Feature Selection: Nearest Neighbor CS 205: Artificial Intelligence

Alexander Nguyen Instructor: Dr. Eamonn Keogh
861005564
anguy086@ucr.edu
March 19, 2018

In completing this project, I consulted:

- The Machine Learning slides from lecture
- The following site for importing the data and using data frames. It also provided nice tutorials to go about certain ideas
    - https://pandas.pydata.org/
- Min-max normalization
    - https://stackoverflow.com/questions/26414913/normalize-columns-of-pandas-data-frame/29651514

All important code is original. Unimportant subroutines that are not completely original include:

- The pandas module and usage of the read_csv function to import the data sets as well as splitting the dataset

CS 205 Feature Selection Write-Up

Introduction

The goal of this project was to derive relevant features from datasets containing mostly irrelevant and useless features. This closely relates to practical settings because irrelevant data is everywhere. There are two provided datasets (a small one with a class label, 10 features, and 100 data points and a large one with a class label, 50 features, and 100 data points). Both of the datasets consist of continuous data and a class label of '1' or '2', but before we can run the algorithms, the data needs to be normalized. The min-max normalization method was used here to bound every point between 0 and 1.

In order to decide which features are actually helpful, three different kinds of search algorithms were implemented: Forward selection, Backwards Elimination, and an original search algorithm. Furthermore, nearest neighbor is the classifier used to determine the class label of a data point. The accuracy is determined using cross validation. After all of these techniques are used, we will then discover which features actually have value in the dataset. The reason behind using nearest neighbor for this is that nearest neighbor is very sensitive to irrelevant features. So, we can take note of whether a feature is good or not by seeing how accurately nearest neighbor would classify it.

For this project, I will be using the language Python3 with small dataset 19 and big dataset 2. My original algorithm will attempt to make the program run faster through the use of pruning.

Nearest Neighbor

The nearest neighbor classifier is a model that takes one data point and compares it position in a field with all of the other data points. Depending on the number of nearest neighbors (denoted as k), the classifier will give the new data point a certain label. In this project, I am using a value of 1 for k. So the new data point's label will be the same as the label of its closest neighbor.

Cross-Validation

Cross-validation is a technique used to evaluate models. It's generally referred to as k-fold cross-validation. Here, the k value that I'm using is 1. This is a special method of cross-validation known as "leave-one-out" which has good accuracy. Basically, one data point will be separated from the main dataset. This one data point will be the test set and the remaining points will be the training set. Since the training set is roughly the size of the original dataset, the accuracy is really good. The trade-off is that this reduces efficiency and cripples the runtime. After some model, such as nearest neighbor in this project, is used between the test and training set, we store the accuracy by comparing the label to the actual label and then we adjust both of the sets. The test

set will now only contain the second data point and the training set will again be the remaining points. This will repeat until every point has entered the test set, where the number of iterations will equal the number of points.

Forward Selection

Forward Selection is a greedy search algorithm whose initial state is zero nodes. The search then uses cross validation and nearest neighbor to check the accuracies generated when adding each of the features individually. The feature with the best accuracy is added to a list and that node with the feature is expanded. The second stage then compares this feature plus every other distinct feature. Again, the new feature, when combined with our good feature, that produced the highest accuracy is added to the list. The third stage then compares these two features with a third feature and the process repeats. Here, I used a bitmap that represents each feature. Whenever a feature is used, I assign the feature with a weight of 1. If it's not used, I assign it with 0. This was helpful because I didn't have to type each feature out in a long statement.

If there are irrelevant features, we would expect to see the accuracy between adding features to eventually decrease. We don't end the search right then and there because our search may be at a local optimum, whereas we want to find the global optimum. When the search is completed, we note the features that generated the largest accuracy. In this project, two of the features are strong, one is weak, and the remaining are irrelevant. So here, the ideal case would be the algorithm generating only 2-3 good features.

Backwards Elimination

Backwards elimination is very similar to forward selection in that it's a greedy algorithm that searches for the best combination of nodes. However, this method begins with all of the nodes in its current list. Then, one node is removed at a time and the accuracy is checked. If a removed node resulted in the largest attained accuracy, we will know that feature is most likely irrelevant. This method continues until the search is over. Just like forward selection, we will have a best-so-far accuracy to check which features are relevant and only return the features that achieved this.

Original Search: Pruning

My original search algorithm is going to attempt to be faster. Currently, the project specifications state that the search on the large dataset should take roughly a minute based on the computer language used. My algorithm takes much longer than a minute (even for the small dataset), so this is why I chose to make it faster. The idea behind speeding the program up comes from alpha-beta pruning. We should not bother fully expanding a node that is going to be considerably worse than any node we've seen thus far. For example, suppose one feature is providing 90 correct labels out of 100. If another feature produced 11 wrong labels, then we

should stop trying to find the accuracy of this feature even if the cross-validation process is not done because we only want the highest accuracy. This should speed up the algorithm drastically.

In order to do this, I made some modifications to the forward selection algorithm. Mainly, I just added a parameter to the cross-validation function stating the highest amount of incorrect labels seen for that iteration so far. Then I added an if-statement inside the function to stop if the number of incorrect labels exceeds that of our best-so-far feature. Since we are looking for the highest accuracy, we'll just return 0 immediately to be safe. Then in the search function, I just updated the number of incorrect labels every time a feature with a higher accuracy is found. This abrupt end to the feature search will make the program run faster.

Results and Conclusion

For the small dataset, forward selection yielded features 9, 1, and 4 as the relevant features. Feature 4 appears to be the weak feature because it did not increase the accuracy when it was added. When more than three features were added, the accuracy began to drop, as shown in Figure 1. The pruning algorithm got the same result but it was faster than this one because it didn't fully complete each iteration of searches. However, it strangely was not as much faster as I anticipated. It mainly saved around five minutes.

For the big dataset, forward selection and my algorithm were not running very quickly. So far, it took about 6 hours to find 7 features, which are 1, 7, 48, 21, 2, 19, and 43 so far. Also, during the choosing of the second node, features 7 and 24 both had an accuracy of 94% so although feature 24 never showed up in the selected nodes by the algorithm, there was a chance that it could have been there. The algorithm was taking too long so I stopped it here at 7 features. However, in Figure 2, there is a noticeable jump in accuracy between the addition of features 1 and 7, so these two may make good candidates for features. But overall, with more data instances, the accuracy of the features can be made more pronounced.

Backwards Elimination returns a weaker subset of features. Feature 4 got eliminated and the best subset of features were 1, 4, 6, 7, 9, 10 for the smaller dataset with an accuracy of 89%. This has the strong features but contains too many spurious features.

In conclusion, it was shown that pruning the sets of features was beneficial in improving performance. Also, the nearest neighbor algorithm was fairly successful in gathering the best features out of a dataset. Some spurious features were grabbed in the big dataset, but it mostly grabbed the strong features, which is a good sign of completeness. The only crucial drawback is the runtime. Therefore, we showed that it's possible to discover which features are irrelevant by running a classifier that is sensitive to irrelevant features and gaining knowledge from there.

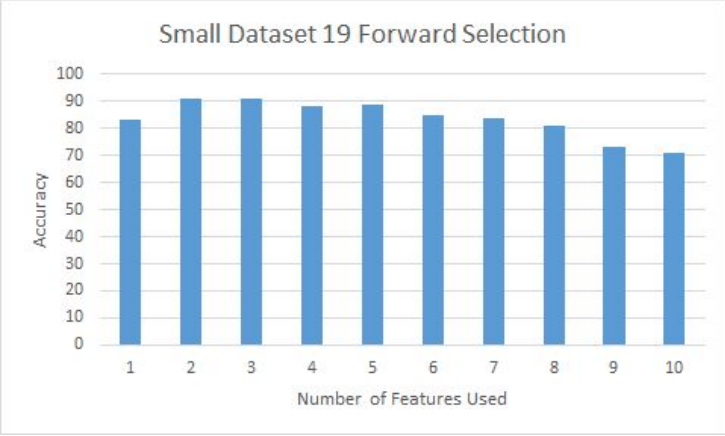
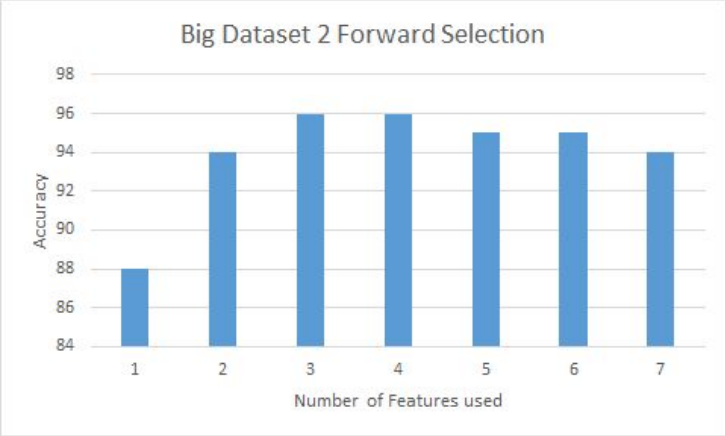


Figure 1: Accuracy of small dataset with different numbers of features



Feature 2: Accuracy of big dataset with different numbers of features

Trace

Below is a trace on the small dataset I was provided (dataset number 19). After three features, subsequent accuracies dropped. When the search completed, the highest accuracy was attained by using features 9, 1, and 4. Please disregard the percent signs at the end of each accuracy.

```
Welcome to Alex Nguyen's Feature Selection Algorithm.
Type in the name of the file to test: CS205_SMALLtestdata__19.txt

Type in the name of the algorithm you want to run.

    1) Forward Selection
    2) Backwards Elimination
    3) Alex's Special Algorithm

            1

This dataset has 10 features (not including the class attribute), with 100 instances

Please wait while I normalize the data...   Done!

Beginning Search.
    Using feature(s) {1}, accuracy is 0.740000%
    Using feature(s) {2}, accuracy is 0.610000%
    Using feature(s) {3}, accuracy is 0.670000%
    Using feature(s) {4}, accuracy is 0.680000%
    Using feature(s) {5}, accuracy is 0.710000%
    Using feature(s) {6}, accuracy is 0.630000%
    Using feature(s) {7}, accuracy is 0.550000%
    Using feature(s) {8}, accuracy is 0.610000%
    Using feature(s) {9}, accuracy is 0.830000%
    Using feature(s) {10}, accuracy is 0.660000%
Added feature 9 to the current set
Feature set {9} was best. Accuracy was 0.830000%

    Using feature(s) {1, 9}, accuracy is 0.910000%
    Using feature(s) {2, 9}, accuracy is 0.890000%
    Using feature(s) {3, 9}, accuracy is 0.780000%
    Using feature(s) {4, 9}, accuracy is 0.830000%
    Using feature(s) {5, 9}, accuracy is 0.760000%
    Using feature(s) {6, 9}, accuracy is 0.770000%
    Using feature(s) {7, 9}, accuracy is 0.800000%
    Using feature(s) {8, 9}, accuracy is 0.860000%
    Using feature(s) {10, 9}, accuracy is 0.800000%
Added feature 1 to the current set
```

```
Feature set {9, 1} was best. Accuracy was 0.910000%

    Using feature(s) {2, 9, 1}, accuracy is 0.900000%
    Using feature(s) {3, 9, 1}, accuracy is 0.880000%
    Using feature(s) {4, 9, 1}, accuracy is 0.910000%
    Using feature(s) {5, 9, 1}, accuracy is 0.900000%
    Using feature(s) {6, 9, 1}, accuracy is 0.840000%
    Using feature(s) {7, 9, 1}, accuracy is 0.840000%
    Using feature(s) {8, 9, 1}, accuracy is 0.880000%
    Using feature(s) {10, 9, 1}, accuracy is 0.880000%
Added feature 4 to the current set
Feature set {9, 1, 4} was best. Accuracy was 0.910000%

    Using feature(s) {2, 9, 1, 4}, accuracy is 0.850000%
    Using feature(s) {3, 9, 1, 4}, accuracy is 0.880000%
    Using feature(s) {5, 9, 1, 4}, accuracy is 0.880000%
    Using feature(s) {6, 9, 1, 4}, accuracy is 0.850000%
    Using feature(s) {7, 9, 1, 4}, accuracy is 0.840000%
    Using feature(s) {8, 9, 1, 4}, accuracy is 0.840000%
    Using feature(s) {10, 9, 1, 4}, accuracy is 0.820000%
Added feature 3 to the current set
Feature set {9, 1, 4, 3} was best. Accuracy was 0.880000%
(WARNING!) Accuracy has decreased! Continue search in case of local maxima)

    Using feature(s) {2, 9, 1, 4, 3}, accuracy is 0.810000%
    Using feature(s) {5, 9, 1, 4, 3}, accuracy is 0.890000%
    Using feature(s) {6, 9, 1, 4, 3}, accuracy is 0.840000%
    Using feature(s) {7, 9, 1, 4, 3}, accuracy is 0.820000%
    Using feature(s) {8, 9, 1, 4, 3}, accuracy is 0.770000%
    Using feature(s) {10, 9, 1, 4, 3}, accuracy is 0.760000%
Added feature 5 to the current set
Feature set {9, 1, 4, 3, 5} was best. Accuracy was 0.890000%
(WARNING!) Accuracy has decreased! Continue search in case of local maxima)

    Using feature(s) {2, 9, 1, 4, 3, 5}, accuracy is 0.850000%
    Using feature(s) {6, 9, 1, 4, 3, 5}, accuracy is 0.840000%
    Using feature(s) {7, 9, 1, 4, 3, 5}, accuracy is 0.780000%
    Using feature(s) {8, 9, 1, 4, 3, 5}, accuracy is 0.820000%
    Using feature(s) {10, 9, 1, 4, 3, 5}, accuracy is 0.780000%
Added feature 2 to the current set
```

```
Feature set {9, 1, 4, 3, 5, 2} was best. Accuracy was 0.850000%
(WARNING!) Accuracy has decreased! Continue search in case of local maxima)

    Using feature(s) {6, 9, 1, 4, 3, 5, 2}, accuracy is 0.840000%
    Using feature(s) {7, 9, 1, 4, 3, 5, 2}, accuracy is 0.810000%
    Using feature(s) {8, 9, 1, 4, 3, 5, 2}, accuracy is 0.740000%
    Using feature(s) {10, 9, 1, 4, 3, 5, 2}, accuracy is 0.790000%
Added feature 6 to the current set
Feature set {9, 1, 4, 3, 5, 2, 6} was best. Accuracy was 0.840000%
(WARNING!) Accuracy has decreased! Continue search in case of local maxima)

    Using feature(s) {7, 9, 1, 4, 3, 5, 2, 6}, accuracy is 0.810000%
    Using feature(s) {8, 9, 1, 4, 3, 5, 2, 6}, accuracy is 0.670000%
    Using feature(s) {10, 9, 1, 4, 3, 5, 2, 6}, accuracy is 0.750000%
Added feature 7 to the current set
Feature set {9, 1, 4, 3, 5, 2, 6, 7} was best. Accuracy was 0.810000%
(WARNING!) Accuracy has decreased! Continue search in case of local maxima)

    Using feature(s) {8, 9, 1, 4, 3, 5, 2, 6, 7}, accuracy is 0.730000%
    Using feature(s) {10, 9, 1, 4, 3, 5, 2, 6, 7}, accuracy is 0.760000%
Added feature 10 to the current set
Feature set {9, 1, 4, 3, 5, 2, 6, 7, 10} was best. Accuracy was 0.760000%
(WARNING!) Accuracy has decreased! Continue search in case of local maxima)

    Using feature(s) {8, 9, 1, 4, 3, 5, 2, 6, 7, 10}, accuracy is 0.710000%
Added feature 8 to the current set
Feature set {9, 1, 4, 3, 5, 2, 6, 7, 10, 8} was best. Accuracy was 0.710000%
(WARNING!) Accuracy has decreased! Continue search in case of local maxima)

Finished Search!! The best feature subset is {9, 1, 4}, which has an accuracy of 0.910000%

Process finished with exit code 0
```

Code Print-out

Main Function:

- Welcome Message
- Use pandas to read in the data files with the column names
- Prompt the user for the choice of algorithm
- Normalize the data using Min-Max Normalization

```
def main():
    print('Welcome to Alex Nguyen\'s Feature Selection Algorithm.')
    fileName = input('Type in the name of the file to test: ')
    if fileName[6:9] == 'BIG':
        df = pd.read_csv(fileName, sep='\s+', names=['Class Label', '1', '2', '3', '4', '5', '6', '7', '8', '9', '10',
            '11', '12', '13', '14', '15', '16', '17', '18', '19', '20', '21', '22', '23', '24', '25', '26', '27', '28',
            '29','30', '31', '32', '33', '34', '35', '36', '37', '38', '39', '40','41', '42', '43', '44', '45', '46',
            '47', '48','49', '50'])
    else:
        # \s is space. \s+ is at least one space. Small dataset has 10 features
        df = pd.read_csv(fileName, sep='\s+', names=['Class Label', '1', '2', '3', '4','5', '6', '7', '8', '9', '10'])
    print('\nType in the name of the algorithm you want to run.\n')
    print('\t1) Forward Selection\n\t2) Backwards Elimination\n\t3) Alex\'s Special Algorithm\n')
    algo_choice = input('\t\t\t')
    print('\nThis dataset has %d features (not including the class attribute), with %d instances' % (df.shape[1]-1, df.shape[0]))
    print('\nPlease wait while I normalize the data...', end='')
    #   Min-Max Normalization
    new_df = (df - df.min()) / (df.max() - df.min())
    new_df['Class Label'] = df['Class Label']   # Don't change the class labels
    print('\tDone!')
    if algo_choice == '1':
        feature_search(new_df,1)
    elif algo_choice == '2':
        backwards_elimination(new_df)
    elif algo_choice == '3':
        feature_search(new_df,3)
```

Nearest Neighbor

- Inputs are test set, training set, feature column names, bitmap for simplicity, and test row to indicate that a point cannot choose itself as a nearest neighbor
- Use Euclidean distance to find the nearest neighbor, and then outputs neighbor's label

```
def nearest_neighbor(test, train, feature, bitmap, test_row):
    bsf_dist = np.inf        #   Best-so-far Distance
    classs = 0               #   The label's class
    for i in range (0, train.shape[0]+1):
        dist = 0
        if i != test_row:
            for j in range(0, train.shape[1]-1):
                dist = dist + bitmap[0][j] * ((float(test[feature[j+1]]) -
                    float(train.loc[i:i][feature[j+1]]))**2)
            dist = np.sqrt(dist)
            if (dist < bsf_dist):
                bsf_dist = dist
                classs = train.loc[i:i][feature[0]]
    return classs.item()
```

Cross-Validation

- Cross validation was very similar for all three searches, so I made a template that shares all the features
- The second cross-validation function below is used for forward selection and backwards elimination
- The last one is used for my original search algorithm

```
def leave_one_out_cross_validation_template(dataset, current_set, feature_to_add, numIncorrect, algo_choice):
    bitmap = np.zeros((1, dataset.shape[1] - 1), dtype=float)
    features = dataset.columns
    for a in range(0, len(current_set)):
        bitmap[0][int(current_set[a]) - 1] = 1  # Make all current features 1's
    bitmap[0][int(feature_to_add) - 1] = 1  # Make the considered feature a 1
    if algo_choice == 1 or 2:
        return leave_one_out_cross_validation(dataset, bitmap, features)
    elif algo_choice == 3:
        return leave_one_out_cross_validation_special(dataset, numIncorrect, bitmap, features)
    else:
        return -1           # This should never be reached

def leave_one_out_cross_validation(dataset, bitmap, features):
    num_correct = 0
    for i in range(0, dataset.shape[0]):  # in range(0, 100)     0->99
        test_set = dataset.loc[i:i]
        df1 = dataset.loc[0:(i - 1)]
        df2 = dataset.loc[i + 1:dataset.shape[0]]  # Should be dataset.loc[i+1, 100]
        training_set = pd.concat([df1, df2])
        if nearest_neighbor(test_set, training_set, features, bitmap, i) \
                == test_set[features[0]].item():
            num_correct += 1
    return (num_correct / dataset.shape[0])

def leave_one_out_cross_validation_special(dataset, numIncorrect, bitmap, features):
    num_correct = 0
    num_wrong = 0
    for i in range(0, dataset.shape[0]):     # in range(0, 100)     0->99
        test_set = dataset.loc[i:i]
        df1 = dataset.loc[0:(i-1)]
        df2 = dataset.loc[i+1:dataset.shape[0]]        # Should be dataset.loc[i+1, 1
        training_set = pd.concat([df1,df2])
        if nearest_neighbor(test_set, training_set, features, bitmap, i)\
                == test_set[features[0]].item():
            num_correct += 1
        else:
            num_wrong += 1
        if num_wrong > numIncorrect:                #   Pruning step
            return 0
    return (num_correct / dataset.shape[0])
```

Forward Selection and Original Search

- Loops through features and calls cross validation function for the accuracy, which calls nearest neighbor
- Store the highest seen accuracy in absolute_bsf and the features in bsf_set

```
def feature_search(data, algo_choice):
    current_features = {}    #   Dictionaries are easier to search for membership
    current_features_list = []
    bsf_set = []
    absolute_bsf = 0         # If something falls below this, stop

    print('\nBeginning Search.')
    #   Iterating through the rows
    for i in range(0, data.shape[1]-1):      # should be 0 to 10
        #print('On level %d of search tree' % (i+1))
        best_so_far_accuracy = 0
        num_Incorrect = data.shape[0]              # On first try, we should not prune
        #   Iterating through the columns
        for j in range(0, data.shape[1] - 1):
            if not (j+1) in current_features:        #   Using the dictionary
                #print('-- Considering adding feature %d\n' % (j+1))
                accuracy = leave_one_out_cross_validation_template(data, current_features_list, j + 1, num_Incorrect, algo_choice)
                print('\tUsing feature(s) {%d' % (j+1), end='')
                for bb in range(0, len(current_features_list)):
                    print(', %d' % current_features_list[bb], end='')
                print('}, accuracy is %f%%' % (accuracy * 100))
                if accuracy > best_so_far_accuracy:
                    best_so_far_accuracy = accuracy
                    feature_to_add = j + 1
                    num_Incorrect = data.shape[0] - accuracy*data.shape[0]      #   Added
        current_features[feature_to_add] = ''
        current_features_list.append(feature_to_add)
        print('Added feature %d to the current set' % feature_to_add)
        print('Feature set {', end='')
        for bc in range(0, len(current_features_list)):
            print('%d' % current_features_list[bc], end = '')
            if (bc != len(current_features_list) - 1):
                print(', ', end='')
        print('} was best. Accuracy was %f%%' % (best_so_far_accuracy * 100))
```

backwards_elimi... › for i in range(...

Backwards Elimination

- Similar to forward selection but starts at a full current_features_list
- Removes them one at a time through a for loop and then calls cross-validation
- Returns feature subset with highest accuracy

```
def backwards_elimination(data):
    removed_features = {}
    removed_features_list = []
    bsf_set = []
    absolute_bsf_accuracy = 0
    current_features_list = []
    for count in range(0, data.shape[1] - 1):
        current_features_list.append(count + 1)
    #   Don't care about feature to add. Just pick a random one inside the current features so bitmap works nicely
    print('\nBeginning search.')
    default_acc = leave_one_out_cross_validation_template(data, current_features_list,current_features_list[0], 0, 2)
    absolute_bsf_accuracy = default_acc
    for i in range(0, data.shape[1] - 2):        # Ran once above
        best_so_far_acc = 0
        for j in range(0, data.shape[1]-1):
            current_features_list = []
            if not (j+1) in removed_features:
                print('-- Considering removing feature %d' % (j+1))
                for k in range(0, data.shape[1] - 1):
                    if (k+1) not in removed_features and k != j:
                        current_features_list.append(k+1)            # Add everything that is not removed and is not being remove
                accuracy = leave_one_out_cross_validation_template(data, current_features_list, current_features_list[0], 0, 2)
                print('\tUsing feature(s) {%d' % current_features_list[0], end='')
                for bb in range(1, len(current_features_list)):
                    print(', %d' % current_features_list[bb], end='')
                print('}, accuracy is %f%%' % (accuracy * 100))
                if accuracy > best_so_far_acc:
                    best_so_far_acc = accuracy
                    feature_to_remove = j + 1
        removed_features_list.append(feature_to_remove)
        removed_features[feature_to_remove] = ''
        print('Removed feature %d and got an accuracy of %f%%' % (feature_to_remove, best_so_far_acc*100))
        if best_so_far_acc >= absolute_bsf_accuracy:
            absolute_bsf_accuracy = best_so_far_acc
```