Project Report 1
Empirical Research on Different Learning Models

Tingyuan LIANG

March 12, 2018

Date Performed:	March 2, 2018
Instructor:	Professor Dit-Yan Yeung
TA:	Zhihan GAO

1 Objectives

To acquire a better understanding of classification methods by using a public-domain software package called scikit-learn.

To compare the performance of several classification methods by conducting empirical comparative study on five data sets.

2 Major Tasks

The project consists of the following tasks:
1. To learn to use a logistic regression (a.k.a. logistic discrimination) model for classification.
2. To learn to use a single-hidden-layer neural network model for classification.
3. To learn to use a support vector machine (SVM) model for classification.
4. To conduct empirical study to compare several classification methods.

3 Task 1: Logistic Regression

In this project, the logistic regression estimator is based on stochastic gradient descent (SGD) learning: the gradient of the loss is estimated each sample at a time and the model is updated along the way with a decreasing strength schedule (aka learning rate). SGD allows minibatch (online/out-of-core) learning, which has been involved in these project. [1]

3.1 The Experiment Settings of the Logistic Regression Model

The major parameter settings of SGD Classifier are presented in Table 1, while other parameters stay the same as default.

Names	Parameter Settings
Classifier	SGD
Loss Function	Logistic Regression
Learning_Rate	Invscaling
η_0	0.01
$power_t$	0.5
Batch Size	$min(200, N_{samples})$
Epoch	300

Table 1: Experiment Settings of the Logistic Regression Model

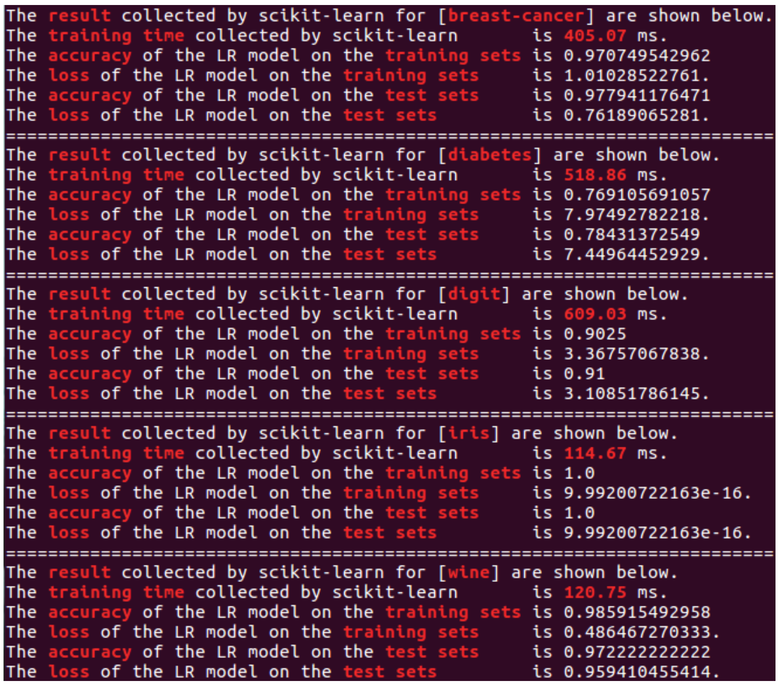
The loss function is set to be logistic regression for the sake of evaluating what we learn from class. The convergence speed of stochastic gradient descent is in fact limited by the noisy approximation of the true gradient. When the gains decrease too slowly, the variance of the parameter estimate weights in the network decreases equally slowly. When the gains decrease too quickly, the expectation of the parameter estimate weights takes a very long time to approach the optimum. The inverse scaling learning rate schedule can lower the rate more quickly than the exponential rates initially and then more slowly for later learning samples. [2] The hyperparameters of learning rate are tuned from the test to obtain high accuracy. This is a popular learning rate because it is guaranteed to converge in the limit. It can be slower to converge once it gets near a solution than exponential decay. The learning rate for logistic regression in this project can be represented as follow:

$$\eta_i = \frac{\eta_0}{i^{power_t}}$$

Moreover, SGD is sensitive to feature scaling, so the training data are scaled by using StandardScaler in order to increase the precision of logistic regression. [3] Note that scaling of features should not affect the result of logistic regression but regularization makes the SGD classifier dependent on the scale of the features. Note that the same scaling has been applied to the test vector to obtain meaningful results. The concrete implementation with scikit-learn is shown as below and the output of scikit-learn for logistic regression is presented in Figure 1.

```python
data = np.load("datasets/"+name+".npz")
X = data['train_X']        #load data for training
Y = data['train_Y']
X_test = data['test_X']    #load data for testing
Y_test = data['test_Y']
scaler = StandardScaler()
scaler.fit(X)  # Initialization of the scaler
X = scaler.transform(X)
X_test = scaler.transform(X_test)  # apply same transformation to test data
model = linear_model.SGDClassifier(loss='log',learning_rate='invscaling',eta0=0.01,\
                                    power_t=0.5)
model.fit(X, Y) #training
mini_batch_size = min(200,len(X))
epoch = 300
model.partial_fit(X[0:1], Y[0:1], classes=np.unique(Y))
for ep in range(epoch):  #repeat the training to increase accuract
    for i in range(0,len(X),mini_batch_size): # train the model with a mini-batch
        model.partial_fit(X[i:(i+mini_batch_size)], Y[i:(i+mini_batch_size)])
print("The accuracy of the MLP model on the training sets  is "\
         +str(sklearn.metrics.accuracy_score(Y, model.predict(X))))
print("The loss of the MLP model on the training sets is "\
        +str(sklearn.metrics.log_loss(Y, model.predict(X)))+".")
print("The accuracy of the MLP model on the test sets is "\
       +str(sklearn.metrics.accuracy_score(Y_test, model.predict(X_test))))
print("The loss of the MLP model on the test sets is "\
       +str(sklearn.metrics.log_loss(Y_test, model.predict(X_test)))+".")
```

: Code Outline for Logistic Regression with scikit-learn



```
The result collected by scikit-learn for [breast-cancer] are shown below.
The training time collected by scikit-learn       is 405.07 ms.
The accuracy of the LR model on the training sets is 0.970749542962
The loss of the LR model on the training sets     is 1.01028522761.
The accuracy of the LR model on the test sets     is 0.977941176471
The loss of the LR model on the test sets         is 0.76189065281.
=========================================================================
The result collected by scikit-learn for [diabetes] are shown below.
The training time collected by scikit-learn       is 518.86 ms.
The accuracy of the LR model on the training sets is 0.769105691057
The loss of the LR model on the training sets     is 7.97492782218.
The accuracy of the LR model on the test sets     is 0.78431372549
The loss of the LR model on the test sets         is 7.44964452929.
=========================================================================
The result collected by scikit-learn for [digit] are shown below.
The training time collected by scikit-learn       is 609.03 ms.
The accuracy of the LR model on the training sets is 0.9025
The loss of the LR model on the training sets     is 3.36757067838.
The accuracy of the LR model on the test sets     is 0.91
The loss of the LR model on the test sets         is 3.10851786145.
=========================================================================
The result collected by scikit-learn for [iris] are shown below.
The training time collected by scikit-learn       is 114.67 ms.
The accuracy of the LR model on the training sets is 1.0
The loss of the LR model on the training sets     is 9.99200722163e-16.
The accuracy of the LR model on the test sets     is 1.0
The loss of the LR model on the test sets         is 9.99200722163e-16.
=========================================================================
The result collected by scikit-learn for [wine] are shown below.
The training time collected by scikit-learn       is 120.75 ms.
The accuracy of the LR model on the training sets is 0.985915492958
The loss of the LR model on the training sets     is 0.486467270333.
The accuracy of the LR model on the test sets     is 0.972222222222
The loss of the LR model on the test sets         is 0.959410455414.
```

Figure 1: Output of scikit-learn for All the Datasets Based on Logistic Regression Model

3.2 The Experiment Reseult of the Logistic Regression Model

Based on scikit-learn, the accuracy results of different datasets, on both the test sets and training test, are obtained, as shown in Table 2. Figure 2 show the change in performance of the logistic regression model over time. It can be noticed that most of datasets result in high accuracy except diabet.

Dataset	Accuracy of Training	Accuracy of Test	Training Time
breast-cancer	0.971	0.978	405.07 ms
diabetes	0.769	0.784	518.86 ms
digit	0.903	0.910	609.03 ms
iris	1.000	1.000	114.67 ms
wine	0.986	0.972	120.75 ms

Table 2: Accuracy of the Logistic Regression for Datasets

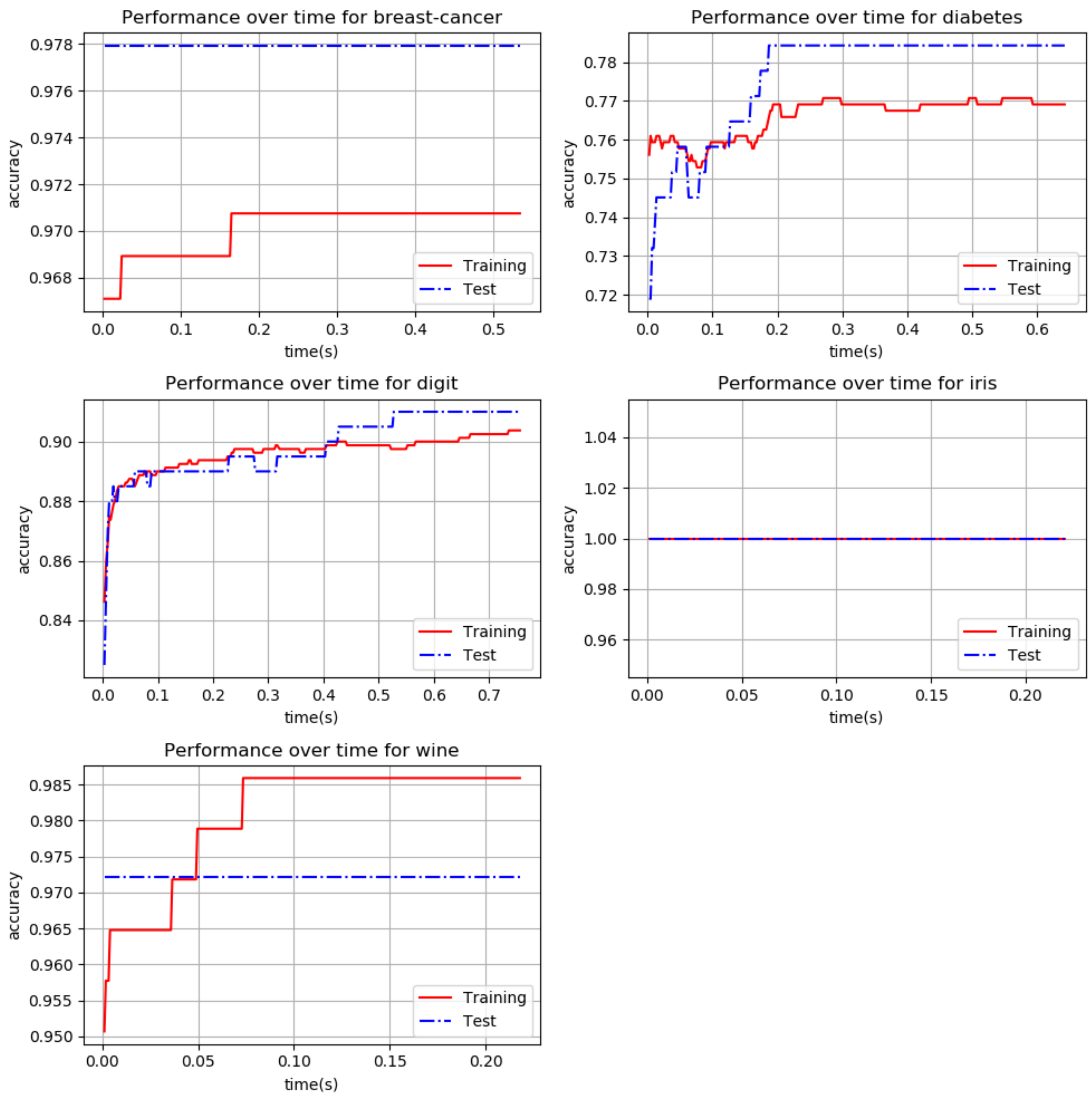


Figure 2: Performance of the Logistic Regression Model over Time

3.3 Further Study of the Logistic Regression Model

As mentioned in previous subsection, whether the data are scaled will impact the performance of the SGD classifier. Again, note that whether scaling data will not effect the logistic regression model. The result based on SGD classifier without data scaling is demonstrated in Figure 3. As shown in the figure, without scaling, SDG algorithm based on logistic regression lead to the low accuracy and low stability of the model for dataset wine, which are almost unacceptable. This point out the sensity of SGD to feature scaling. Moreover, the accuracy of diabet will be also undermined significantly if without data scaling.

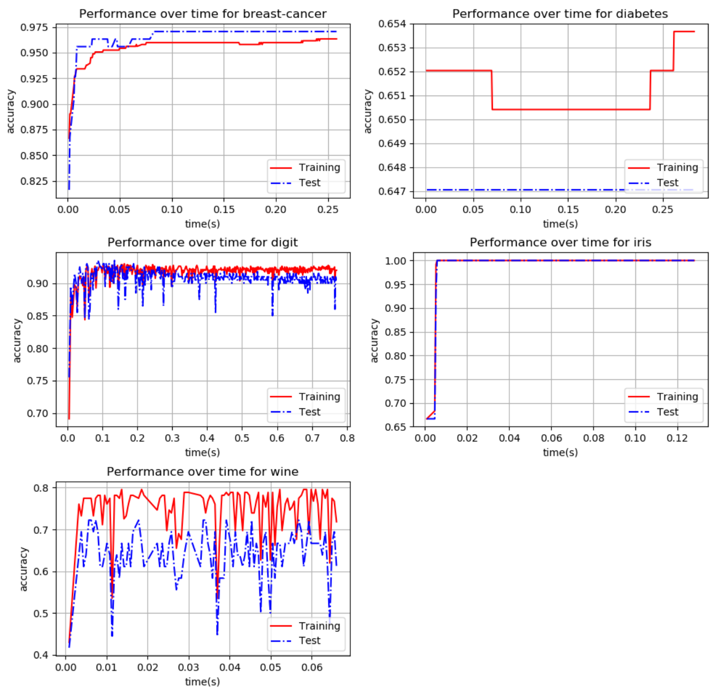
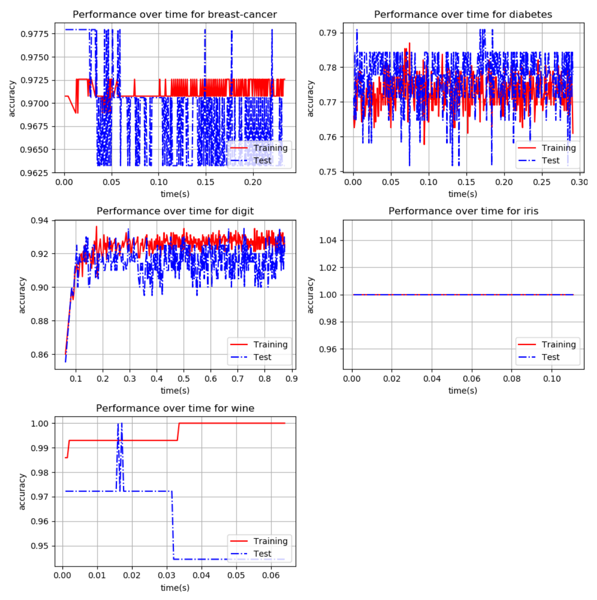
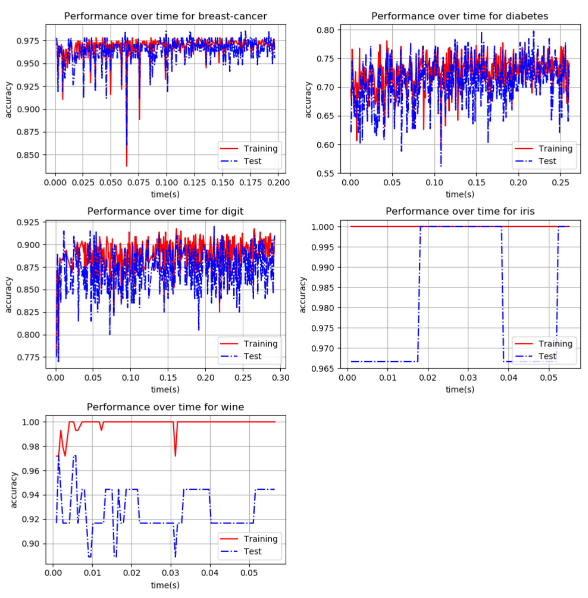


Figure 3: Logistic Regression Model without Data Scaling

Since the learning rate is also an important factor of SDG classifier, the comparsion among learning rate is also conducted during the implementation of project. Figure 2 shows the performance with inverse scaling learning rate, Figure 4(a) and Figure 4(b) are for constant learning rate and optimal learning rate (default by scikit-learn respectively. From these figures, it can be found that inverse scaling learning rate is better than other alternatives in the aspect of accuracy and result stability.



(a) Logistic Regression Model with Constant Learning Rate



(b) Logistic Regression Model with Optimal Learning Rate (default by scikit-learn)

Figure 4: Logistic Regression Model with Different Learning Rates

Moreover, the size of mini-batch will effect the performance of SGD classifier. Such impact have been evaluated via different datasets with different sizes of mini-batch, as shown in Figure 5 and Figure 6. The result demonstrates that as the batch size increases, the speed of processing samples is accelerated but more epoches will be required to obtain the same accuracy. Therefore, the selection of batch size involves some trade-off.

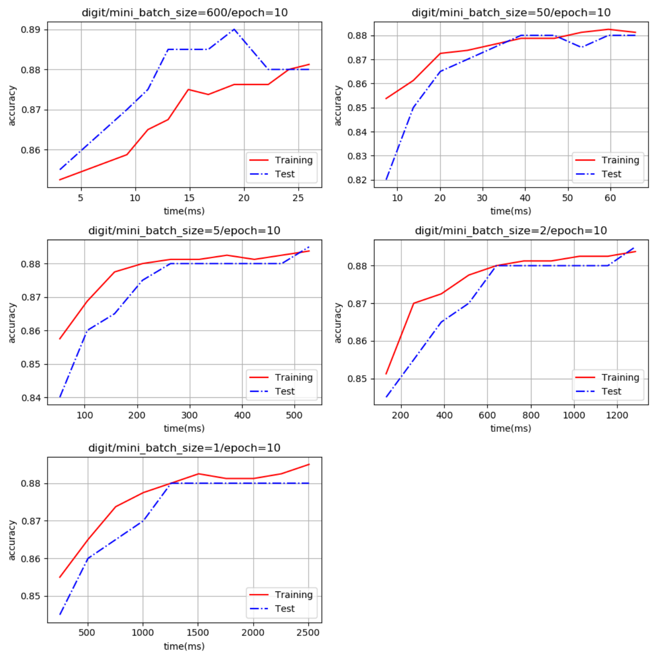


Figure 5: dataset Digit with different sizes of mini-batch

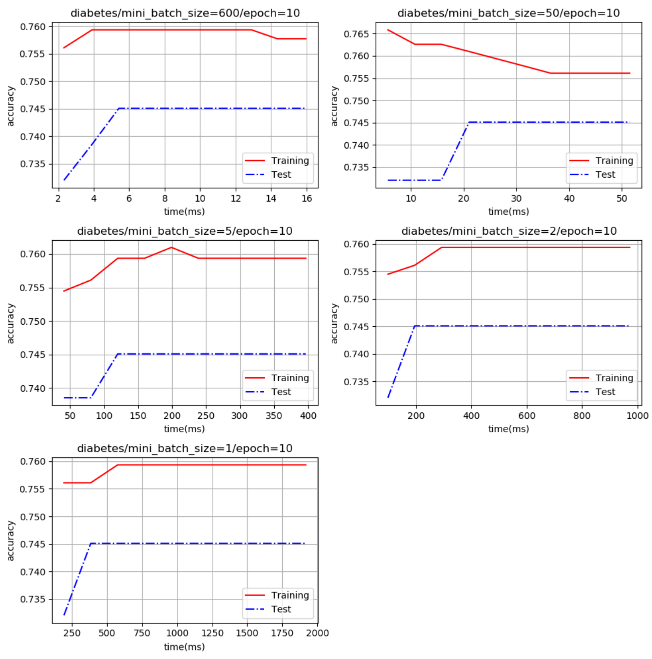


Figure 6: dataset Diabetes with different sizes of mini-batch

4 Task 2: Single-Hidden-Layer Neural Network

In this project, the model of single-hidden-layer nerual network is based on a multi-layer perceptron (MLP) algorithm that trains using backpropagation. It is different from logistic regression, in that between the input and the output layer, there is one non-linear layers, called hidden layer. Such learning process is implemented by MLPClassifier of scikit-learn.

4.1 The Experiment Settings of the Neural Network Model

The major parameter settings of MLP Classifier are presented in Table 3, while other parameters stay the same as default.

Names	Parameter Settings
Classifier	MLPClassifier
Activation Function	Logistic Sigmoid Function
Solver	Adam
Alpha	0.0001
Batch Size	$N_{samples}/20$
Epoch	150
Maximum of Iteration	10000
Number of Units in Hidden Layer	Will Be Tuned

Table 3: Experiment Settings of the Neural Network Model

In order to evaluate what we learn from class, the activation function is set to be the logistic sigmoid function. In this project, a stochastic gradient-based optimizer proposed by Kingma, Diederik, and Jimmy Ba, called Adam, is involved to solve weight optimzation. [4] Since Adam is an adaptive learning rate method, the learning rate η does not need to be specified in advance. Moreover, the number of hidden unit H is determined using cross validation. The generalization performance of the model is estimated for each candidate value of $H \in \{1, 2, ..., 10\}$. The maximum of iteration is set to 10000 to train the model sufficiently and improve the accuracy. The tuning of this parameter can be accomplished by the class GridSearchCV in scikit-learn. Similar to logistic regression, MLP Classifier is sensitive to feature scaling, scaling data properly can improve the performance of classifier. An example of single-hidden-layer neural network implementation with scikit-learn is shown as below.

```
data = np.load("datasets/"+name+".npz")
X = data['train_X']        #load data for training
Y = data['train_Y']
X_test = data['test_X']    #load data for testing
Y_test = data['test_Y']
scaler = StandardScaler()
scaler.fit(X)  # fit on training data
X = scaler.transform(X) #scale the data
X_test = scaler.transform(X_test)  # apply same transformation to test data
model = MLPClassifier()
parameters = {'activation':['logistic'], 'hidden_layer_sizes':[1,2,3,4,5,6,7,8,9,10], \
              'max_iter':[10000]} # tune the best number of neurons in the hidden layer
clf = GridSearchCV(model, parameters,scoring='accuracy',cv=5)
clf.fit(X, Y) #training with different paramemters
#print("Candidate parameters and corresponding performance for "+ name +" are shown below.")
#print(clf.cv_results_) # performance for different parameters
print("The best parameters"+" selected by scikit-learn for "+name +" are shown below.")
print(clf.best_params_) # the best parameters
print("The accuracy"+" of the MLP model on the training sets is "\
      +str(sklearn.metrics.accuracy_score(Y, clf.predict(X))))
print("The loss  of the MLP model on the training sets is "\
      +str(sklearn.metrics.log_loss(Y, clf.predict(X)))+".")
print("The accuracy of the MLP model on the test sets is "\
      +str(sklearn.metrics.accuracy_score(Y_test, clf.predict(X_test))))
print("The loss of the MLP model on the test sets is "\
      +str(sklearn.metrics.log_loss(Y_test, clf.predict(X_test)))+".")
```

: Code Outline for Single-Hidden-Layer Neural Network with scikit-learn

4.2 Number of Hidden Units H

Before presenting the detailed statistics, the number of hidden neurons H should be determined using k-fold cross validation. In k-fold cross-validation, the original sample is randomly partitioned into k equal sized subsamples. Of the$k$ subsamples, a single subsample is retained as the validation data for testing the model, and the remaining $k-1$ subsamples are used as training data. The cross-validation process is then repeated k times (the folds), with each of the k subsamples used exactly once as the validation data. The k results from the folds can then be averaged to produce a single estimation. According the requirement of the project and the lower variance displayed by 5-fold or 10-fold cross-validation [5], the value of k is set to be 5.

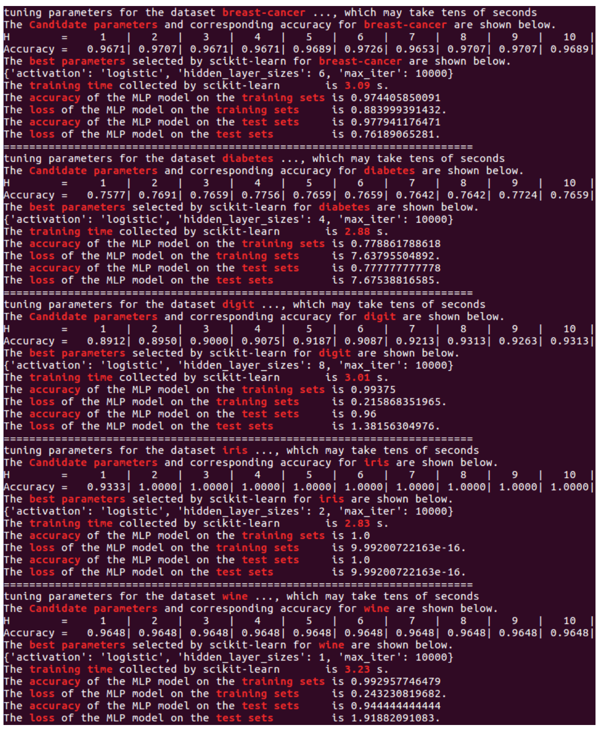
Going through papers, it seems that there is no hard-and-fast rule for the selection of number of hidden units [6], even analytic approach can lead to improper parameters for the datasets in this project [7]. Therefore, for this project, the selection is based on computing accuracy from accuracy scores of cross-validation. In scikit-learn, a class called GridSearchCV, from submodule model selection in scikit-learn, can exhaustively search over specified parameter values for an estimator based on cross validation. With this search method, the optimal parameters for different datasets are obtained, as shown in Table 4. The performance of each value of $H \in 1, 2, ..., 10$, for determining the best value H^*, in the cross validation procedure is reported in Table 5. The process of tuning is also demonstrated via the output of the scikit-learn, which is presented in Figure 7. However, due to the small number of samples, features and lables, the performance of neural network with different numbers of hidden unit may be similar to each other.

Dataset	Optimal Number of Hidden Units H	Accuracy of Training	Training Time
breast-cancer	6	0.974	3.09 s
diabetes	4	0.779	2.88 s
digit	8	0.994	3.01 s
iris	2	1.000	2.83 s
wine	1	0.993	3.23 s

Table 4: Experiment Settings of the Neural Network Model

H	1	2	3	4	5	6	7	8	9	10
breast-cancer	0.9671	0.9707	0.9671	0.9671	0.9689	0.9726	0.9653	0.9707	0.9707	0.9689
diabetes	0.7577	0.7691	0.7659	0.7756	0.7659	0.7659	0.7642	0.7642	0.7724	0.7659
digit	0.8912	0.8950	0.9000	0.9075	0.9187	0.9087	0.9213	0.9313	0.9263	0.9313
iris	0.9333	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000
wine	0.9648	0.9648	0.9648	0.9648	0.9648	0.9648	0.9648	0.9648	0.9648	0.9648

Table 5: Accuracy with Each Value of H for MLP Model



```
tuning parameters for the dataset breast-cancer ..., which may take tens of seconds
The Candidate parameters and corresponding accuracy for breast-cancer are shown below.
H        =     1   |   2   |   3   |   4   |   5   |   6   |   7   |   8   |   9   |   10  |
Accuracy =   0.9671| 0.9707| 0.9671| 0.9671| 0.9689| 0.9726| 0.9653| 0.9707| 0.9707| 0.9689|
The best parameters selected by scikit-learn for breast-cancer are shown below.
{'activation': 'logistic', 'hidden_layer_sizes': 6, 'max_iter': 10000}
The training time collected by scikit-learn       is 3.09 s.
The accuracy of the MLP model on the training sets is 0.974405850091
The loss of the MLP model on the training sets     is 0.883999391432.
The accuracy of the MLP model on the test sets     is 0.977941176471
The loss of the MLP model on the test sets         is 0.76189065281.
=========================================================================
tuning parameters for the dataset diabetes ..., which may take tens of seconds
The Candidate parameters and corresponding accuracy for diabetes are shown below.
H        =     1   |   2   |   3   |   4   |   5   |   6   |   7   |   8   |   9   |   10  |
Accuracy =   0.7577| 0.7691| 0.7659| 0.7756| 0.7659| 0.7659| 0.7642| 0.7642| 0.7724| 0.7659|
The best parameters selected by scikit-learn for diabetes are shown below.
{'activation': 'logistic', 'hidden_layer_sizes': 4, 'max_iter': 10000}
The training time collected by scikit-learn       is 2.88 s.
The accuracy of the MLP model on the training sets is 0.778861788618
The loss of the MLP model on the training sets     is 7.63795504892.
The accuracy of the MLP model on the test sets     is 0.777777777778
The loss of the MLP model on the test sets         is 7.67538816585.
=========================================================================
tuning parameters for the dataset digit ..., which may take tens of seconds
The Candidate parameters and corresponding accuracy for digit are shown below.
H        =     1   |   2   |   3   |   4   |   5   |   6   |   7   |   8   |   9   |   10  |
Accuracy =   0.8912| 0.8950| 0.9000| 0.9075| 0.9187| 0.9087| 0.9213| 0.9313| 0.9263| 0.9313|
The best parameters selected by scikit-learn for digit are shown below.
{'activation': 'logistic', 'hidden_layer_sizes': 8, 'max_iter': 10000}
The training time collected by scikit-learn       is 3.01 s.
The accuracy of the MLP model on the training sets is 0.99375
The loss of the MLP model on the training sets     is 0.215868351965.
The accuracy of the MLP model on the test sets     is 0.96
The loss of the MLP model on the test sets         is 1.38156304976.
=========================================================================
tuning parameters for the dataset iris ..., which may take tens of seconds
The Candidate parameters and corresponding accuracy for iris are shown below.
H        =     1   |   2   |   3   |   4   |   5   |   6   |   7   |   8   |   9   |   10  |
Accuracy =   0.9333| 1.0000| 1.0000| 1.0000| 1.0000| 1.0000| 1.0000| 1.0000| 1.0000| 1.0000|
The best parameters selected by scikit-learn for iris are shown below.
{'activation': 'logistic', 'hidden_layer_sizes': 2, 'max_iter': 10000}
The training time collected by scikit-learn       is 2.83 s.
The accuracy of the MLP model on the training sets is 1.0
The loss of the MLP model on the training sets     is 9.99200722163e-16.
The accuracy of the MLP model on the test sets     is 1.0
The loss of the MLP model on the test sets         is 9.99200722163e-16.
=========================================================================
tuning parameters for the dataset wine ..., which may take tens of seconds
The Candidate parameters and corresponding accuracy for wine are shown below.
H        =     1   |   2   |   3   |   4   |   5   |   6   |   7   |   8   |   9   |   10  |
Accuracy =   0.9648| 0.9648| 0.9648| 0.9648| 0.9648| 0.9648| 0.9648| 0.9648| 0.9648| 0.9648|
The best parameters selected by scikit-learn for wine are shown below.
{'activation': 'logistic', 'hidden_layer_sizes': 1, 'max_iter': 10000}
The training time collected by scikit-learn       is 3.23 s.
The accuracy of the MLP model on the training sets is 0.992957746479
The loss of the MLP model on the training sets     is 0.243230819682.
The accuracy of the MLP model on the test sets     is 0.944444444444
The loss of the MLP model on the test sets         is 1.91882091083.
```

Figure 7: Output of scikit-learn for All the Datasets Based on MLP Model

4.3 Accuracy of the Neural Network Model

Based on the parameters set according to previous illustration, the accuracy of the neural network model on the test set can be obtained by the accuracy_score function from the submodule metrics of scikit-learn. Such result is presented in Table 6. Mini-batch learning of MLP is also tried in the project and the result is shown in Figure 8. From Figure 8, compared to Figure 2 for logistic regression, MLP classifier generates a more smooth learning curve at the cost of more time for training.

Dataset	Accuracy of Test
breast-cancer	0.978
diabetes	0.778
digit	0.960
iris	1.000
wine	0.944

Table 6: Accuracy of the Neural Network Model for Datasets

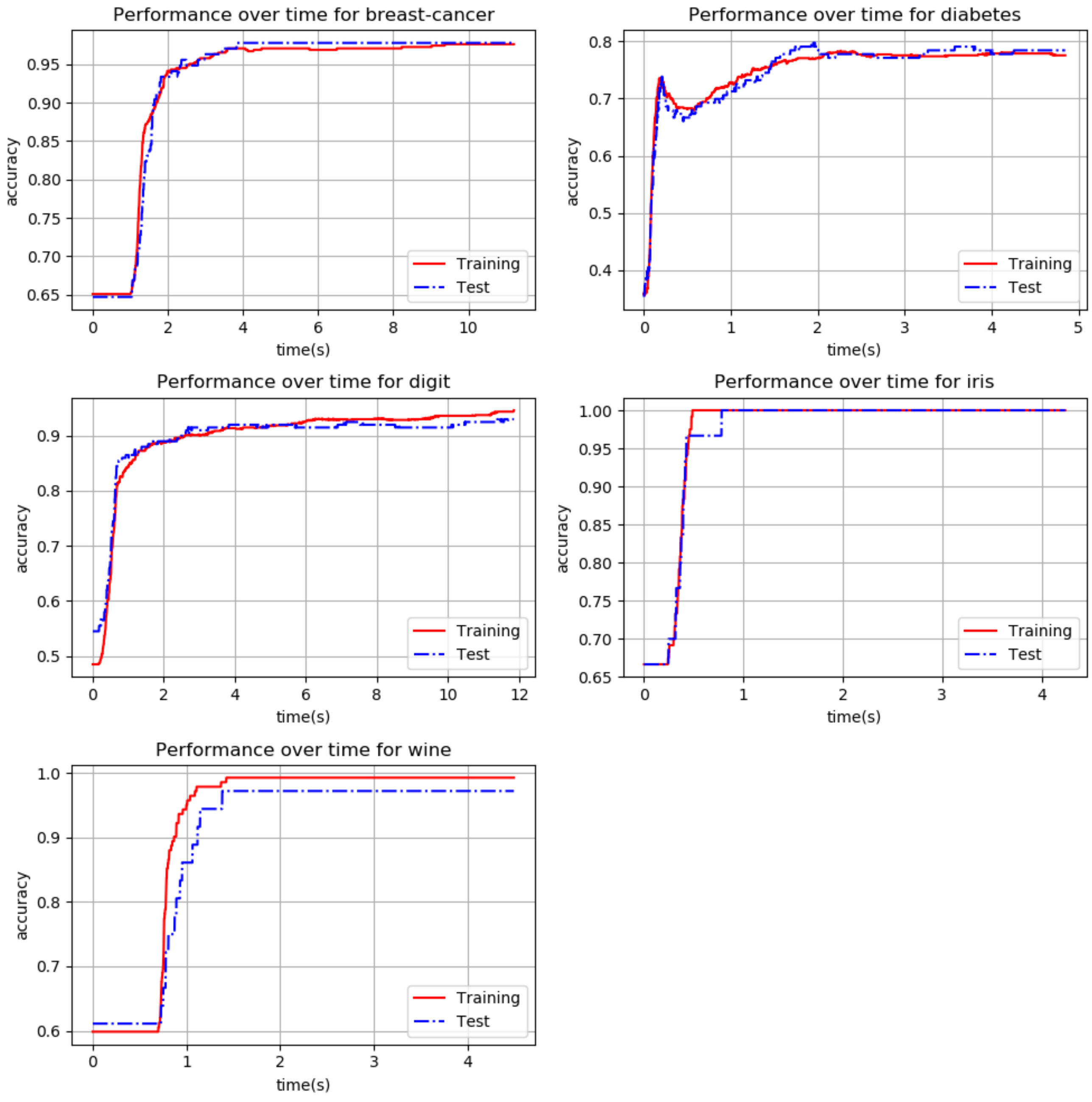


Figure 8: Performance of the Single-Hidden-Layer Neural Network over Time

4.4 Further Study of the Neural Network Model

Beside the basic logistic actication function, the experiments based on ReLU function have also been done for comparison. As shown in Figure 9, the result indicates that ReLU function can lead to faster learning, compared to conventional logistic function, result of which is presented in Figure 8 .

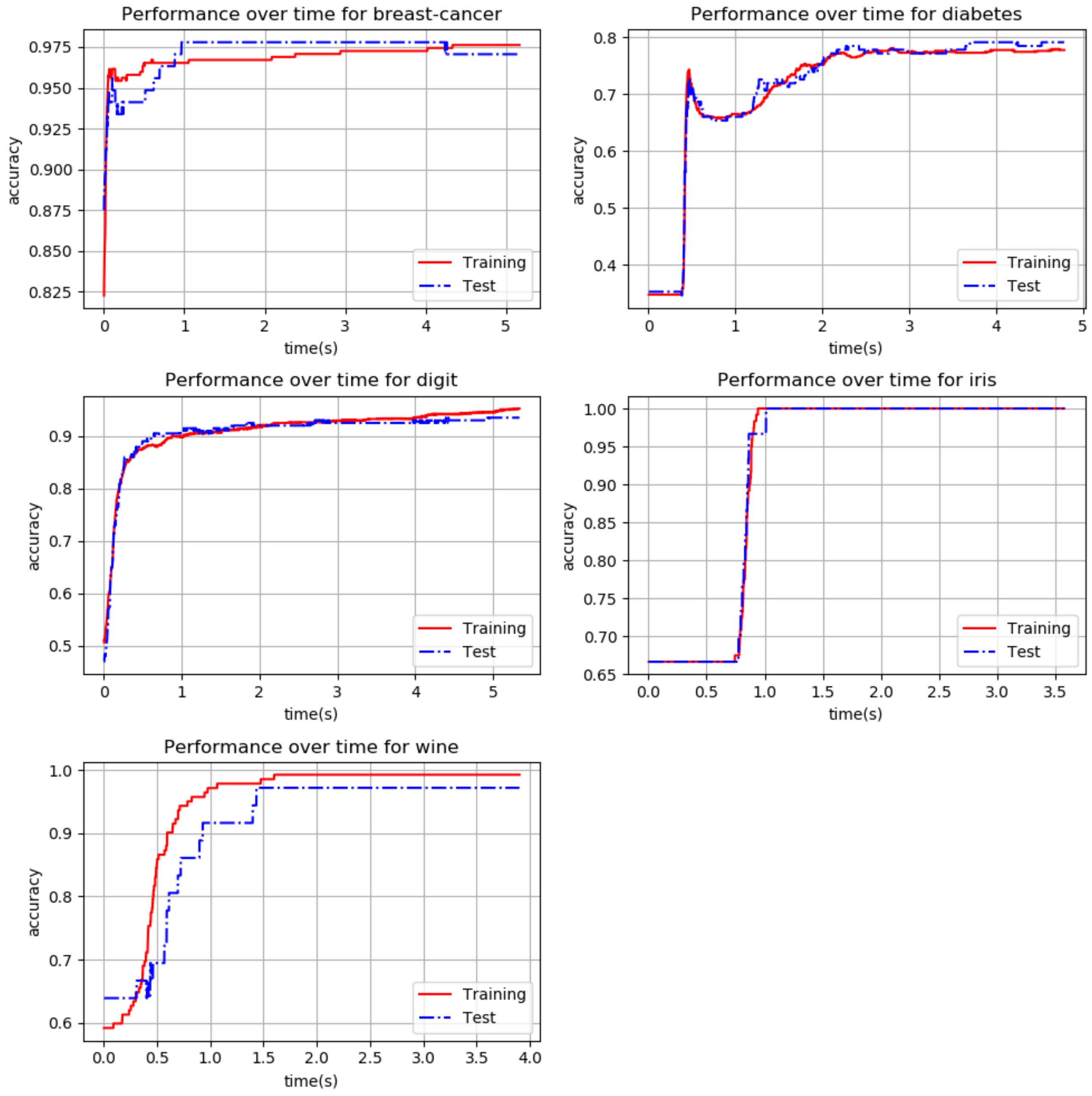


Figure 9: Performance of the Single-Hidden-Layer Neural Network Based on ReLU over Time

5 Task 3: SVM

An SVM model is a representation of the examples as points in space, mapped so that the examples of the separate categories are divided by a clear gap that is as wide as possible. New examples are then mapped into that same space and predicted to belong to a category based on which side of the gap they fall. In addition to performing linear classification, SVMs can efficiently perform a non-linear classification using what is called the kernel trick, implicitly mapping their inputs into high-dimensional feature spaces. [8]

5.1 The Experiment Settings of the Linear SVM Model

Class LinearSVC in scikit-learn is used to solve the SVM model. The major parameter settings of LinearSVC Classifier are presented in Table 7, while other parameters stay the same as default.

Names	Parameter Settings
Classifier	LinearSVC

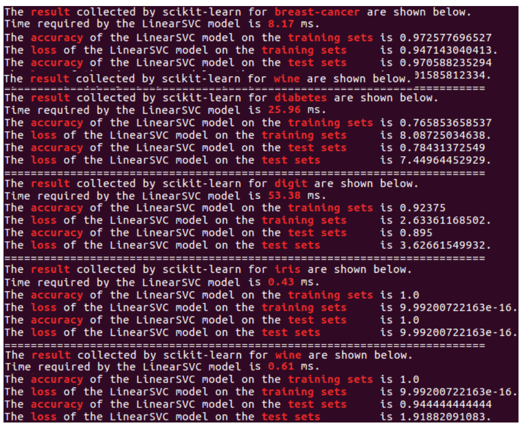
Table 7: Experiment Settings of the Linear SVM Model

One of alternatives to LinearSVC is class SVC. LinearSVC is similar to SVC with parameter kernel=linear, but implemented in terms of liblinear rather than libsvm, so it has more flexibility in the choice of penalties and loss functions and should scale better to large numbers of samples.

Intuitively, feature normalization is needed because SVMs are not invariant to the scale of their input feature spaces: multiplying a feature dimension by a fixed constant gives that dimension more weight in the value of the SVM objective function and, therefore, in the choice of the decision boundary. Therefore, in the absence of prior knowledge, one should choose a normalization method that leaves all feature dimensions in a comparable range. [9] Similar to previous tasks, StandardScaler in scikit-learn is involved to solve the problem. The concrete implementation with scikit-learn is shown as below and the output of scikit-learn for logistic regression is presented in Figure 10.

```
data = np.load("datasets/"+name+".npz")
X = data['train_X']        #load data for training
Y = data['train_Y']
X_test = data['test_X']    #load data for testing
Y_test = data['test_Y']
scaler = StandardScaler()
scaler.fit(X)  # Initialization of the scaler
X = scaler.transform(X)
X_test = scaler.transform(X_test)  # apply same transformation to test data
model = LinearSVC() #use default parameters
model.fit(X, Y) #training
print("The accuracy of the MLP model on the training sets  is "\
        +str(sklearn.metrics.accuracy_score(Y, model.predict(X))))
print("The loss of the MLP model on the training sets is "\
       +str(sklearn.metrics.log_loss(Y, model.predict(X)))+".")
print("The accuracy of the MLP model on the test sets is "\
      +str(sklearn.metrics.accuracy_score(Y_test, model.predict(X_test))))
print("The loss of the MLP model on the test sets is "\
      +str(sklearn.metrics.log_loss(Y_test, model.predict(X_test)))+".")
```

: Code Outline for Linear SVM with scikit-learn



```
The result collected by scikit-learn for breast-cancer are shown below.
Time required by the LinearSVC model is 8.17 ms.
The accuracy of the LinearSVC model on the training sets is 0.972577696527
The loss of the LinearSVC model on the training sets     is 0.947143040413.
The accuracy of the LinearSVC model on the test sets     is 0.970588235294
The result collected by scikit-learn for wine are shown below.91585812334.
==========================================================================
The result collected by scikit-learn for diabetes are shown below.
Time required by the LinearSVC model is 25.96 ms.
The accuracy of the LinearSVC model on the training sets is 0.765853658537
The loss of the LinearSVC model on the training sets     is 8.08725034638.
The accuracy of the LinearSVC model on the test sets     is 0.78431372549
The loss of the LinearSVC model on the test sets         is 7.44964452929.
==========================================================================
The result collected by scikit-learn for digit are shown below.
Time required by the LinearSVC model is 53.38 ms.
The accuracy of the LinearSVC model on the training sets is 0.92375
The loss of the LinearSVC model on the training sets     is 2.63361168502.
The accuracy of the LinearSVC model on the test sets     is 0.895
The loss of the LinearSVC model on the test sets         is 3.62661549932.
==========================================================================
The result collected by scikit-learn for iris are shown below.
Time required by the LinearSVC model is 0.43 ms.
The accuracy of the LinearSVC model on the training sets is 1.0
The loss of the LinearSVC model on the training sets     is 9.99200722163e-16.
The accuracy of the LinearSVC model on the test sets     is 1.0
The loss of the LinearSVC model on the test sets         is 9.99200722163e-16.
==========================================================================
The result collected by scikit-learn for wine are shown below.
Time required by the LinearSVC model is 0.61 ms.
The accuracy of the LinearSVC model on the training sets is 1.0
The loss of the LinearSVC model on the training sets     is 9.99200722163e-16.
The accuracy of the LinearSVC model on the test sets     is 0.944444444444
The loss of the LinearSVC model on the test sets         is 1.91882091083.
```

Figure 10: Output of scikit-learn for All the Datasets Based on Linear SVM Model

5.2 The Experiment Reseult of the Linear SVM Model

Based on scikit-learn, the accuracy results of different datasets, on both the test sets and training test, are obtained, as shown in Table 8. Figure 11 show the change in performance of the linear SVM model over time. It can be noticed that most of datasets result in high accuracy except diabet.

Dataset	Accuracy of Training	Accuracy of Test	Time
breast-cancer	0.972	0.970	8.17ms
diabetes	0.766	0.784	25.96ms
digit	0.923	0.895	53.38ms
iris	1.000	1.000	0.43ms
wine	1.000	0.944	0.61ms

Table 8: Experiment Reseult of the Linear SVM for Datasets

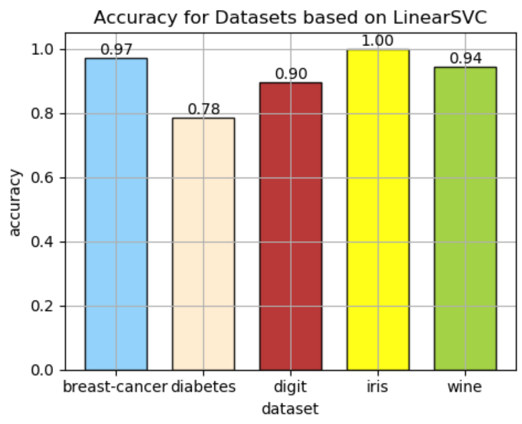


Figure 11: Test Performance of Linear SVM Model

5.3 The Experiment Settings of the RBF SVM Model

Class SVC with default RBF kernel in scikit-learn is used to solve the SVM model. The major parameter settings of LinearSVC Classifier are presented in Table 9, while other parameters stay the same as default.

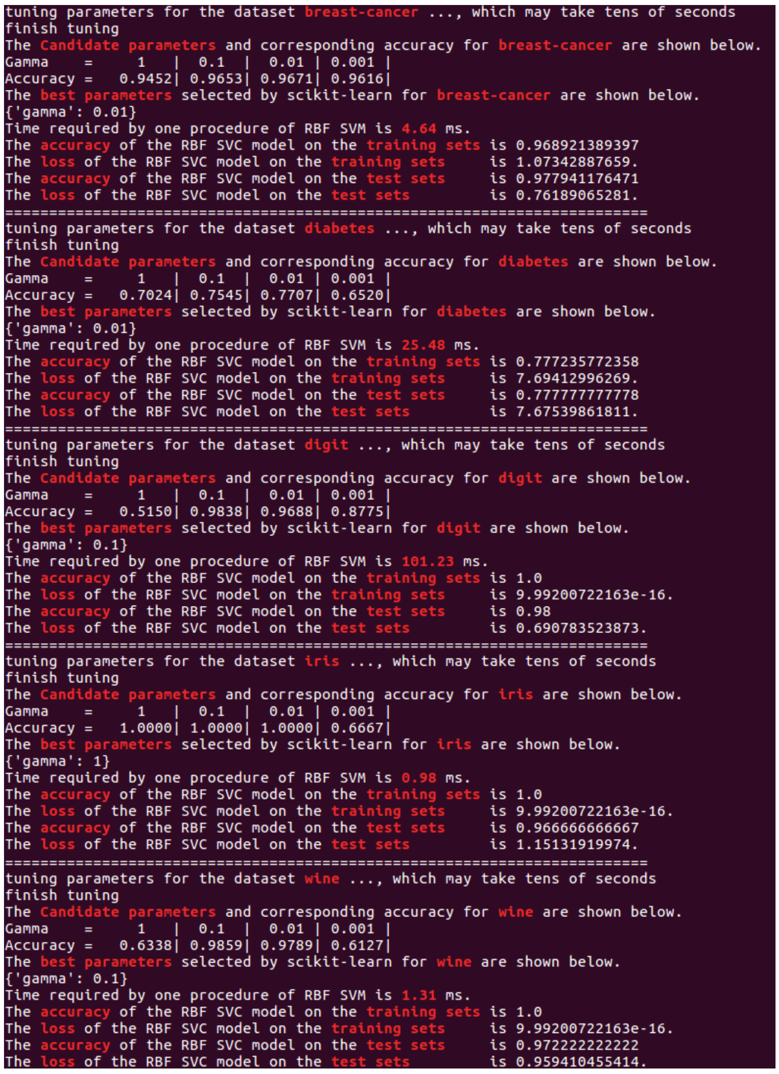
Names	Parameter Settings
Classifier	SVC
Kernel	RBF
γ	Will be Tuned

Table 9: Experiment Settings of the RBF SVM Model

The γ parameter defines how far the influence of a single training example reaches, with low values meaning far and high values meaning close. If γ is too large, the radius of the area of influence of the support vectors only includes the support vector itself, which may fail to prevent overfitting. When γ is very small, the model is too constrained and cannot capture the complexity or shape of the data. [10] The concrete implementation with scikit-learn is shown as below and the output of scikit-learn for logistic regression is presented in Figure 12.

```
data = np.load("datasets/"+name+".npz")
X = data['train_X']         #load data for training
Y = data['train_Y']
X_test = data['test_X']     #load data for testing
Y_test = data['test_Y']
scaler = StandardScaler()
scaler.fit(X)   # fit on training data
X = scaler.transform(X) #scale the data
X_test = scaler.transform(X_test)  # apply same transformation to test data
model = SVC()
parameters = {'gamma':[1,0.1,0.01,0.001]}# tune the best number of neurons in the hidden layer
clf = GridSearchCV(model, parameters,scoring='accuracy',cv=5)
clf.fit(X, Y) #training with different paramemters
#print("Candidate parameters and corresponding performance for "+ name +" are shown below.")
#print(clf.cv_results_) # performance for different parameters
print("The best parameters"+" selected by scikit-learn for "+name +" are shown below.")
print(clf.best_params_) # the best parameters
print("The accuracy"+" of the RBF SVM model on the training sets is "\
        +str(sklearn.metrics.accuracy_score(Y, clf.predict(X))))
print("The loss  of the RBF SVM model on the training sets is "\
        +str(sklearn.metrics.log_loss(Y, clf.predict(X)))+".")
print("The accuracy of the RBF SVM model on the test sets is "\
        +str(sklearn.metrics.accuracy_score(Y_test, clf.predict(X_test))))
print("The loss of the RBF SVM model on the test sets is "\
        +str(sklearn.metrics.log_loss(Y_test, clf.predict(X_test)))+".")
```

: Code Outline for RBF SVM with scikit-learn



```
tuning parameters for the dataset breast-cancer ..., which may take tens of seconds
finish tuning
The Candidate parameters and corresponding accuracy for breast-cancer are shown below.
Gamma    =     1   |  0.1  |  0.01 | 0.001 |
Accuracy =   0.9452| 0.9653| 0.9671| 0.9616|
The best parameters selected by scikit-learn for breast-cancer are shown below.
{'gamma': 0.01}
Time required by one procedure of RBF SVM is 4.64 ms.
The accuracy of the RBF SVC model on the training sets is 0.968921389397
The loss of the RBF SVC model on the training sets     is 1.07342887659.
The accuracy of the RBF SVC model on the test sets     is 0.977941176471
The loss of the RBF SVC model on the test sets         is 0.76189065281.
=========================================================================
tuning parameters for the dataset diabetes ..., which may take tens of seconds
finish tuning
The Candidate parameters and corresponding accuracy for diabetes are shown below.
Gamma    =     1   |  0.1  |  0.01 | 0.001 |
Accuracy =   0.7024| 0.7545| 0.7707| 0.6520|
The best parameters selected by scikit-learn for diabetes are shown below.
{'gamma': 0.01}
Time required by one procedure of RBF SVM is 25.48 ms.
The accuracy of the RBF SVC model on the training sets is 0.777235772358
The loss of the RBF SVC model on the training sets     is 7.69412996269.
The accuracy of the RBF SVC model on the test sets     is 0.777777777778
The loss of the RBF SVC model on the test sets         is 7.67539861811.
=========================================================================
tuning parameters for the dataset digit ..., which may take tens of seconds
finish tuning
The Candidate parameters and corresponding accuracy for digit are shown below.
Gamma    =     1   |  0.1  |  0.01 | 0.001 |
Accuracy =   0.5150| 0.9838| 0.9688| 0.8775|
The best parameters selected by scikit-learn for digit are shown below.
{'gamma': 0.1}
Time required by one procedure of RBF SVM is 101.23 ms.
The accuracy of the RBF SVC model on the training sets is 1.0
The loss of the RBF SVC model on the training sets     is 9.99200722163e-16.
The accuracy of the RBF SVC model on the test sets     is 0.98
The loss of the RBF SVC model on the test sets         is 0.690783523873.
=========================================================================
tuning parameters for the dataset iris ..., which may take tens of seconds
finish tuning
The Candidate parameters and corresponding accuracy for iris are shown below.
Gamma    =     1   |  0.1  |  0.01 | 0.001 |
Accuracy =   1.0000| 1.0000| 1.0000| 0.6667|
The best parameters selected by scikit-learn for iris are shown below.
{'gamma': 1}
Time required by one procedure of RBF SVM is 0.98 ms.
The accuracy of the RBF SVC model on the training sets is 1.0
The loss of the RBF SVC model on the training sets     is 9.99200722163e-16.
The accuracy of the RBF SVC model on the test sets     is 0.966666666667
The loss of the RBF SVC model on the test sets         is 1.15131919974.
=========================================================================
tuning parameters for the dataset wine ..., which may take tens of seconds
finish tuning
The Candidate parameters and corresponding accuracy for wine are shown below.
Gamma    =     1   |  0.1  |  0.01 | 0.001 |
Accuracy =   0.6338| 0.9859| 0.9789| 0.6127|
The best parameters selected by scikit-learn for wine are shown below.
{'gamma': 0.1}
Time required by one procedure of RBF SVM is 1.31 ms.
The accuracy of the RBF SVC model on the training sets is 1.0
The loss of the RBF SVC model on the training sets     is 9.99200722163e-16.
The accuracy of the RBF SVC model on the test sets     is 0.972222222222
The loss of the RBF SVC model on the test sets         is 0.959410455414.
```

Figure 12: Output of scikit-learn for All the Datasets Based on RBF SVM Model

5.4 The Experiment Reseult of the RBF SVM Model

Via grid search, the performance of RBF SVM with different γ have been presented in Table 11. Based on scikit-learn, the accuracy results of different datasets, on both the test sets and training test, are obtained, as shown in Table 10. Figure 13 show the change in performance of the RBF SVM model for different dataset. It can be noticed that most of datasets result in high accuracy except diabet.

Dataset	Optimal γ	Accuracy of Training	Accuracy of Test	Time
breast-cancer	0.01	0.969	0.978	4.57ms
diabetes	0.01	0.777	0.778	24.23ms
digit	0.1	1.000	0.980	96.66ms
iris	1	1.000	0.967	1.00ms
wine	0.1	1.000	0.972	1.25ms

Table 10: Experiment Reseult of the RBF SVM for Datasets

γ	1	0.1	0.01	0.001
breast-cancer	0.9452	0.9653	0.9671	0.9616
diabetes	0.7024	0.7545	0.7707	0.6520
digit	0.5150	0.9838	0.9688	0.8775
iris	1.0000	1.0000	1.0000	0.6667
wine	0.6338	0.9859	0.9789	0.6127

Table 11: Accuracy with Each Value of γ for RBF SVM Model

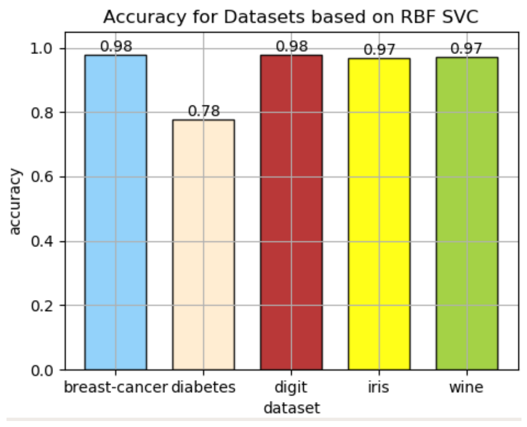


Figure 13: Test Performance of RBF SVM Model

6 Conclusion: Comparison Between Machine Learning Models

Without parameter-selection, experiments in this project demonstrate training time is much shorter for SVMs, followed by logistice regression and neural network. For the classification of datasets in this project, SVMs obtains higher accuracies and more stable results. In this project, the ability of regression has not been consider. However, due to the limited size of datasets, the small number of features and binary outputs, the comparison might not be sufficient to be generalized for other applications.

References

[1] D. Saad, “Online algorithms and stochastic approximations,” *Online Learning*, vol. 5, 1998.

[2] L. Bottou, “Large-scale machine learning with stochastic gradient descent,” in *Proceedings of COMPSTAT’2010*. Springer, 2010, pp. 177–186.

[3] S. Raschka, “About Feature Scaling and Normalization,” http://sebastianraschka.com/Articles/2014_about_feature_scaling.html#feature-scaling---standardization, 2014.

[4] D. P. Kingma and J. Ba, “Adam: A method for stochastic optimization,” *arXiv preprint arXiv:1412.6980*, 2014.

[5] L. Breiman and P. Spector, “Submodel selection and evaluation in regression. the x-random case,” *International statistical review/revue internationale de Statistique*, pp. 291–319, 1992.

[6] A. Elisseeff and H. Paugam-Moisy, “Size of multilayer networks for exact learning: analytic approach,” in *Advances in Neural Information Processing Systems*, 1997, pp. 162–168.

[7] S. Lawrence, C. L. Giles, and A. C. Tsoi, “What size neural network gives optimal generalization? convergence properties of backpropagation,” Tech. Rep., 1998.

[8] V. Roth and V. Steinhage, “Nonlinear discriminant analysis using kernel functions,” in *Advances in neural information processing systems*, 2000, pp. 568–574.

[9] A. Stolcke, S. Kajarekar, and L. Ferrer, “Nonparametric feature normalization for svm-based speaker verification,” in *2008 IEEE International Conference on Acoustics, Speech and Signal Processing*, March 2008, pp. 1577–1580.

[10] scikit-learn v0.19.1, “RBF SVM parameters,” http://scikit-learn.org/stable/auto_examples/svm/plot_rbf_parameters.html, 2017.