Assignment 2 Report

20180396 오지오

A.

```
def _fit(self, X, t):
    """
    estimate parameter given training dataset

    Parameters
    ----------
    X : (N, D) np.ndarray
        training dataset independent variable
    t : (N,) np.ndarray
        training dataset dependent variable
        binary 0 or 1
    """
    X0 = X[t == 0]
    X1 = X[t == 1]
    m0 = np.average(X0,axis=0)
    m1 = np.average(X1,axis=0)
    s0 = np.zeros((np.size(m0,0), np.size(m0,0)))
    s1 = np.zeros((np.size(m1,0), np.size(m1,0)))
    for i in range(X0.shape[0]):
        k = np.array((X0[i]-m0))[np.newaxis]
        s0 += k.T@k
    for i in range(X1.shape[0]):
        k = np.array((X1[i]-m1))[np.newaxis]
        s1 += k.T@k
    cov_inclass = s0 +s1
    self.w = np.linalg.inv(cov_inclass)@(m1-m0)


    self.w /= np.linalg.norm(self.w).clip(min=1e-10)
    self._threshold(X, t)
```

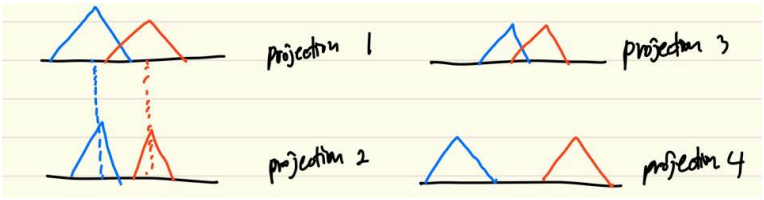
B.

```
def _fit(self, X, t, max_iter=100):
    """
    maximum likelihood estimation of logistic regression model

    Parameters
    ----------
    X : (N, D) np.ndarray
        training data independent variable
    t : (N,) np.ndarray
        training data dependent variable
        binary 0 or 1
    max_iter : int, optional
        maximum number of paramter update iteration (the default is 100)
    """
    self.w = np.zeros(np.size(X, 1))
    w = self.w
    for _ in range(max_iter):
        w_prev = np.copy(w)
        y_hat = self.proba(X)

        grad = X.T @ (y_hat-t)

        w = gradient_descent(w, grad, learning_rate=0.1)
        if np.allclose(w, w_prev):
            break
```

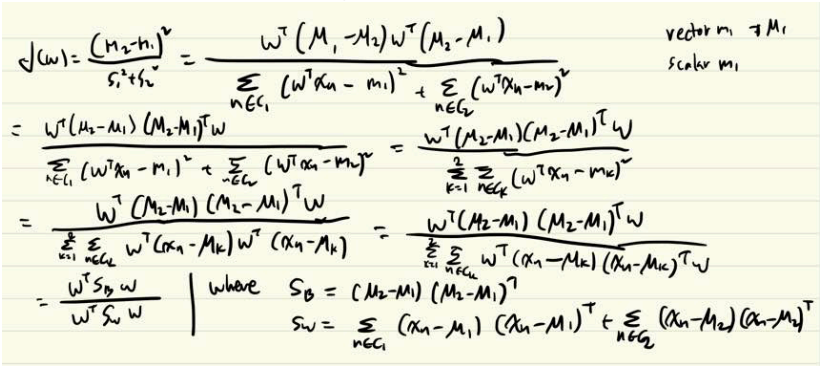
C.

Linear discriminant analysis is generalization of the Fisher's linear discriminant, where we try to find a linear combination of features that separates two or more classes. This combination is used as a linear classifier. In particular, Fisher's linear discriminant also tries to find the coefficient for the linear combination in order to classify two or more classes. In a two-class set, intuitively, we try to find a line in a d-dimension space so that the two classes are separated, so that we could minimize the error of classification. We tend to find a vector in a canonical direction so that when we do the projection of all data to that vector, the least number of points from two other classes overlap. In order to achieve this, we should try to find the vector that makes the two classes have the furthest center of mass, in other words, mean and each class has the smallest variance, so that

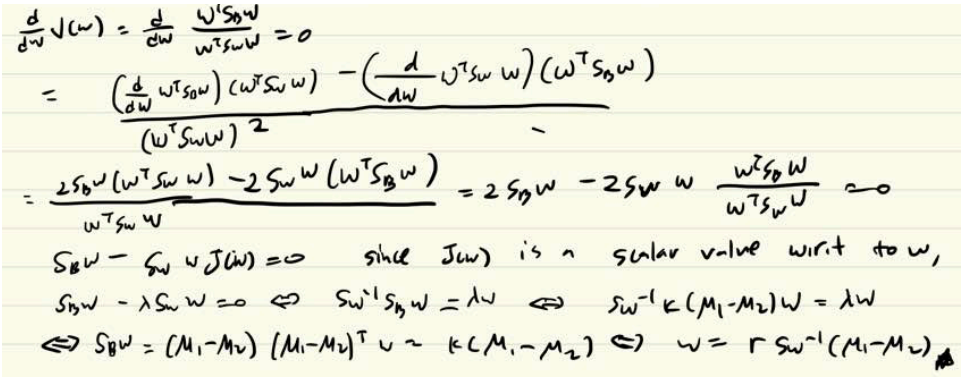
all data in each data are clustered. We can clearly see below that projection 2 is better than projection 1 and projection 4 is better than projection 3.



Now, the derivation can be showed as the following. If we assume to have two classes C_1 and C_2 and put the input vector with d dimension as x and the weight vector as w, our prediction can be denoted as y_hat = $w^T x$. For each classes the center of mass can be computed by the average of each classes as $\boldsymbol{m_1} = 1/N_1 \sum_{n\in C_1} x_n$, $\boldsymbol{m_2} = 1/N_2 \sum_{n\in C_2} x_n$. Here we make a projection of these center of mass, $m_1 = w^T \boldsymbol{m_1}, m_2 = w^T \boldsymbol{m_2}$. We now try to measure the variance within each classes, as $s_1^2 = \sum_{n\in C_1}(w^T x - m_1)^2$, $s_2^2 = \sum_{n\in C_2}(w^T x - m_2)^2$. As mentioned above we would like to maximize the distance between each center of mass and minimize the variances, so we could write down the criterion as $J(w) = (m_2 - m_1)^2/(s_2^2 + s_1^2)$. This could be formulated as the matrix form as the following.



We want to maximize this J(w) and in order to do that we differentiate this equation by w set it to zero in order to find the local maximum.



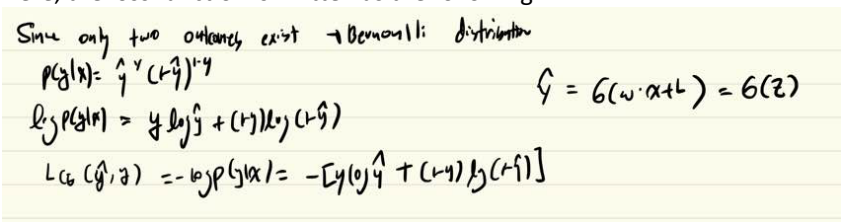
Here, r is a constant and we do not really care about the actual value of w, so in the code I set it to 1.

D.

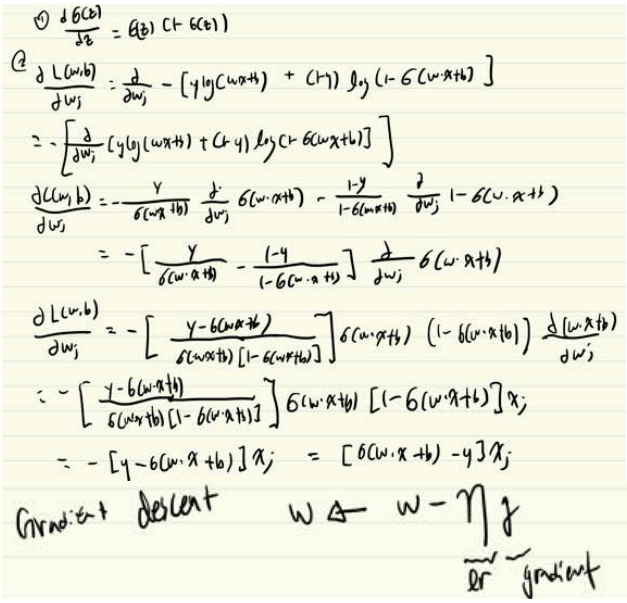
A discriminative classifier tries to learn how to distinguish the classes and they do not care about the general characteristics of each class. They try to compute the $P(C_1|x)$ directly. There are four basic components in a classification system, a feature representation of the input, the classification function that determines the y_hat, objective function, and the algorithm to optimize the objective function. In logistic regression in this project, we try to train the system using the SGD (stochastic gradient descent) and the BCE and test is by computing p(C|x) and return the higher probability label, for instance if it is bigger than .5, y is 1 and 0 if not. We have a goal to train a classifier that can make a binary decision about the class of a new input.

So, in order to perform logistic regression, we try to find a score function, which is consisted of weights and bias, that gives us a score when we put the input features x to the function. ($z = b + w^T x$) Using this score, we tend to calculate the probability of each cases, we'll pass z through the sigmoid function. ($\sigma(z) = \frac{1}{1+e^{-z}}$) This indicates the probability of each type to be in class 1 and if we subtract this value from 1, that will indicate the probability

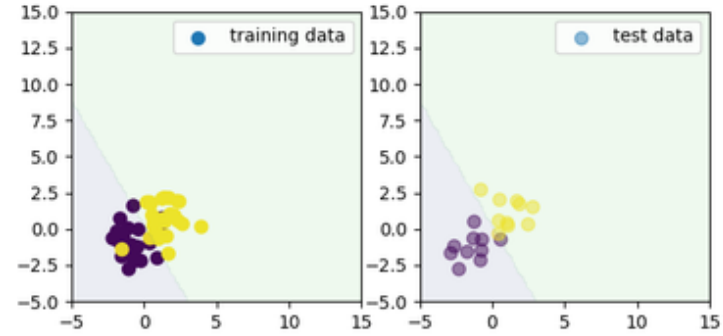
of that data to be in class 0. This is because the sigmoid function itself calculates the probability of each data being in one of the classes, which I denoted as class 1 above.
Here, the loss function is written as the following.



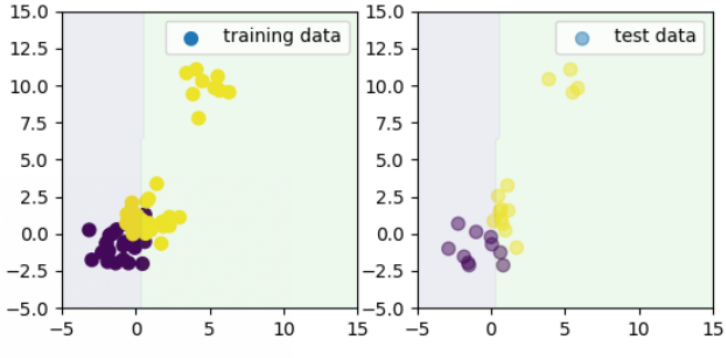
Using, this loss function we try to train our model using the concept of stochastic gradient descent. Here, we compute the predicted y_hat and compute the loss for each prediction. Then, compute the gradient of the loss function with respect to the weights of the score function, which tells us how to move the weight to maximize the loss. Then, we go the other way, in other words, we subtract the product of the hyperparameter (learning rate) and the gradient from the original weight and get a new weight. We iterate this over and over to optimize the model. The following indicates the process and derivation of gradients. This only shows the gradient of one weight and datum, in our code division via the size of the data is added considering the deviation terms in the actual loss function.



E.
For the dataset with no outliers, i.e. dataset 0~4, the average BCE was 1.9342 and its standard error was 0.8219. On the contrary, for the dataset with outliers, i.e. dataset 5~9, the average BCE was 2.8207 and its standard error was 1.3023. As we can see in the numerical values of BCE and standard error, the LDA method is quite sensitive to outliers. Plus, we can confirm this in the result plots. For dataset without any outliers the LDA method, classification seems to work well with the classification boundary having a negative slope with relatively small absolute value. However, due to the introduction of outliers (for dataset 5~9), the classification boundary's slope has a very high absolute value, and is definitely not classifying the two classes precisely. The boundary is somewhat distorted due to these outliers. Thus, we can derive the solution that the LDA method is highly sensitive to outliers.



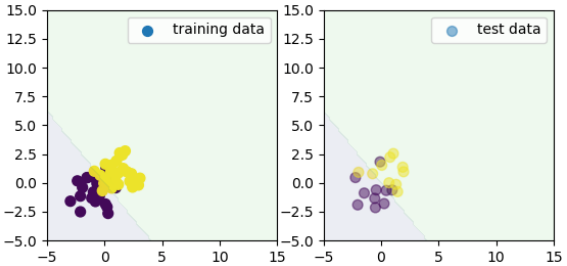
LDA, dataset 4



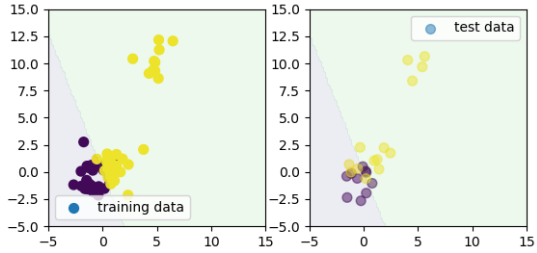
LDA, dataset 9

F.

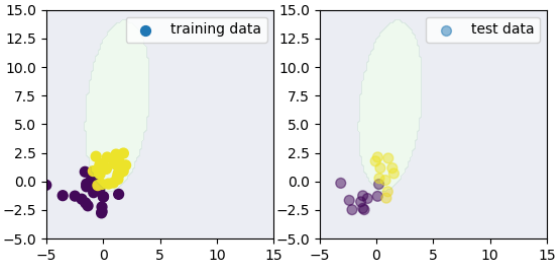
If we fix feature as PolynomialFeatures(2), in other words enable feature engineering with dimension 2, the average BCE for dataset without any outliers (dataset 0~4) is 2.2565 and the standard error is 0.6031. For dataset with outliers (dataset 5~9) the average BCE is (dataset 5~9) is 1.7461 and the standard error is 1.5077. If we turn off the feature engineering the average BCE for dataset without outliers is 1.9342 and the standard error is 0.8219. For dataset with outliers, the average BCE is 1.4775 and the standard error is 2.0103. Also, for other odd number features, the average BCE gap between dataset with and without outliers wasn't that big. Plus, as we can see the pictures below, the classifier itself seems to distinguish two classes very well regardless of the presence of outliers. It seems as the outliers make the performance slightly worse. However, the model is relatively much less sensitive and I think we might conclude that the model itself is robust to outliers. (Though, the average BCE got higher for higher *even* dimensions for dataset with outliers, this seems to be a result of another problem rather than sensitivity. It seems that it is hard to say that the logistic regression model with even number features are highly sensitive. We should rather say that it is troublesome. This will be explained in detail in the next part.)

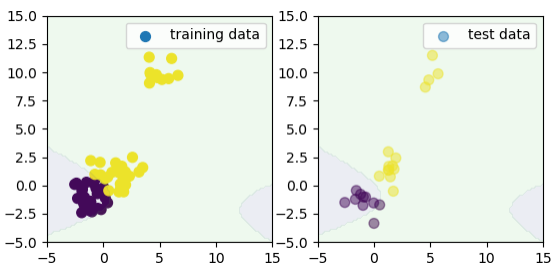


Logistic Regression, no f.e., dataset 0

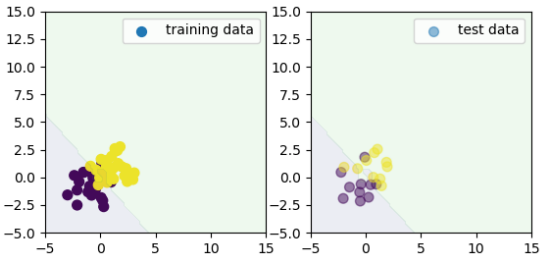


Logistic Regression, no f.e., dataset 8

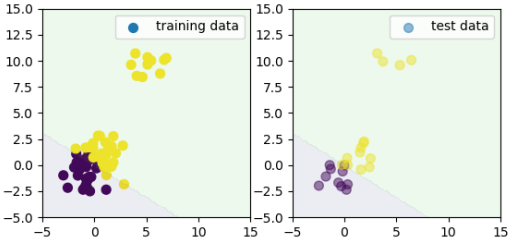




Logistic Regression, f.e.=2, dataset 2



Logistic Regression, f.e.=2, dataset 7



Logistic Regression, f.e.=7, dataset 0

Logistic Regression, f.e.=7, dataset 6

	Avg BCE	Std BCE	Avg BCE	Std BCE	Avg BCE	Std BCE
None	1.9342	0.8219	1.4775	2.0103	1.7058	1.5526
1	1.9342	0.8219	1.3432	1.7513	1.6387	1.3995
2	2.2565	0.6031	1.7461	1.5077	2.0013	1.1763
3	2.5789	1.0692	1.0745	1.2456	1.8267	1.3831
4	3.2236	1.9071	4.4325	2.0192	3.828	2.0549
5	2.7401	1.3094	1.2089	1.4341	1.9745	1.5722
6	2.9013	1.6595	5.6413	1.7306	4.2713	2.1798
7	2.2565	0.9398	0.9402	0.911	1.5984	1.1357
8	3.7072	1.736	6.0443	0.9498	4.8757	1.823
9	2.2565	0.9398	1.2089	1.1554	1.7327	1.1763
10	3.7072	1.9476	6.3129	0.5373	5.01	1.9335
11	1.773	0.9398	1.0745	1.2456	1.4238	1.1573
13	1.6118	1.1397	1.0745	1.2456	1.3432	1.2237
15	1.4506	0.9398	1.0745	1.2456	1.2626	1.1193
17	1.6118	1.1397	1.0745	1.2456	1.3432	1.2237
19	1.6118	1.1397	1.0745	1.2456	1.3432	1.2237
	Dataset 0~4		Dataset 5~9		Dataset 0~9	

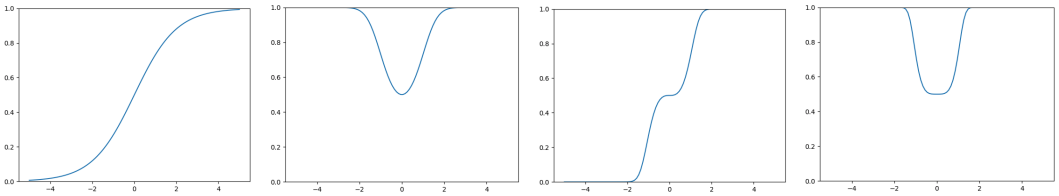
BCE data(BCE compare.xlsx)

G.

It is a bit difficult to pinpoint the best model that explains this dataset. Before I start with the selection of the model, I would like to start some points with the observation of the model.

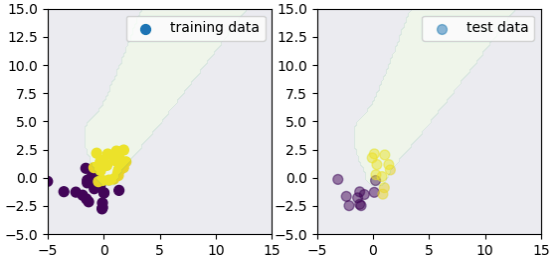
The sigmoid function that we implemented here looks like the following. $\sigma(z) = \frac{1}{1+e^{-z}}$

Now, let's see the sigmoid function as we increase the dimension of z as z^2, z^3, … (saved in results/sigmoid)

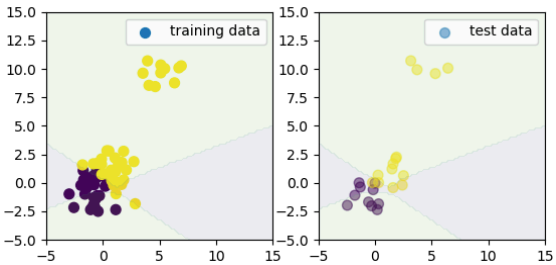


z z^2 z^3 z^4

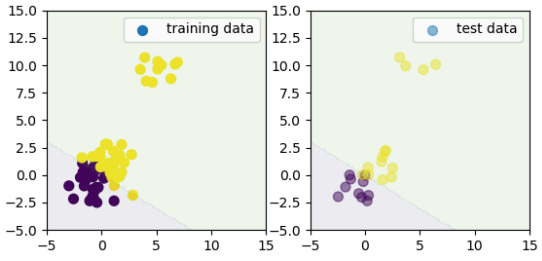
With this information on our mindset, if we analyze the numerical results (BCE) that we've gotten from the model, we can see that for even features, the average BCE is increasing as feature gets bigger, while for odd features, the average BCE repeats to increase and decrease by a very small amount. Plus, if we see the plot above in the answer F, for odd number features, no matter how big the feature is, the classification boundary seems to be linear. On the other hand, for even number features, the classification boundary seems somewhat nonlinear, but inaccurate especially for data with outliers. Why does this happen?



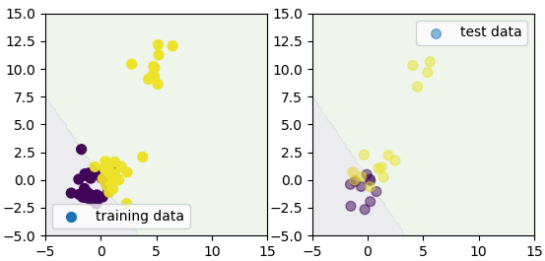
Logistic Regression, f.e.=4, dataset 2



Logistic Regression, f.e.=4, dataset 6

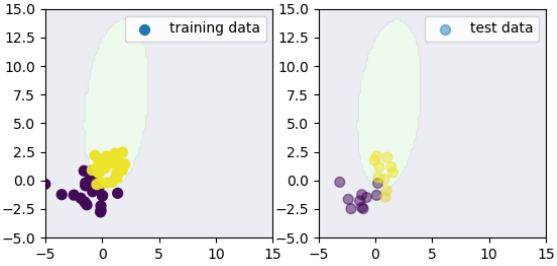


Logistic Regression, f.e.=11, dataset 6

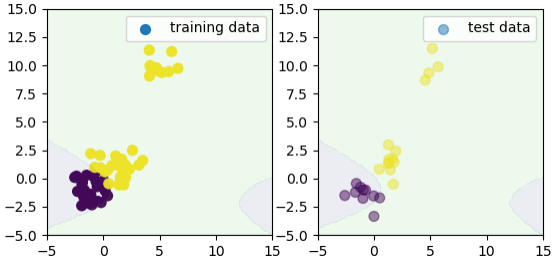


Logistic Regression, f.e.=3, dataset 8

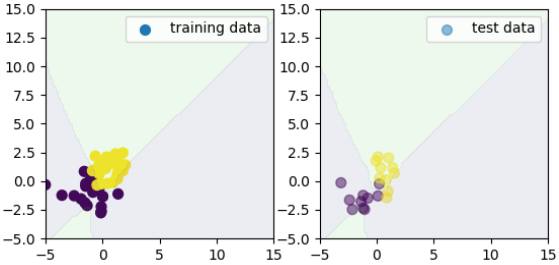
Let's start with even features. As mentioned above, the sigmoid with even dimensional features tend to have a plot, which has the value extremely close to 1, when the input (score) goes to negative infinity. Plus, the lower limit of the function itself tends to be quite high. So, the score function itself mostly evaluates the input as class 1. Though weights can control this fatality, there are limitations. For smaller even number dimensions (i.e. 2), the effects of odd number features still modify the classification boundary to be quite accurate. However, as the dimension gets higher and higher the score function classifies more and more inputs to a particular class, leading to even bigger errors. We can also see this visually since as the feature dimension gets higher and higher, the classification boundary doesn't really work well. This even gets worse for data with outliers since the modification of weights cannot prevent the misclassification well enough as outliers tend to distort the model. Therefore, when selecting the optimal model for datasets, I tried to exclude the model with even number features.



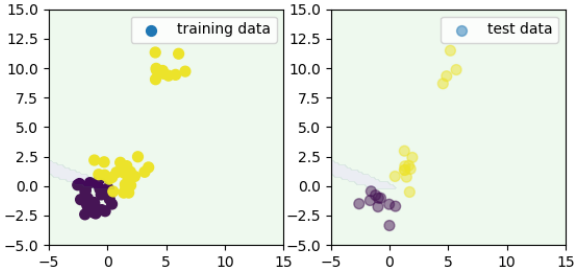
Logistic Regression, f.e.=2, dataset 2



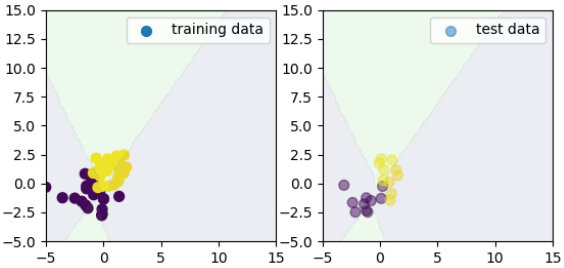
Logistic Regression, f.e.=2, dataset 7



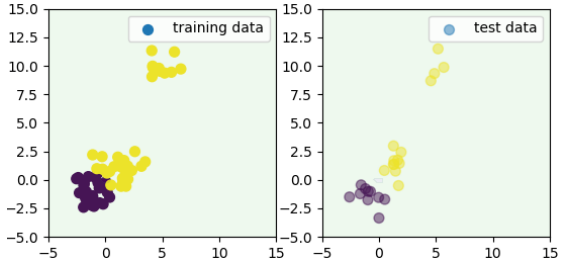
Logistic Regression, f.e.=6, dataset 2



Logistic Regression, f.e.=6, dataset 7



Logistic Regression, f.e.=10, dataset 2



Logistic Regression, f.e.=10, dataset 7

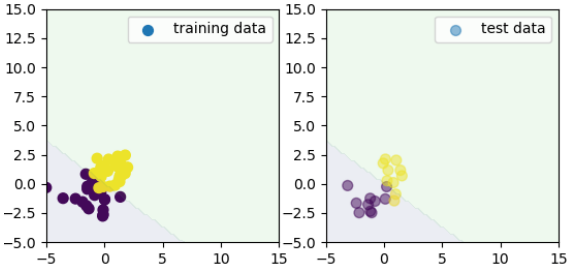
Next, for the odd features, no matter how big the feature is, the classification boundary seems to be linear. I believe that this happens due to the shape of the sigmoid function. The sigmoid function of odd dimensional features has a shape that ranges from 0 to 1, having a point symmetry at the middle. Similar to the original sigmoid function that we know, for negative infinity input the output goes to 0 and for positive infinity input the output goes to 1. The only difference is that the functions get steeper as the dimension increases. So the BCE average will not differ that much compared to the ones with the original sigmoid function (with no feature engineering). If we see the graph of the sigmoid function once again saved in "./Results/sigmoid", we see that

the steep structure distinguishes data near the point of symmetry very well, by returning huge difference in outputs, for small input differences aren't that big. Hence, we can say that the model will get more sensitive to the training data.

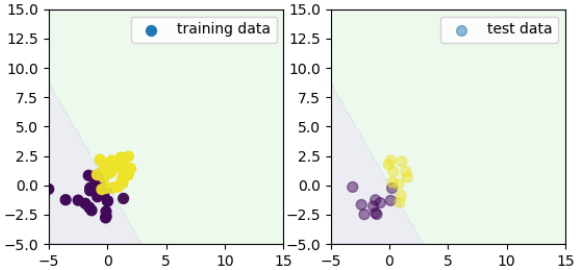
Keeping all of the information in mind, let's try to find the best model for these datasets. There are two things to point out here. First, as dimensions increase, the average BCE and the standard error rate seems to be constant. This seems to happen because as more and more features are added, new features have smaller and smaller influences on the output. This effect is even enlarged since we multiply the small learning weight and the weights when we actually deal with the gradient descent. (convergence) Thus, at some point the BCE score values remain unchanged. Second, as mentioned above, we exclude the model with even number dimensional features.

(1) Dataset without outliers

If we have preprocessed the data to have no outliers, in other words, the dataset is guaranteed to have no outliers. We have two models that come up to our mind. The one with no features (Avg BCE: 1.9342, std: 0.8219) and the model with feature of 15 dimensions (Avg BCE: 1.4506, std: 0.9398). Though the one with feature dimension 15 has a slightly higher standard error rate, the average BCE differs a lot by approximately 0.5, so it seems reasonable to choose the one with feature dimension 15 over the one with no feature engineering.



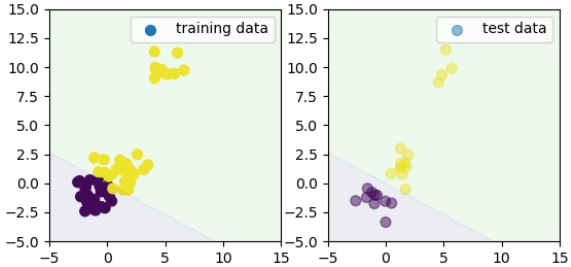
Logistic Regression, f.e.= None, dataset 2



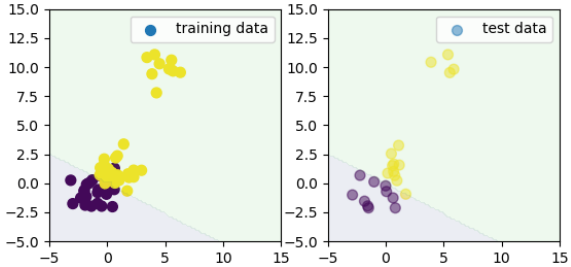
Logistic Regression, f.e. =15, dataset 2

(2) Dataset with outliers

Let's say somehow, we are confident that outliers will exist in the data. Here, the model with feature dimension 7 (Avg BCE: 0.9402, std: 0.911) seems to be the most appropriate since it has both the lowest average BCE and the standard error rate.



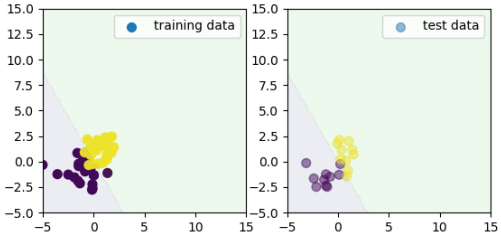
Logistic Regression, f.e.= 7, dataset 7



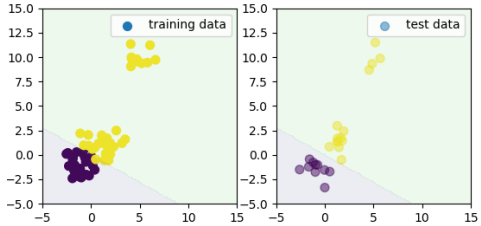
Logistic Regression, f.e. =7, dataset 9

(3) Unknown dataset

Lastly, let's say some arbitrary dataset will come for the next dataset and we would like to use the classification algorithm with logistic regression. We don't have any clue whether they have outliers or not. I think it might be a good idea to use the result from dataset 0~9, which includes both data with and without outliers, since it gives us an average performance metric. The model with feature dimension 15 (avg BCE: 1.2626. std BCE: 1.1193) seems to have the best BCE score among all the dimensions. Though they underperform slightly than the model with feature dimension 7, its performance on datasets without outliers outweigh a lot than the comparison target. Therefore, a more stable model should be chosen, since we don't know which data will arrive for testing.



Logistic Regression, f.e. =15, dataset 2



Logistic Regression, f.e. =15, dataset 7

(Some additional Points)
This result was a little shocking for me, since I thought that the higher the dimensions, overfitting will happen and the performance will get worsen by a big margin as we've seen in linear regression. After a long time of consideration, I got a conclusion of my own, that this happened due to the special characteristic that we have. As we increase the number of feature dimensions, the smaller dimensional features still exist in the equation, and the sigmoid function itself, doesn't really react sensitively as higher dimensional features are introduced. Thus, it fits slightly more to the training data, but doesn't really change a lot (also due to the SGD method that changes the weights by a smaller value). Overfitting problems are in some way removed in these parts.
If we increase the learning rate a lot, this might change, but large learning rates might trigger oscillations or diversions.