Comparison of Some Statistical Learning Methods' Performance on Iris Data Set in R

Zhengfu Li
Georgia Institute of Technology
zhengfuli@gatech.edu

Abstract

Some classic but popular statistical machine learning algorithms were briefly introduced, and then performed on the Iris flower data set, including K-nearest neighbors (KNN), logistical regression (LR), linear/quadratic discriminant analysis (LDA/QDA), support vector machine (SVM), decision trees and random forest. Especially, some methods that were not addressed in the class were experimented, i.e., weighted KNN and multi-layer perceptron learning machine (MLP), a kind of neural net. Besides, dimensionality of the data set was reduced from 4 to 2 using principle component analysis (PCA) and LDA respectively, and then simple linear SVM was performed on the reduced data to compare the difference of these two de-dimensional methods. During all the experiments, 10-fold cross-validation was employed to either help decide the optimal parameters for a certain model or estimate the test error of the model. The objectives of the experiments are to compare the performance of all these methods on the Iris flower data set and explore their properties.

1 Introduction

The Iris flower data set is one of the most popular data set in the UCI repository. [1] The data set consists of 50 samples from each of three species of Iris (Iris setosa, Iris virginica and Iris versicolor). Four features were measured from each sample: the length and the width of the sepals and petals, in centimeters, which are illustrated as the following figures:

[1] *Iris flower species in the Iris data set*: *Iris setosa, Iris versicolor, and Iris virginica from left to right.*

The Iris data set that has already been included in the standard R installation was used to conduct all the experiments. Considering the categorical feature of sub-species of the flower in this data set, the experiments on it are to solve a multivariate multi-class classification problem. KNN/weighted KNN methods can easily handle this type of problem, while the simple LR method may need to be generalized using a function referred as [3] softmax function and multinomial probit model instead of a binomial one. And for LDA/QDA and SVM, the multi-class issue can be solved by employing a one-over-rest strategy, which can automatically be applied in R and thus requires no worry. [4] A MLP is a class of feedforward artificial neural network. An MLP consists of, at least, three layers of nodes: an input layer, a hidden layer and an output layer. Except for the input nodes, each node is a neuron that uses a

nonlinear activation function. MLP utilizes a supervised learning technique called backpropagation for training. Its multiple layers and non-linear activation distinguish MLP from a linear perceptron. It can distinguish data that is not linearly separable and is capable to handle multi-class classification problem very well. As introduced in my presentation, PCA and LDA have very different goals when reducing the dimensionality of data, despite the form of their solutions are similar (eigenvectors of a particular matrix). Intuitively, PCA aims to maintain the maximum volume of information of data (proportion of variation explained) after projecting the original data into the de-dimensional space, while LDA aims to make the projected data most separated in the new space. To see their difference and influence on the data and classification, they were performed on the Iris data set and a simple linear SVM was then performed. For all the experiments, the main indicate of the performance was test error produced by 10-fold cross-validation.

2 Related Work

[1] The Iris flower data set is also called Fisher's Iris data set because it was firstly introduced by the British statistician and biologist Ronald Fisher in his 1936 paper "*The use of multiple measurements in taxonomic problems as an example of linear discriminant analysis*", where he developed a linear discriminant model to distinguish the species from each other. Based on Fisher's linear discriminant model, this data set became a typical test case for many statistical classification techniques in machine learning such as support vector machines. [5] A lot of unsupervised learning research was also conducted on this data set to compare supervised learning and unsupervised learning. Since Iris data set is pretty "clean" and sound, it is very popular among machine learning newbies and has been widely used to tweak or develop methods such as [6] MLP and it is usually not hard to obtain a good prediction accuracy on this data set.

3 Methods

A high-level review of each method used was provided. Sound mathematical proof or technical details may be missing as they are outside of the content of this paper.

3.1 Analyze the data

It is necessary to display the original data set in terms of each pairs of features and show their distributions (estimated probability density functions). Also it's important to estimate the correlation indexes between each group of data in terms of these features. Another important index to evaluate the data set is the area under the curve (AUC), [7] the AUC is equal to the probability that a classifier will rank a randomly chosen positive instance higher than a randomly chosen negative one (assuming 'positive' ranks higher than 'negative'). By calculating the AUC of this data set, we can evaluate the prediction value of this data set.

3.2 K-nearest neighbors and weighted KNN

[8] In KNN classification, the output is a class membership. An object is classified by a majority vote of its neighbors, with the object being assigned to the class

most common among its k nearest neighbors (k is a positive integer, typically small). The weighted KNN classifier can be viewed as an extension of simple KNN assigning the k nearest neighbors a weight 1 / k and all others 0 weight. Moreover, the definition of distance when looking for neighbors can be customized as chebyscheff or rectangle and triangle distance and by default it is Euclidean distance.

3.3 Multinomial logistic regression

[9] [10] In our problem setting, we are solving a multinomial disordered logistic regression problem. Compared to the normal logistic regression, we have to consider it as a multinomial probit model rather than a binomial model when constructing the likelihood function. And more importantly, we need to have a function that converts the output of the linear model into probabilities (values that range from 0 to 1). [3] A usual option is softmax function, suppose our model is:

$$\begin{aligned}\ln \Pr(Y_i = 1) &= \boldsymbol{\beta}_1 \cdot \mathbf{X}_i - \ln Z \\ \ln \Pr(Y_i = 2) &= \boldsymbol{\beta}_2 \cdot \mathbf{X}_i - \ln Z \\ &\cdots\cdots \\ \ln \Pr(Y_i = K) &= \boldsymbol{\beta}_K \cdot \mathbf{X}_i - \ln Z\end{aligned}$$

Where the extra terms –ln$Z$ is to make sure that all of them add up to 1:

$$\sum_{k=1}^{K} \Pr(Y_i = k) = 1$$

Get the exponentials on each side:

$$\begin{aligned}\Pr(Y_i = 1) &= \frac{1}{Z} e^{\boldsymbol{\beta}_1 \cdot \mathbf{X}_i} \\ \Pr(Y_i = 2) &= \frac{1}{Z} e^{\boldsymbol{\beta}_2 \cdot \mathbf{X}_i} \\ &\cdots\cdots \\ \Pr(Y_i = K) &= \frac{1}{Z} e^{\boldsymbol{\beta}_K \cdot \mathbf{X}_i}\end{aligned}$$

We can then solve Z out by summing them up and substituting in the previous formula:

$$\begin{aligned}1 = \sum_{k=1}^{K} \Pr(Y_i = k) &= \sum_{k=1}^{K} \frac{1}{Z} e^{\boldsymbol{\beta}_k \cdot \mathbf{X}_i} \\ &= \frac{1}{Z} \sum_{k=1}^{K} e^{\boldsymbol{\beta}_k \cdot \mathbf{X}_i}\end{aligned}$$

$$Z = \sum_{k=1}^{K} e^{\boldsymbol{\beta}_k \cdot \mathbf{X}_i}$$

And thus the probabilities of each predicted class is in the following form:

$$\begin{aligned}\Pr(Y_i = 1) &= \frac{e^{\boldsymbol{\beta}_1 \cdot \mathbf{X}_i}}{\sum_{k=1}^{K} e^{\boldsymbol{\beta}_k \cdot \mathbf{X}_i}} \\ \Pr(Y_i = 2) &= \frac{e^{\boldsymbol{\beta}_2 \cdot \mathbf{X}_i}}{\sum_{k=1}^{K} e^{\boldsymbol{\beta}_k \cdot \mathbf{X}_i}} \\ &\cdots\cdots \\ \Pr(Y_i = K) &= \frac{e^{\boldsymbol{\beta}_K \cdot \mathbf{X}_i}}{\sum_{k=1}^{K} e^{\boldsymbol{\beta}_k \cdot \mathbf{X}_i}}\end{aligned}$$

And it is referred as softmax function, by using maximum likelihood estimation we can get the parameters of our model just like the binomial logistic regression. In practice, the optimization of the likelihood function is actually complex and trivial, but we can leave it to R.

3.4 Linear/Quadratic discriminant analysis

Two-class LDA for classification problem is actually assuming the observations are subject to normal distributions with the same variance but different mean values. By estimating mean values from the observations, and plug it in the probability density function of normal distributions, the prediction is made by employing maximum a posterior rule. [11] After simplifying, the decision criterion becomes a threshold on the dot product:

$$\vec{w} \cdot \vec{x} > c$$

Where w is our linear model weights and c is some constant defined as:

$$\vec{w} = \Sigma^{-1}(\vec{\mu}_1 - \vec{\mu}_0)$$
$$c = \frac{1}{2}(T - \vec{\mu}_0{}^T \Sigma_0^{-1} \vec{\mu}_0 + \vec{\mu}_1{}^T \Sigma_1^{-1} \vec{\mu}_1)$$

When multi-class classification is needed, the classes may be partitioned, and a standard LDA used to classify each partition. A common example of this is "one against the rest" where the points from one class are put in one group, and everything else in the other, and then LDA applied. This will result in multiple classifiers, whose results are combined. Another common method is pairwise classification, where a new classifier is created for each pair of classes (giving $C(C-1)/2$ classifiers in total), with the individual classifiers combined to produce a final classification.

3.5 Support vector machine

SVM aims to find out the separating hyperplane that mostly separates each group of data points. For the case that observations are not linearly separable, a hyper-parameter *cost* may be customized to allow some observations to be in the wrong margin or hyperplane. By maximizing the margin (soft margin), we are trying to minimize:
[12]

$$\left[\frac{1}{n}\sum_{i=1}^{n}\max\left(0, 1 - y_i(\vec{w}\cdot\vec{x}_i - b)\right)\right] + \lambda\|\vec{w}\|^2$$

We can write the objective function into a form of some target function with a constraint (primal problem). Then we can construct a Lagrangian function as its dual problem. By solving the dual problem, we can get the solutions to the primal problem. Usually, stochastic gradient descent or Newton's method is employed in the optimization process. For multi-class classification problem, SVM can still producing classification rules by using the "one-over-the-rest" rule introduced in the last section.

3.6 Decision trees and random forest

The (binary) decision tree splits each time according to the feature that reduces the deviance of observations the most. And when deciding which child node to extend the tree, it's the side that lower the impurity of the data the most and the evaluation of impurity can be Gini index, cross entropy or misclassification rate. For random forest, it generates a bunch of classification trees and for each one, it uses only parts of the features of the original data randomly. In this way, random forest is robust to noise/outliers and usually has better performance than the normal classification tree. And in practice, 100 stumps is a good start.

3.7 Multi-layer perceptron learning machine

[13] The MLP learns by changing connection weights after each piece of data is processed (feed forward), based on the amount of error in the output compared to the expected result. This is an example of supervised learning, and is carried out through back propagation. After feeding the data to the network with initial random weights, suppose the error between predicted results and ground truth is:

$$e_j(n) = d_j(n) - y_j(n),$$

The nodes weights are adjusted to minimize the entire error:

$$\mathcal{E}(n) = \frac{1}{2}\sum_j e_j^2(n).$$

Using gradient descent, the adjustment at each node is:

$$\Delta w_{ji}(n) = -\eta \frac{\partial \mathcal{E}(n)}{\partial v_j(n)} y_i(n)$$

where

$$-\frac{\partial \mathcal{E}(n)}{\partial v_j(n)} = e_j(n)\phi'(v_j(n))$$

And the fi function is the activation function (radial basis function in our experiment) at each node discussed before. The calculations of adjustment to the nodes in the hidden layers are much more complicated and trivial, but it should follow the chain rule of getting derivatives:

$$-\frac{\partial \mathcal{E}(n)}{\partial v_j(n)} = \phi'(v_j(n)) \sum_k -\frac{\partial \mathcal{E}(n)}{\partial v_k(n)} w_{kj}(n)$$

In this way, MLP iteratively updates the weights in the network by feeding observations forward and backing the error so that the entire error is consistently reduced. But there is no guarantee that the network will converge even there are hyper-parameters to tune. In R, [14] RSNNS package provides a great manual for creating and tuning all kinds of neural networks.

3.8 Principle component analysis and linear discriminant analysis for dimensionality reduction

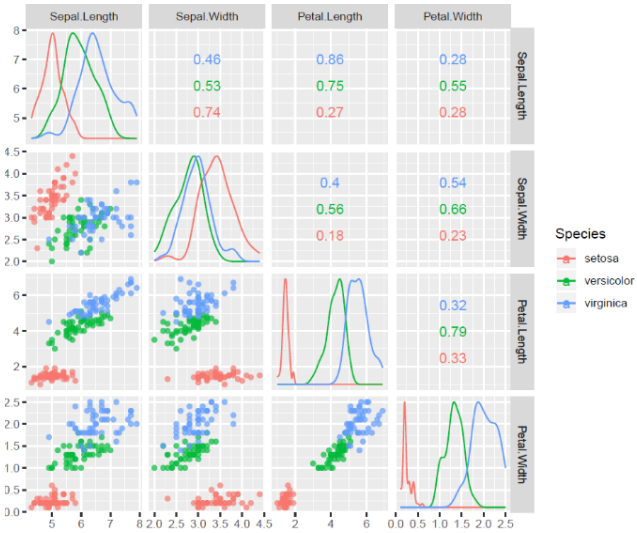
PCA and LDA for dimensionality reduction are detailed in my presentation script. Basically, they seek to projecting the data into a lower rank space with different objectives. And the trade-off between these two different methods really depends on the goal after reducing the dimensions of the original data.

4 Experiments

4.1 Analyze the data set

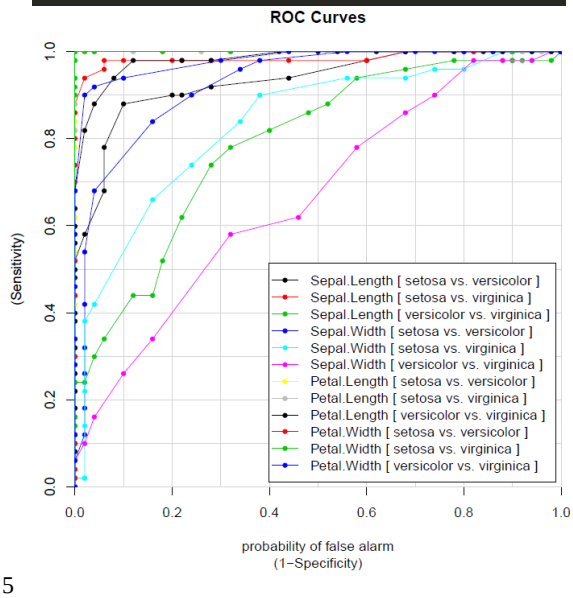
```
library(ggplot2)
library(GGally)
p = ggscatmat(iris, columns = 1:4, color
= "Species", alpha = 0.7); print(p)
```

It can be seen that Iris setosa observations are totally linear separable with the other



two groups of observations in all features. And Iris versicolor and Iris virginica observations more or less overlap each other. Note that the feature of sepal's length has pretty notable correlations with the features of sepal's width and petal's length. Then calculate the AUC:

```
library(caTools)
irisValues =
iris[sample(1:nrow(iris),length(1:nrow(ir
is))),1:ncol(iris)]
colAUC(irisValues[,1:4], irisValues[,5],
plotROC = TRUE)
```



Intuitively, all AUC is larger than 0.5 and it means a classifier may be better than random guess if setting threshold and parameters properly and thus the data set has prediction value.

4.2 K-nearest neighbor and weighted KNN

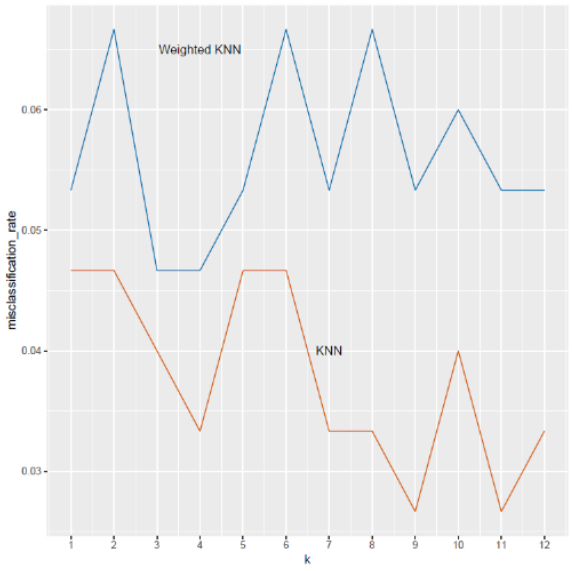
The weighted KNN can be implemented by using kknn package in R:

```
kknn(Species ~ ., k = k, iris[train,],
iris[valid,], kernel = "optimal")
```

and KNN can be implemented by simply using:

```
knn(iris[train, -5], iris[valid, -5],
iris[train, 5], k = k)
```

Using 10-fold cross-validation, the misclassification rate of simple KNN and weighted KNN with different K values is plotted as follows:



In the case of iris data set, it can be seen that KNN generally has better performance than weighted KNN. The highest classification accuracy in 10-fold cross-validation of KNN is about 97.3% when K equals to 9.

4.3 Multinomial logistic regression

The multinomial logistical regression can be implemented by:

```
library(caret)
library(nnet)
mlr = multinom(Species ~ ., data = train)
```

Especially, since the logistic regression outputs probabilities that a data point belongs to a certain class, we need to convert the predicted probabilities into class indexes:

```
probs = predict(mlr, valid, "probs")
cum_probs = t(apply(probs, 1, cumsum))
# Draw random values
vals = runif(nrow(valid))
# Join cumulative probabilities and
random draws
tmp = cbind(cum_probs, vals)
# For each row, get choice index.
k = ncol(probs)
ids = 1 + apply(tmp, 1,
function(x){length(which(x[1:k] <
x[k+1]))})
# print(ids)
cv_error[i] = mean(ids !=
as.numeric(valid$Species))
```

The prediction accuracy of multinomial logistic regression in 10-fold cross-validation is 96%.

4.4 Linear / Quadratic discriminant analysis

The LDA/QDA can be implemented in R by:

```
lda = lda(Species ~ ., data = train)
qda = qda(Species ~ ., data = train)
```

The predicted values and misclassify error are:

```
pred = predict(lda, newdata = valid)
lda_cv_error[i] = mean(pred$class !=
valid$Species)
```

The prediction accuracies of LDA and QDA in 10-fold cross-validation are 98% and 97.3% respectively.

4.5 Support vector machine

For SVM, the string format feature of

Species in Iris data set needs to be converted into a numerical factor:

```
iris$Species = factor(iris$Species,
labels = c(1,2,3))
```

For linear kernel SVM, the penalty parameter *cost* needs to be found, for radial kernel SVM, *gamma* needs to be found except for *cost*, and for polynomial kernel SVM, *degree* needs to be found except for *cost:*

```
tuning_linear = tune(svm, Species ~ .,
data = iris, kernel = "linear", ranges =
list(cost = c(1e-3, 1e-2, 1e-1, 5e-1, 1,
5, 10, 100)))

tuning_radial = tune(svm, Species ~ .,
data = iris, kernel = "radial", ranges =
list(cost = c(1e-3, 1e-2, 1e-1, 5e-1, 1,
5, 10, 100), gamma = c(1e-3, 1e-2, 1e-1,
5e-1, 1, 5, 10, 100)))

tuning_poly = tune(svm, Species ~ ., data
= iris, kernel = "polynomial", ranges =
list(cost = c(1e-3, 1e-2, 1e-1, 5e-1, 1,
5, 10, 100), degree = c(2, 3, 4)))
print(summary(tuning_poly))
```

The prediction accuracy of linear kernel SVM in 10-fold cross-validation is 96.6% with *cost* equals to 1; the prediction accuracy of radial kernel SVM in 10-fold cross-validation is 97.3% with *cost* equals to 1 and *gamma* equals to 0.1; the prediction accuracy of polynomial kernel SVM in 10-fold cross-validation is 95.3% with *cost* equals to 100 and *degree* equals to 3.

4.6 Decision trees and random forest

The classification tree in R can be implemented by:

```
library(party)
ct = ctree(Species ~ ., data = train)
```

The random forest is implemented by:

```
randomForest(Species ~ ., data
= train, mtry = 2, ntree = 100)
```

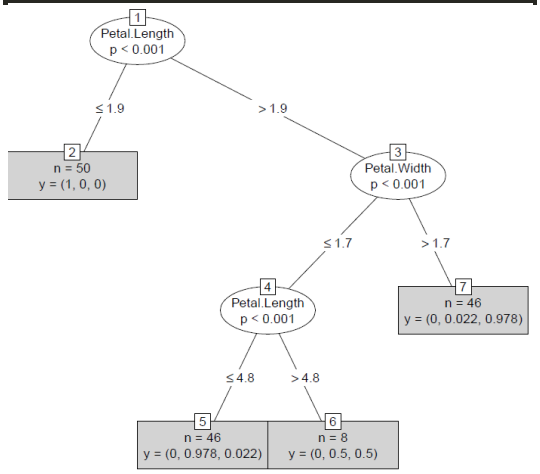
The prediction results are:

```
pred =  predict(ct, newdata = valid)
ct_cv_error[i] = mean(pred !=
valid$Species)
```

The prediction accuracy of classification tree in 10-fold cross-validation is 94.7%. The prediction accuracy of random forest in 10-fold cross-validation is 96%.

Visualize the classification tree:

```
plot(classification_tree, type =
"simple")
```



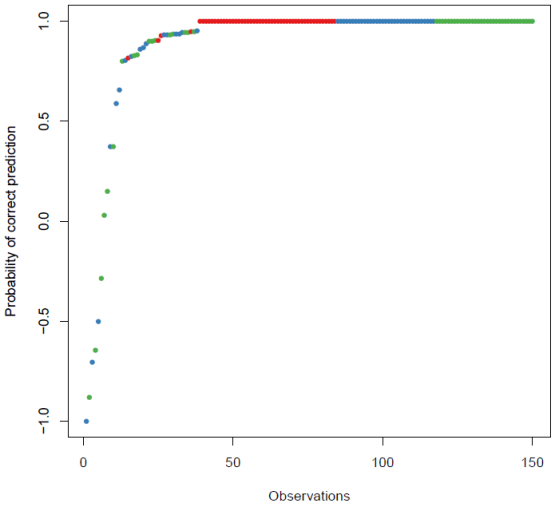
Display the importance of 4 features in the random forest model:

```
print(importance(random_forest))
```

Output:

```
             MeanDecreaseGini
Sepal.Length         11.094097
Sepal.Width           2.479974
Petal.Length         38.582073
Petal.Width          47.123523
```

Besides, the probability of correctly predicting each observation is plotted:



It can be seen that some data points in the Iris versicolor and Iris virginica groups are really hard to predict.

4.7 Multi-layer perceptron learning machine

For MLP, the data set also needs to convert the string format feature into numerical class indexes:

```
library(caret)
library(RSNNS)
set.seed(1)
data(iris)
iris =
iris[sample(1:nrow(iris),length(1:nrow(ir
is))),1:ncol(iris)]
irisValues= iris[,1:4]
irisTargets = decodeClassLabels(iris[,5])
iris =
splitForTrainingAndTest(irisValues,
irisTargets, ratio = 0)
iris = normTrainingAndTestSet(iris)
```

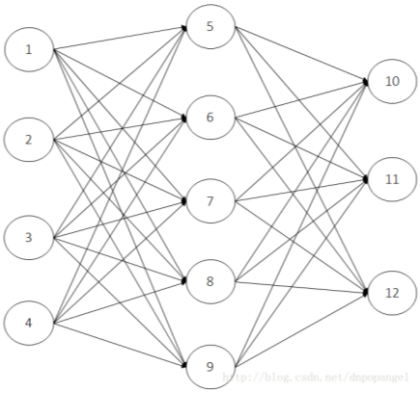
The MLP can be implemented by:

```
model = mlp(X_train, y_train, size = 5,
learnFunc = "BackpropBatch",
          learnFuncParams = c(10, 0.1),
maxit = 100,
          inputsTest = X_valid,
targetsTest = y_valid)
```

The predictions are:

```
pred = predict(model, X_valid)
result = confusionMatrix(y_valid, pred)
cv_error[i] =
sum(diag(result))/sum(result)
```

The MLP model generated is:



The model has one hidden layer consisted of 5 nodes and the network converged in 60 iterations. The prediction accuracy of MLP in 10-fold cross-validation is 96.7%.

4.8 Principle component analysis and linear discriminant analysis for dimensionality reduction

To easier implement the dimensionality reduction and visualize the result, this section is implemented using Python. The projections are done by:

```
import matplotlib as mpl
import matplotlib.pyplot as plt
import numpy as np
from sklearn import datasets
from sklearn.decomposition import PCA
from sklearn import model_selection
from sklearn import svm
from sklearn.discriminant_analysis import
LinearDiscriminantAnalysis
iris = datasets.load_iris()
X = iris.data
y = iris.target
target_names = iris.target_names
pca = PCA(n_components=2)
X_r = pca.fit(X).transform(X)
lda =
LinearDiscriminantAnalysis(n_components=2
)
X_r2 = lda.fit(X, y).transform(X)
```

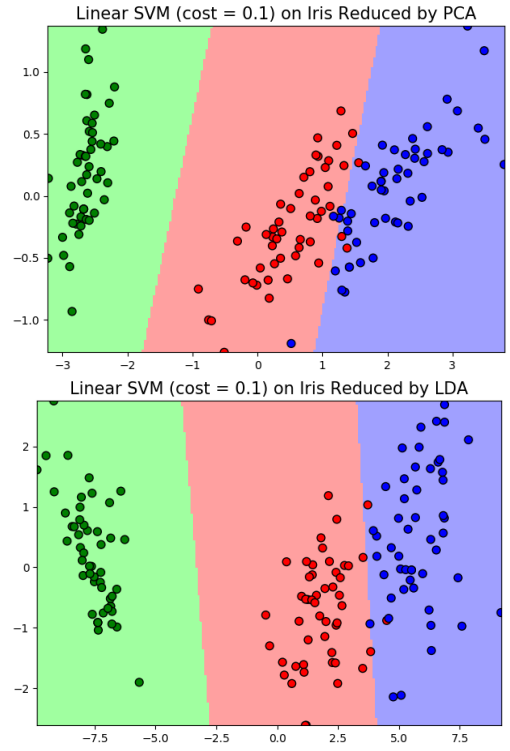
And visualize:

```
X_train, X_valid, y_train, y_valid =
model_selection.train_test_split(X_r, y,
test_size = 0.1, random_state = 0)
clf = svm.SVC(C = 0.1, kernel = 'linear',
decision_function_shape = 'ovr')
clf.fit(X_train, y_train.ravel())
print(clf.score(X_valid, y_valid))

clf.fit(X_r, y.ravel())
x1_min, x1_max = X_r[:, 0].min(), X_r[:,
0].max()
x2_min, x2_max = X_r[:, 1].min(), X_r[:,
```

```
1].max()
x1, x2 = np.mgrid[x1_min:x1_max:200j,
x2_min:x2_max:200j]
grid_test = np.stack((x1.flat, x2.flat),
axis = 1)
grid_hat = clf.predict(grid_test)
grid_hat = grid_hat.reshape(x1.shape)

cm_light =
mpl.colors.ListedColormap(['#A0FFA0',
'#FFA0A0', '#A0A0FF'])
cm_dark = mpl.colors.ListedColormap(['g',
'r', 'b'])
plt.figure()
plt.pcolormesh(x1, x2, grid_hat,
cmap=cm_light)
plt.scatter(X_r[:, 0], X_r[:, 1], c = y,
edgecolors = 'k', s = 50, cmap = cm_dark)
plt.scatter(X_r[:, 0], X_r[:, 1], s =
120, facecolors = 'none', zorder = 10)
# plt.xlabel(u'花萼长度', fontsize = 13)
# plt.ylabel(u'花萼宽度', fontsize = 13)
plt.xlim(x1_min, x1_max)
plt.ylim(x2_min, x2_max)
plt.title(u'Linear SVM (cost = 0.1) on
Iris Reduced by PCA', fontsize = 15)
```



When using *cost* = 0.1, the prediction accuracy of linear SVM on data after PCA is 93.3% and on data after LDA is 100%. When using *cost* = 0.5, both accuracies are 100%. However, it's still obvious to see that for classification after dimensionality reduction, LDA may be a better choice than PCA in that it separates the projected data better thus it becomes easier to classify.

5 Conclusions

KNN	97.3%
Weighted KNN	95.7%
Multinomial LR	96%
LDA	98%
QDA	97.3%
SVM (Radial)	97.3%
Decision Tree	94.7%
Random Forest	96%
Multilayer Perceptron	96.7%
Linear SVM after PCA (Dimensions down to 2)	100%
Linear SVM after LDA (Dimensions down to 2)	100%

For the original Iris flower data, KNN has pretty good performance but it may be hard to interpret when the dimension of data is 4. What's weird is that weighted KNN has worse performance than the normal KNN. Multinomial logistic regression and decision tree are relatively weaker compared to LDA and random forest. After dimensionality reduction, classification on the data using simple SVM can produce really great results.

References

[1] https://en.wikipedia.org/wiki/Iris_flower_data_set

[2] https://cran.r-project.org/web/packages/kknn/kknn.pdf

[3] https://en.wikipedia.org/wiki/Multinomial_logistic_regression
[4] https://en.wikipedia.org/wiki/Multilayer_perceptron
[5] https://archive.ics.uci.edu/ml/datasets/Iris/
[6] http://lab.fs.uni-lj.si/lasin/wp/IMIT_files/neural/doc/seminar8.pdf
[7] https://en.wikipedia.org/wiki/Receiver_operating_characteristic#Area_under_the_curve
[8] https://en.wikipedia.org/wiki/K-nearest_neighbors_algorithm#The_weighted_nearest_neighbour_classifier
[9] https://www.r-bloggers.com/how-to-multinomial-regression-models-in-r/
[10] http://www.ats.ucla.edu/stat/r/dae/mlogit.htm
[11] https://en.wikipedia.org/wiki/Linear_discriminant_analysis#Multiclass_LDA
[12] https://en.wikipedia.org/wiki/Support_vector_machine
[13] https://en.wikipedia.org/wiki/Multilayer_perceptron
[14] https://cran.r-project.org/web/packages/RSNNS/RSNNS.pdf