February 13, 2018

Lab Assignment 2

Sanchit Singhal

INF 385T – Introduction to Machine Learning with Danna Gurari

Spring 2018

School of Information

Hyperlink to code: https://introtoml-sanchit1276.notebooks.azure.com/nb/notebooks/IntroToML/LabAssignment2.ipynb

1. Construct Datasets for Training and Evaluation

```
# Load data and split into train/test
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split

iris = load_iris()
X_train, X_test, y_train, y_test = train_test_split(iris.data, iris.target, test_size=0.2
print("Number samples in training: ", len(X_train))
print("Number samples in testing: ", len(X_test))
```

```
Number samples in training:  120
Number samples in testing:  30
```

2. Optimize Hyperparameters for Each Classification Model :

a) Decision Tree

```
# 2a - Optimize Hyperparameters for Decision Tree
```

```
import numpy as np
from sklearn.tree import DecisionTreeClassifier
from sklearn.grid_search import GridSearchCV
from operator import itemgetter

clf = DecisionTreeClassifier()

param_grid = {"max_depth": [1,2,3,4,5, None],
              "criterion": ["gini", "entropy"]}


grid_search = GridSearchCV(clf, param_grid=param_grid, cv=10)

grid_search.fit(X_train, y_train)

top_scores = sorted(grid_search.grid_scores_, key=itemgetter(1), reverse=True)
for i, score in enumerate(top_scores):
    print("Model with rank: {0}".format(i + 1))
    print("Mean validation score: {0:.3f} (std: {1:.3f})".format(
        score.mean_validation_score,
        np.std(score.cv_validation_scores)))
    print("Parameters: {0}".format(score.parameters))
    print("")
```

```
Model with rank: 1
Mean validation score: 0.958 (std: 0.043)
Parameters: {'criterion': 'gini', 'max_depth': 3}

Model with rank: 2
Mean validation score: 0.958 (std: 0.060)
Parameters: {'criterion': 'gini', 'max_depth': 5}

Model with rank: 3
Mean validation score: 0.958 (std: 0.060)
Parameters: {'criterion': 'entropy', 'max_depth': 3}
```

I tested 12 combinations of hyper parameters in total and the optimal ones were:

Criterion: Gini, Max_depth: 3

b) K-Nearest Neighbor

```
# 2b - Optimize Hyperparameters for K-Nearest Neighbors
```

```
# Normalize data
from sklearn.preprocessing import MinMaxScaler

mms = MinMaxScaler()
X_train_norm = mms.fit_transform(X_train)
X_test_norm = mms.transform(X_test)
```

```
from sklearn.neighbors import KNeighborsClassifier

clf = KNeighborsClassifier()

param_grid = {"n_neighbors": [1,2,3,4,5],
              "metric": ["euclidean", "manhattan"]}


grid_search = GridSearchCV(clf, param_grid=param_grid, cv=10)

grid_search.fit(X_train_norm, y_train)

top_scores = sorted(grid_search.grid_scores_, key=itemgetter(1), reverse=True)
for i, score in enumerate(top_scores):
    print("Model with rank: {0}".format(i + 1))
    print("Mean validation score: {0:.3f} (std: {1:.3f})".format(
        score.mean_validation_score,
        np.std(score.cv_validation_scores)))
    print("Parameters: {0}".format(score.parameters))
    print("")
```

```
Model with rank: 1
Mean validation score: 0.958 (std: 0.060)
Parameters: {'metric': 'euclidean', 'n_neighbors': 1}

Model with rank: 2
Mean validation score: 0.958 (std: 0.060)
Parameters: {'metric': 'euclidean', 'n_neighbors': 4}

Model with rank: 3
Mean validation score: 0.958 (std: 0.045)
Parameters: {'metric': 'euclidean', 'n_neighbors': 5}
```

I tested 10 combinations of hyper parameters in total and the optimal ones were:

Metric: Euclidean, n_neighbors = 1

c) Support Vector Machine

```
# 2c - Optimize Support Vector Machine
```

```
# Normalize data
from sklearn.svm import LinearSVC
from sklearn.preprocessing import StandardScaler

ss = StandardScaler()
X_train_scaled = ss.fit_transform(X_train)
X_test_scaled = ss.transform(X_test)
```

```
from sklearn.svm import SVC

clf = SVC(kernel="poly")

param_grid = {"degree": [2,3,4],
              "C": [0.01, 0.1, 1, 10, 100],
              "gamma": [0.0001, 0.001, 0.01, 0.1,1]}


grid_search = GridSearchCV(clf, param_grid=param_grid, cv=10)

grid_search.fit(X_train_scaled, y_train)

top_scores = sorted(grid_search.grid_scores_, key=itemgetter(1), reverse=True)
for i, score in enumerate(top_scores):
    print("Model with rank: {0}".format(i + 1))
    print("Mean validation score: {0:.3f} (std: {1:.3f})".format(
        score.mean_validation_score,
        np.std(score.cv_validation_scores)))
    print("Parameters: {0}".format(score.parameters))
    print("")
```

```
Model with rank: 1
Mean validation score: 0.950 (std: 0.043)
Parameters: {'C': 0.1, 'degree': 3, 'gamma': 1}

Model with rank: 2
Mean validation score: 0.950 (std: 0.043)
Parameters: {'C': 100, 'degree': 3, 'gamma': 0.1}

Model with rank: 3
Mean validation score: 0.942 (std: 0.057)
Parameters: {'C': 1, 'degree': 3, 'gamma': 1}
```

I tested 75 combinations of hyper parameters in total and the optimal were either of the two:

C: 0.1, degree: 3, gamma: 1

C: 100, degree: 3, gamma: 0.1

3. Comparative Analysis of Optimized Classification Models

a) Retrain Decision Tree, K-NN, SVM

```
# 3a
```

```
decisiontree_model = DecisionTreeClassifier(criterion = 'gini', max_depth = 3).fit(X_train, y_train)
knn_model = KNeighborsClassifier(n_neighbors=4, metric = "euclidean").fit(X_train_norm, y_train)
svm_model = SVC(kernel="poly", degree=3, C = 0.1, gamma = 1).fit(X_train_scaled, y_train)
```

b) Retrain Gaussian Naïve Bayes

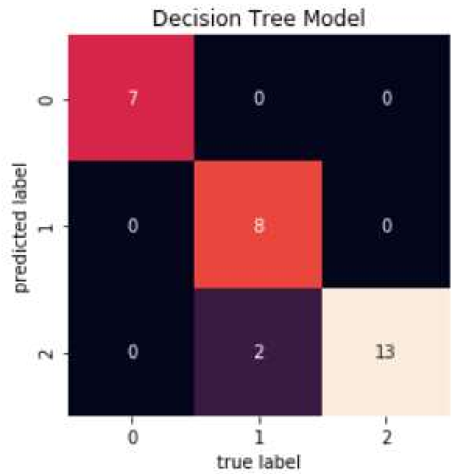
```
# 3b
```

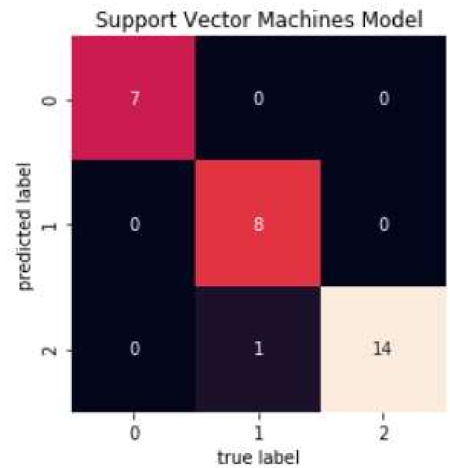
```
from sklearn.naive_bayes import GaussianNB

gaussian_model = GaussianNB().fit(X_train, y_train)
```

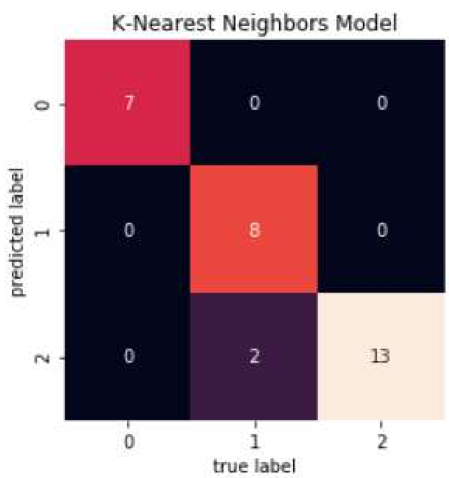
c) Predictive Performance

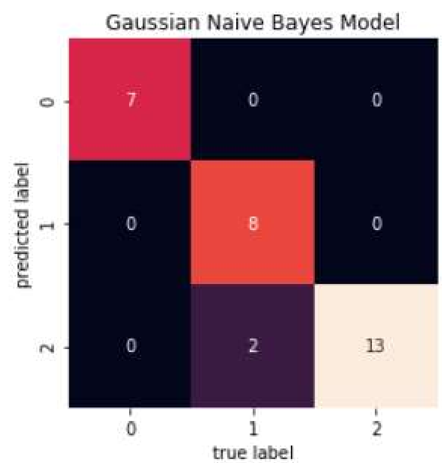
```
Decision Tree Model
Accuracy : 0.9333333333333333
             precision    recall  f1-score   support

          0       1.00      1.00      1.00         7
          1       1.00      0.80      0.89        10
          2       0.87      1.00      0.93        13

avg / total       0.94      0.93      0.93        30

K-Nearest Neighbors Model
Accuracy : 0.9333333333333333
             precision    recall  f1-score   support

          0       1.00      1.00      1.00         7
          1       1.00      0.80      0.89        10
          2       0.87      1.00      0.93        13

avg / total       0.94      0.93      0.93        30

Support Vector Machines Model
Accuracy : 0.9666666666666667
             precision    recall  f1-score   support

          0       1.00      1.00      1.00         7
          1       1.00      0.89      0.94         9
          2       0.93      1.00      0.97        14

avg / total       0.97      0.97      0.97        30

Gaussian Naive Bayes Model
Accuracy : 0.9333333333333333
             precision    recall  f1-score   support

          0       1.00      1.00      1.00         7
          1       1.00      0.80      0.89        10
          2       0.87      1.00      0.93        13

avg / total       0.94      0.93      0.93        30
```

d) Confusion Matrix









e) Analysis and Comparison of Performance

The Support Vector Machine model performed the best on the test set while the Decision Tree was the worst. KNN was very close to SVM but slightly worse in recall. Both of these were better than the Decision Tree and Naïve Bayes models – although not by much. Changing the data set split state (during reruns of the code) alters the model performance slightly – presumably by how the data is initially split up.

The performance metrics tell us that all the models were very close to perfect with almost all the points falling on the diagonal center line. There do appear to be some records that were predicted to be in the level 2 but the true label was 1. SVM model was able to reduce this error and had one less data point in this bracket.

The algorithms performed the way they did because of how they are calculated. SVM’s are usually less vulnerable to outliers whereas KNN are more susceptible to irrelevant or noisy data. It is possible that the dataset contains a few bad data elements which causes the SVM model to perform slightly better.