the model $\widehat{W}$. We will pick up from the computation that is given from the HW sheet and continue:

$$\begin{aligned}
&\sum_{j=1}^{k} \left(\|Xw_j - Ye_j\|_2^2 + \lambda \|w_j\|_2^2 \right) \qquad \text{(A.6.1)} \\
=&\sum_{j=1}^{k} \left(\|Xw_j - (Y)_{:,j}\|_2^2 \right) \lambda \sum_{j=1}^{k} \|w_j\|_2^2 \\
=&\sum_{j=1}^{k} \left(\|(Xw)_{:,j} - (Y)_{:,j}\|_2^2 \right) + \lambda \|W\|_F^2 \\
=&\|XW - Y\|_2^2 + \lambda \|W\|_F^2
\end{aligned}$$

Note that, next we are taking the derivative of this scalar error function wrt to the weight matrix W, which should produce the gradient of the above error function. And the gradient will be a matrix of size $d \times k$, and the solution to gradient zero will be the model that minizes the square loss of the ridge regression.
We will also make use of the rule that $\nabla[g(Ax)] = A^T \nabla[g](Ax)$ where $g(x)$ is a $\mathbb{R}^m \mapsto \mathbb{R}^n$, and the matrix $A \in \mathbb{R}^{m\times n}$. I learn this technique in math 515 and I am just going to use this here. In addition, take the Frobenius Norm is easy because it's a giant sum of all the entries squared in the matrix, and each weight of the matrix is independent from eaco other, so $\partial_{w_{ij}}[||W||_F^2] = 2w_{ij}$. Using these fact we can take the gradient of the above loss function and get:

$$\begin{aligned}
\nabla_W \left[\|XW - Y\|_2^2 + \|W\|_F^2 \right] &= 0 \qquad \text{(A.6.2)} \\
2X^T(XW - Y) + 2\lambda W &= \mathbf{0} \\
X^T XW - X^T Y + \lambda W &= 0 \\
(X^T X + \lambda I)W &= X^T Y \\
W &= (X^T X + \lambda I)^{-1} X^T Y
\end{aligned}$$

And this is the solution for the optimal model that set the gradient of the ridge regression to zero gradient. And notice that in practice, due to sparcity of the matrix X, it's better to use Penrose Psuedo Inverse when the size of the regularizer is small. Also I am not sure whether to standardize the data for each features to zero mean and unit variance, I would assume not.

B.2