SrikanthDerebail

SVM with principal component analysis

3
voters

last run 2 years ago · IPython Notebook HTML · 1,082 views
using data from Pima Indians Diabetes Database · Public

Notebook

This is my first machine learning algorithm in Kaggle. I will explore this dataset and try Logisctic regression, and Support Vector Machine classification algorithm on the diabetes dataset. The plan in the mind is as follows:

1. Apply principal Principal Component Analysis and use fewer components to train and test a logistic regression classifier. If the accuracy is not good, it probably means that the decision boundary is nonlinear, hence go to SVM.
2. Train and test a SVM algorithm and optimize the hyperparameters using grid search. Apply PCA and see how the number of principal components influence the accuracy.

In [1]:

```
import numpy as np # linear algebra
import pandas as pd # data processing, CSV file I/O (e.g. pd.read_csv)
import matplotlib.pyplot as plt

# Input data files are available in the "../input/" directory.
# For example, running this (by clicking run or pressing Shift+Enter) will list the files in the input directory
data = pd.read_csv("../input/diabetes.csv")
data.head(5)

#from subprocess import check_output
#print(check_output(["ls", "../input"]).decode("utf8"))
```

Out[1]:

	Pregnancies	Glucose	BloodPressure	SkinThickness	In
0	6	148	72	35	0
1	1	85	66	29	0
2	8	183	64	0	0
3	1	89	66	23	9
4	0	137	40	35	1

In [2]:

```
# Total number of rows in the dataset
print(len(data))
```

```
768
```

In [3]:

```
#Separate features and labels
X = data.iloc[:,0:8].values
y = data.iloc[:,8].values

#Test train split
from sklearn.model_selection import train_test_split
X_train,X_test,y_train,y_test = train_test_split(X,y,te
st_size=0.2)

#Standard scaling
from sklearn.preprocessing import StandardScaler
sc = StandardScaler()
X_train = sc.fit_transform(X_train)
X_test = sc.transform(X_test)

#Applying PCA here
from sklearn.decomposition import PCA
pca = PCA(n_components= None) #We will set it none so th
at we can see the variance explained and then choose no o
f comp.
X_train = pca.fit_transform(X_train)
X_test = pca.transform(X_test)

explained_variance = pca.explained_variance_ratio_
explained_variance
```

Out[3]:

```
array([ 0.25937072,  0.21459004,  0.1260821
8,  0.11376638,  0.09528477,
        0.08698106,  0.05264971,  0.0512751
4])
```

Let us try with 2 principal components and fit a logistic regression to see the performance.

In [4]:

```
X_train,X_test,y_train,y_test = train_test_split(X,y,te
st_size=0.2)
sc = StandardScaler()
X_train = sc.fit_transform(X_train)
X_test = sc.transform(X_test)

pca = PCA(n_components= 2) # here you can change this nu
mber to play around
X_train = pca.fit_transform(X_train)
X_test = pca.transform(X_test)

# Create the classifier and train using training data
from sklearn.linear_model import LogisticRegression
classifier = LogisticRegression(random_state=0)
classifier.fit(X_train,y_train)

#Predict the test set values
y_pred = classifier.predict(X_test)

#Compute confusion matrix
from sklearn.metrics import confusion_matrix
cm = confusion_matrix(y_test,y_pred)
cm
```

Out[4]:

```
array([[82, 15],
       [23, 34]])
```

Let us try to play with the number of components

In [5]:

```
X_train,X_test,y_train,y_test = train_test_split(X,y,te
st_size=0.2)
sc = StandardScaler()
X_train = sc.fit_transform(X_train)
X_test = sc.transform(X_test)

pca = PCA(n_components= 4) #I have tried different no of
 components here
X_train = pca.fit_transform(X_train)
X_test = pca.transform(X_test)
```

```
# Create the classifier and train using training data
from sklearn.linear_model import LogisticRegression
classifier = LogisticRegression(random_state=0)
classifier.fit(X_train,y_train)

#Predict the test set values
y_pred = classifier.predict(X_test)

#Compute confusion matrix
from sklearn.metrics import confusion_matrix
cm = confusion_matrix(y_test,y_pred)
cm
```

Out[5]:

```
array([[79,  8],
       [39, 28]])
```

Logistic classification is a linear classification. Let us proceed to try some nonlinear classifiers such as Support vector machines.

In [6]:

```
X_train,X_test,y_train,y_test = train_test_split(X,y,te
st_size=0.2)
sc = StandardScaler()
X_train = sc.fit_transform(X_train)
X_test = sc.transform(X_test)

#Create classifier object
from sklearn.svm import SVC
classifier_svm_kernel = SVC(C=5.0,kernel='rbf', gamma=
0.12,tol=0.00001)
classifier_svm_kernel.fit(X_train,y_train)

#Predict the result for test values
y_pred = classifier_svm_kernel.predict(X_test)

#Compute confusion matrix
from sklearn.metrics import confusion_matrix
cm = confusion_matrix(y_test,y_pred)
cm
```

Out[6]:

```
array([[84, 12],
       [37, 21]])
```

In [7]:

```
#Comparing the predictions with the actual results
comparison = pd.DataFrame(y_test,columns=['y_test'])
comparison['y_predicted'] = y_pred
comparison.head(5)
```

Out[7]:

	y_test	y_predicted
0	1	0
1	0	0
2	0	0
3	1	1
4	0	0

Let us now try to implement a k-fold cross validation with a grid search routine to figure the best hyperparameters along with the statistics of our accuracy, precision and recall. Applying a ten-fold cross validation.

In [8]:

```
#Apply k-fold validation here
from sklearn.model_selection import cross_val_score
accuracies = cross_val_score(estimator=classifier_svm_k
ernel,X=X_train,y=y_train,cv=10)
accuracies
```

Out[8]:

```
array([ 0.74193548,  0.64516129,  0.8064516
1,  0.75806452,  0.78688525,
        0.70491803,  0.73770492,  0.7377049
2,  0.72131148,  0.83606557])
```

In [9]:

```
plt.hist(accuracies)
plt.show()
```

From the plot of accuracy histogram, we can see that the variance is less where as the accuracy is centered around 0.76. Note that we have used all the features in this SVM classifier. However, the hyperparameters were chosen at random. Let us try to pass a grid-search method to figure out optimal values for hyperparameters.

In [10]:

```
#Applying grid search for optimal parameters and model af
ter k-fold validation
from sklearn.model_selection import GridSearchCV

parameters = [{'C':[0.01,0.1,1,10,50,100,500,1000], 'ke
rnel':['rbf'], 'gamma': [0.1,0.125,0.15,0.17,0.2]}]
grid_search = GridSearchCV(estimator=classifier_svm_ker
nel, param_grid=parameters, scoring ='accuracy',cv=10,n
_jobs=-1)
grid_search = grid_search.fit(X_train,y_train)
```

In [11]:

```
best_accuracy = grid_search.best_score_
best_accuracy
```

Out[11]:

```
0.76872964169381108
```

In [12]:

```
opt_param = grid_search.best_params_
opt_param
```

Out[12]:

```
{'C': 0.1, 'gamma': 0.1, 'kernel': 'rbf'}
```

It seems, that even with the optimal parameters, the accuracy is still around 0.76 with a support vector classifier. It might be interesting to study if using a different classifier might be able to give us better accuracy.

One last thing before I end this notebook is to see whether using just few principal components can give us similar accuracy. Hence I will apply PCA and then use SVM on the few components.

In [13]:

```
#Reloading the features and labels and normalizing them
X_train,X_test,y_train,y_test = train_test_split(X,y,te
st_size=0.2)
sc = StandardScaler()
X_train = sc.fit_transform(X_train)
X_test = sc.transform(X_test)

#Choosing 2 principal components
pca = PCA(n_components= 2) # here you can change this nu
mber to play around
X_train = pca.fit_transform(X_train)
X_test = pca.transform(X_test)

#Create classifier object
classifier_svm_kernel = SVC(C=5.0,kernel='rbf', gamma=
0.12,tol=0.00001)
classifier_svm_kernel.fit(X_train,y_train)

# Grid search and k fold validation libraries already imp
orted. So start the grid search
```

Did you find this Kernel useful?
Show your appreciation with an upvote

3

Comments (0)

Please sign in to leave a comment.