InClass_Lab4

February 6, 2018

In [1]:
```
# Load data and split into train/test
from sklearn.datasets import load_breast_cancer
from sklearn.model_selection import train_test_split

cancer = load_breast_cancer()
X_train, X_test, y_train, y_test = train_test_split(cancer.data, cancer.target, random_s
```

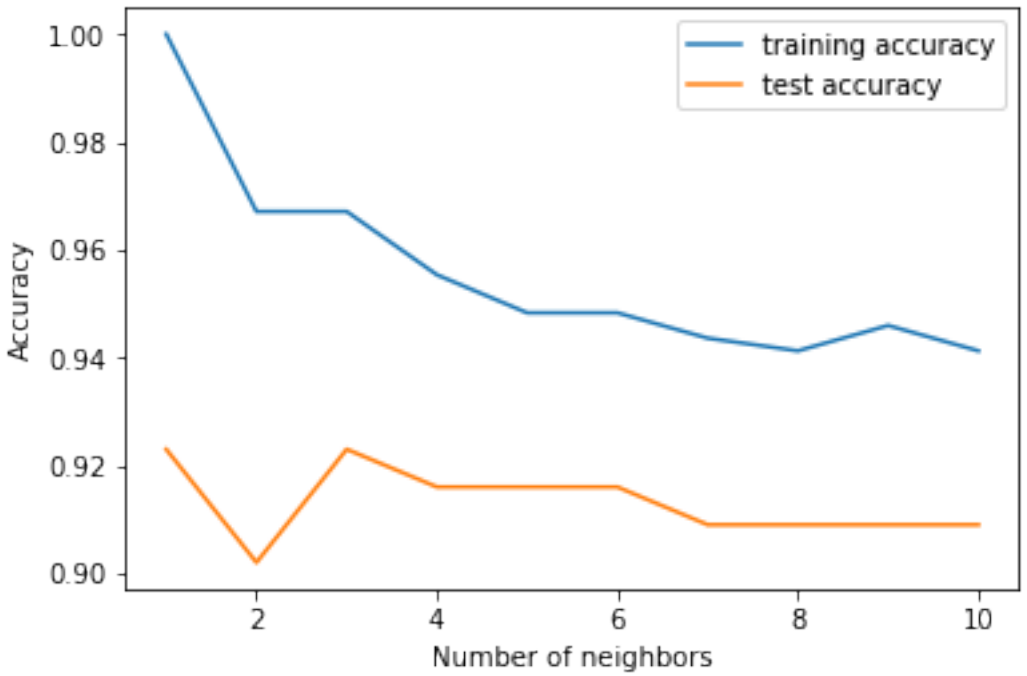
In [2]:
```
# Normalize training data and then use those
# parameters to transform the test set or any new data point
from sklearn.preprocessing import MinMaxScaler

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

In [3]:
```
# Test for n_neighbors 1 to 10

from sklearn.neighbors import KNeighborsClassifier

training_accuracy = []
test_accuracy = []
neighbor_settings = range(1, 11)
for n_neighbors in neighbor_settings:
    #build the model
    clf = KNeighborsClassifier(n_neighbors=n_neighbors)
    clf.fit(X_train, y_train)
    # record training set accuracy
    current_train_accuracy = clf.score(X_train, y_train)
    training_accuracy.append(current_train_accuracy)
    # record test set accuracy
    current_test_accuracy = clf.score(X_test, y_test)
    test_accuracy.append(current_test_accuracy)
```

In [4]:
```
# Visualize predictive accuracy when using different values for n_neighbors
import matplotlib.pyplot as plt

%matplotlib inline
```

```
        plt.plot(neighbor_settings, training_accuracy, label="training accuracy")
        plt.plot(neighbor_settings, test_accuracy, label="test accuracy")
        plt.ylabel("Accuracy")
        plt.xlabel("Number of neighbors")
        plt.legend()
```

Out[4]: <matplotlib.legend.Legend at 0x7f2607eedeb8>



In [5]:
```
        # See documentation for what are default values and how to change them
        # http://scikit-learn.org/stable/modules/generated/sklearn.neighbors.KNeighborsClassifie
        # Use Manhattan Distance instead.
        knn = KNeighborsClassifier(n_neighbors=5, p=1, metric='minkowski')
        knn.fit(X_train, y_train)
        manhattan_accuracy = clf.score(X_test, y_test)
        manhattan_accuracy
```

Out[5]: 0.90909090909090906

In [6]:
```
        # Now train a Linear Support Vector Machine (SVM)
        # http://scikit-learn.org/stable/modules/generated/sklearn.svm.LinearSVC.html#sklearn.sv
        from sklearn.svm import LinearSVC
        from sklearn.preprocessing import StandardScaler

        # First, rescale training data and then use those
        # parameters to transform the test set or any new data point
```

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

        # Now train and evaluate the SVM
        svm_clf = LinearSVC()
        svm_clf.fit(X_train_scaled, y_train)
        current_test_accuracy = svm_clf.score(X_test_scaled, y_test)
        current_test_accuracy
```

```
Out[6]: 0.97202797202797198
```

```
In [7]: # Now train a SVM with polynomial features
        from sklearn.svm import SVC

        poly_kernel_svm_clf = SVC(kernel="poly", degree=3)
        poly_kernel_svm_clf.fit(X_train_scaled, y_train)
        current_test_accuracy = poly_kernel_svm_clf.score(X_test_scaled, y_test)
        current_test_accuracy
```

```
Out[7]: 0.93006993006993011
```

```
In [8]: # Gaussian RBF Kernel
        # gamma acts a regularization parameter: increasing it makes bell-shape
        # curver more narrow and so each instance's influence smaller; decision
        # boundary ends up more irregular, wiggling around individual instances
        rbf_kernel_svm_clf = SVC(kernel="rbf", gamma=5, C=0.001)
        rbf_kernel_svm_clf.fit(X_train_scaled, y_train)
        current_test_accuracy = rbf_kernel_svm_clf.score(X_test_scaled, y_test)
        current_test_accuracy
```

```
Out[8]: 0.64335664335664333
```

```
In [9]: from sklearn.datasets import load_iris
        from sklearn.linear_model import LogisticRegression

        # Load data
        iris = load_iris()
        X = iris.data
        y = iris.target

        # Instantiate classifier
        logreg = LogisticRegression()
```

```
In [10]: # Perform three-fold cross-validation
         from sklearn.model_selection import cross_val_score
         from sklearn.model_selection import KFold

         kfold = KFold(n_splits=3)
```

```
        fold_accuracies = cross_val_score(logreg, X, y, cv=kfold)

        print("Cross-validation scores:\n{}".format(fold_accuracies))

        # Summarize cross-validation accuracy by computing the mean
        print("Average cross-validation score: {:.2f}".format(fold_accuracies.mean()))
```

```
Cross-validation scores:
[ 0.  0.  0.]
Average cross-validation score: 0.00
```

```
In [11]: # Why do you think accuracy is 0?
         iris.target
```

```
Out[11]: array([0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
                0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
                0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
                1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
                1, 1, 1, 1, 1, 1, 1, 1, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,
                2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,
                2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2])
```

```
In [12]: # Perform three-fold cross-validation after shuffling data
         kfold = KFold(n_splits=3, shuffle=True, random_state=0)
         fold_accuracies = cross_val_score(logreg, X, y, cv=kfold)

         print("Cross-validation scores:\n{}".format(fold_accuracies))
         print("Average cross-validation score: {:.2f}".format(fold_accuracies.mean()))
```

```
Cross-validation scores:
[ 0.9   0.96  0.96]
Average cross-validation score: 0.94
```

```
In [13]: # Evaluate by using cross validation: by default, performs
         # stratified 3-fold cross-validation and returns three accuracy values
         # http://scikit-learn.org/stable/modules/generated/sklearn.model_selection.cross_val_sc
         fold_accuracies = cross_val_score(logreg, X, y)
         print("Cross-validation scores:\n{}".format(fold_accuracies))
         print("Average cross-validation score: {:.2f}".format(fold_accuracies.mean()))
```

```
Cross-validation scores:
[ 0.96078431  0.92156863  0.95833333]
Average cross-validation score: 0.95
```

```
In [14]: # Leave-one-out cross-validation: good approach only for small datasets
         from sklearn.model_selection import LeaveOneOut
```

```
loo = LeaveOneOut()
scores = cross_val_score(logreg, X, y, cv=loo)
print("Number of cv iterations: ", len(scores))
print("Mean accuracy: {:.2f}".format(scores.mean()))
```

```
Number of cv iterations:  150
Mean accuracy: 0.95
```

In [15]:
```
# Tuning hyperparameters for SVM
# Split data into train+validation set and test set
X_trainval, X_test, y_trainval, y_test = train_test_split(iris.data, iris.target, rando
print("Size of training+validation set: {} size of test set: {}".format(X_train.shape[0

# Split train+validation set into training and validation sets
X_train, X_val, y_train, y_val = train_test_split(X_trainval, y_trainval, random_state=
print("Size of training set: {} size of validation set: {} size of test set: {}".format

best_score = 0
for curGamma in [0.001, 0.01, 0.1, 1, 10, 100]:
    for curC in [0.001, 0.01, 0.1, 1, 10, 100]:
        # for each combination of parameters, train an SVC
        svm = SVC(gamma=curGamma, C=curC)
        svm.fit(X_train, y_train)

        # evaluate the SVC on the test set
        score = svm.score(X_val, y_val)

        # if we got a better score, store the score and parameters
        if score > best_score:
            best_score = score
            best_parameters = {'C': curC, 'gamma': curGamma}

# Rebuild a model on the combined training and validation set,
# and evaluate it on the test set
svm = SVC(**best_parameters)
svm.fit(X_trainval, y_trainval)
test_score = svm.score(X_test, y_test)
print("Best score on validation set: {:.2f}".format(best_score))
print("Best parameters: {}".format(best_parameters))
print("Test set score with best parameters: {:.2f}".format(test_score))
```

```
Size of training+validation set: 426 size of test set: 38
Size of training set: 84 size of validation set: 28 size of test set: 38
Best score on validation set: 0.96
Best parameters: {'C': 10, 'gamma': 0.001}
Test set score with best parameters: 0.92
```

```
In [16]: best_score = 0
         for curGamma in [0.001, 0.01, 0.1, 1, 10, 100]:
             for curC in [0.001, 0.01, 0.1, 1, 10, 100]:
                 # for each combination of parameters, train an SVC
                 svm = SVC(gamma=curGamma, C=curC)

                 # perform cross-validation
                 fold_accuracies = cross_val_score(svm, X_trainval, y_trainval)

                 # evaluate the SVC on the test set
                 score = fold_accuracies.mean()

                 # if we got a better score, store the score and parameters
                 if score > best_score:
                     best_score = score
                     best_parameters = {'C': curC, 'gamma': curGamma}

         # Rebuild a model on the combined training and validation set,
         # and evaluate it on the test set
         svm = SVC(**best_parameters)
         svm.fit(X_trainval, y_trainval)
         test_score = svm.score(X_test, y_test)
         print("Best score on validation set: {:.2f}".format(best_score))
         print("Best parameters: {}".format(best_parameters))
         print("Test set score with best parameters: {:.2f}".format(test_score))
```

```
Best score on validation set: 0.96
Best parameters: {'C': 100, 'gamma': 0.01}
Test set score with best parameters: 0.97
```