Room Occupancy Detection

Muhammet Abdullah Soyturk

January 10, 2020

This report includes some code snippets from the project. To see the full code of the project check https://github.com/mabdullahsoyturk/Occupancy-Detection

1. Download the dataset(s) for your project. If a train set and a test set are not already available, randomly split the dataset into a train and a test set using stratified sampling so that 80% of the samples go to train set and 20% to test set.

Data was retrieved from: https://archive.ics.uci.edu/ml/datasets/Occupancy+Detection+

It consists of one train and two test sets.

2. Randomly split your train set into a validation and a new train set (called train set 2) such that the validation set contains 1/5 of the samples in original train set and the train set 2 contains the remaining. Use stratified sampling to assign features to train set 2 and validation set. This should ensure that your validation set contains samples from both classes (i.e. ciliary and non-ciliary with equal proportions)

[2]:
```
train_data = pd.read_csv("datatraining.csv")

y_train = train_data['Occupancy']
X_train = train_data.loc[:, train_data.columns != 'Occupancy']

X_train_two, X_validation, y_train_two, y_validation = train_test_split(X_train,␣
 ↪y_train, test_size=0.20, stratify=y_train, random_state=42)

test_data = pd.read_csv("datatest.csv") # My dataset has two test sets. This is␣
 ↪test1.

y_test = test_data['Occupancy']
X_test = test_data.loc[:, test_data.columns != 'Occupancy']

test_data2 = pd.read_csv("datatest2.csv") # My dataset has two test sets. This␣
 ↪is test1.

y_test2 = test_data2['Occupancy']
X_test2 = test_data2.loc[:, test_data2.columns != 'Occupancy']
```

3. Normalize features in your train set 2 and validation set using min-max scaling to interval [0,1]. For this purpose you can first normalize features in your train set 2 and use the same scaling coefficients to normalize validation set. Save the normalized versions as separate files. Repeat normalizing your original train set and use the same normalization coefficients to normalize the two test sets.

[3]:
```
scaler = MinMaxScaler()

scaler.fit(X_train_two)
normalized_x_train_two = scaler.transform(X_train_two)
normalized_x_validation_train_two = scaler.transform(X_validation)

np.savetxt("normalized_x_train_two.csv", normalized_x_train_two, delimiter=",")
np.savetxt("normalized_x_validation.csv", normalized_x_validation_train_two,␣
 ↪delimiter=",")

scaler.fit(X_train)
normalized_x_train = scaler.transform(X_train)
normalized_x_test = scaler.transform(X_test)
normalized_x_test2 = scaler.transform(X_test2)
normalized_x_validation_with_orig = scaler.transform(X_validation)
```

4. Perform a 10-fold cross-validation experiment for the random forest classifier on normalized and unnormalized versions of train set 2. You can set the number of trees to 100. Do you get better accuracy when you perform data normalization?

[4]:
```
clf = RandomForestClassifier(n_estimators=100, random_state=42)
kfold = StratifiedKFold(n_splits=10, random_state=42)

unnormalized_accuracy = cross_val_score(clf, X_train_two, y_train_two, cv=kfold).
 ↪mean()
normalized_accuracy = cross_val_score(clf, normalized_x_train_two, y_train_two,␣
 ↪cv=kfold).mean()

if unnormalized_accuracy >= normalized_accuracy:
    print("No, I did not. Unnormalized Accuracy: {}, Normalized Accuracy: {}"
          .format(unnormalized_accuracy,normalized_accuracy))
else:
    print("Yes, I did. Unnormalized Accuracy: {}, Normalized Accuracy: {}"
          .format(unnormalized_accuracy,normalized_accuracy))
```

```
Yes, I did. Unnormalized Accuracy: 0.99416494608, Normalized Accuracy:
0.994471930145
```

5. Perform a 10-fold cross-validation experiment on train set 2 that corresponds to the best performing normalization strategy (i.e. normalized or unnormalized) for the following classifiers:

Logistic regression

k-nearest neighbor (with k=1)

Naïve Bayes

Decision tree

Random forest (number of trees=100)

SVM (RBF kernel C=1.0 gamma=0.125)

Linear Discriminant Analysis (This model is used instead of RBF Network)

Adaboost (number of iterations=10)

You can use default values for other hyper-parameters of the classifiers Report the following accuracy measures for each of these classifiers: overall accuracy, F-measure, sensitivity, specificity, precision, area under the ROC curve, area under the precision recall curve, MCC scores. These will be cross-validation accuracies.

[5]:
```
models = [
    ("Logistic Regression",LogisticRegression(random_state=42)),
    ("K-Nearest Neighbour",KNeighborsClassifier(n_neighbors=1)),
    ("Naive Bayes",GaussianNB()),
    ("Decision Tree",DecisionTreeClassifier(random_state=42)),
    ("Random Forest",RandomForestClassifier(n_estimators=100, random_state=42)),
    ("Support Vector Machine",SVC(kernel="rbf", C=1,gamma=0.125,␣
 ↪random_state=42)),
    ("Linear Discriminant Analysis", LinearDiscriminantAnalysis()),
    ("AdaBoostClassifier",AdaBoostClassifier(n_estimators=10, random_state=42))
]

metrics = [
    ("Accuracy Score", accuracy_score),
    ("F-Measure", f1_score),
    ("Sensitivity", recall_score),
    ("Specificity", recall_score),
    ("Precision", precision_score),
    ("Area Under ROC Curve", roc_auc_score),
    ("Area Under Precision Recall Curve", average_precision_score),
    ("MCC", matthews_corrcoef)
]

def dump_metrics_with_cv(features, label, name, model):
    accuracy = 0.0

    for metric_name, metric in metrics:
        kfold = StratifiedKFold(n_splits=10, random_state=42)
```

```
        scorer = make_scorer(metric,pos_label=0) if metric_name == "Specificity"␣
 ↪else make_scorer(metric)

        cv_result = cross_val_score(model,features,label.values.ravel(), cv =␣
 ↪kfold,scoring = scorer)

        if metric_name == "Accuracy Score":
            accuracy = cv_result

        print("{} {}: {}".format(name, metric_name, cv_result.mean()))
    print("")

    return accuracy.mean()

for name,model in models:
    print("Unnormalized Results for {}:\n".format(name))
    dump_metrics_with_cv(X_train_two, y_train_two, name, model)

    print("\nNormalized Results for {}:\n".format(name))
    dump_metrics_with_cv(normalized_x_train_two, y_train_two, name, model)
```

```
Unnormalized Results for Logistic Regression:

Logistic Regression Accuracy Score: 0.982649389021
Logistic Regression F-Measure: 0.96063221029
Logistic Regression Sensitivity: 0.99133041393
Logistic Regression Specificity: 0.980313787062
Logistic Regression Precision: 0.932265962074
Logistic Regression Area Under ROC Curve: 0.985822100496
Logistic Regression Area Under Precision Recall Curve: 0.925995821007
Logistic Regression MCC: 0.950474980812


Normalized Results for Logistic Regression:

Logistic Regression Accuracy Score: 0.988945271712
Logistic Regression F-Measure: 0.974631060518
Logistic Regression Sensitivity: 0.997826086957
Logistic Regression Specificity: 0.986551603826
Logistic Regression Precision: 0.952632448389
Logistic Regression Area Under ROC Curve: 0.992188845391
Logistic Regression Area Under Precision Recall Curve: 0.951010106793
Logistic Regression MCC: 0.968055905242

Unnormalized Results for K-Nearest Neighbour:

K-Nearest Neighbour Accuracy Score: 0.993398543946
```

K-Nearest Neighbour F-Measure: 0.98445991284
K-Nearest Neighbour Sensitivity: 0.985543738922
K-Nearest Neighbour Specificity: 0.995517706935
K-Nearest Neighbour Precision: 0.983501777729
K-Nearest Neighbour Area Under ROC Curve: 0.990530722928
K-Nearest Neighbour Area Under Precision Recall Curve: 0.972319984519
K-Nearest Neighbour MCC: 0.980319749123

Normalized Results for K-Nearest Neighbour:

K-Nearest Neighbour Accuracy Score: 0.993706940876
K-Nearest Neighbour F-Measure: 0.985086637998
K-Nearest Neighbour Sensitivity: 0.981221978939
K-Nearest Neighbour Specificity: 0.997077161126
K-Nearest Neighbour Precision: 0.989155665262
K-Nearest Neighbour Area Under ROC Curve: 0.989149570032
K-Nearest Neighbour Area Under Precision Recall Curve: 0.974525374825
K-Nearest Neighbour MCC: 0.98118028929

Unnormalized Results for Naive Bayes:

Naive Bayes Accuracy Score: 0.9788093774
Naive Bayes F-Measure: 0.952521719428
Naive Bayes Sensitivity: 0.997101449275
Naive Bayes Specificity: 0.973881797013
Naive Bayes Precision: 0.912119800735
Naive Bayes Area Under ROC Curve: 0.985491623144
Naive Bayes Area Under Precision Recall Curve: 0.910104028771
Naive Bayes MCC: 0.940630372536

Normalized Results for Naive Bayes:

Naive Bayes Accuracy Score: 0.989405865607
Naive Bayes F-Measure: 0.975666131904
Naive Bayes Sensitivity: 0.997826086957
Naive Bayes Specificity: 0.987136399147
Naive Bayes Precision: 0.954620247581
Naive Bayes Area Under ROC Curve: 0.992481243052
Naive Bayes Area Under Precision Recall Curve: 0.952988461982
Naive Bayes MCC: 0.969353111156

Unnormalized Results for Decision Tree:

Decision Tree Accuracy Score: 0.991249419899
Decision Tree F-Measure: 0.979375261706
Decision Tree Sensitivity: 0.979777916797

```
Decision Tree Specificity: 0.994348116292
Decision Tree Precision: 0.979241945317
Decision Tree Area Under ROC Curve: 0.987063016544
Decision Tree Area Under Precision Recall Curve: 0.963669231603
Decision Tree MCC: 0.973930345142


Normalized Results for Decision Tree:

Decision Tree Accuracy Score: 0.991095810068
Decision Tree F-Measure: 0.979009158276
Decision Tree Sensitivity: 0.979053279116
Decision Tree Specificity: 0.994348116292
Decision Tree Precision: 0.979216972787
Decision Tree Area Under ROC Curve: 0.986700697704
Decision Tree Area Under Precision Recall Curve: 0.963099108555
Decision Tree MCC: 0.97345996652

Unnormalized Results for Random Forest:

Random Forest Accuracy Score: 0.99416494608
Random Forest F-Measure: 0.986254839634
Random Forest Sensitivity: 0.986263163382
Random Forest Specificity: 0.996296675541
Random Forest Precision: 0.986355589415
Random Forest Area Under ROC Curve: 0.991279919462
Random Forest Area Under Precision Recall Curve: 0.975708013286
Random Forest MCC: 0.982595044853


Normalized Results for Random Forest:

Random Forest Accuracy Score: 0.994471930145
Random Forest F-Measure: 0.986979384422
Random Forest Sensitivity: 0.987707225524
Random Forest Specificity: 0.996296675541
Random Forest Precision: 0.986380706185
Random Forest Area Under ROC Curve: 0.992001950532
Random Forest Area Under Precision Recall Curve: 0.976844475321
Random Forest MCC: 0.983522506563

Unnormalized Results for Support Vector Machine:

Support Vector Machine Accuracy Score: 0.787688495819
Support Vector Machine F-Measure: 0.0
Support Vector Machine Sensitivity: 0.0
Support Vector Machine Specificity: 1.0
Support Vector Machine Precision: 0.0
```

```
Support Vector Machine Area Under ROC Curve: 0.5
Support Vector Machine Area Under Precision Recall Curve: 0.212311504181
Support Vector Machine MCC: 0.0


Normalized Results for Support Vector Machine:

Support Vector Machine Accuracy Score: 0.988792132355
Support Vector Machine F-Measure: 0.974310144264
Support Vector Machine Sensitivity: 0.998550724638
Support Vector Machine Specificity: 0.986162119523
Support Vector Machine Precision: 0.951353025314
Support Vector Machine Area Under ROC Curve: 0.99235642208
Support Vector Machine Area Under Precision Recall Curve: 0.950280821792
Support Vector Machine MCC: 0.967670688953

Unnormalized Results for Linear Discriminant Analysis:

Linear Discriminant Analysis Accuracy Score: 0.988791897479
Linear Discriminant Analysis F-Measure: 0.974307691132
Linear Discriminant Analysis Sensitivity: 0.998550724638
Linear Discriminant Analysis Specificity: 0.986161740278
Linear Discriminant Analysis Precision: 0.951343714701
Linear Discriminant Analysis Area Under ROC Curve: 0.992356232458
Linear Discriminant Analysis Area Under Precision Recall Curve: 0.950271511179
Linear Discriminant Analysis MCC: 0.96766639464


Normalized Results for Linear Discriminant Analysis:

Linear Discriminant Analysis Accuracy Score: 0.988791897479
Linear Discriminant Analysis F-Measure: 0.974307691132
Linear Discriminant Analysis Sensitivity: 0.998550724638
Linear Discriminant Analysis Specificity: 0.986161740278
Linear Discriminant Analysis Precision: 0.951343714701
Linear Discriminant Analysis Area Under ROC Curve: 0.992356232458
Linear Discriminant Analysis Area Under Precision Recall Curve: 0.950271511179
Linear Discriminant Analysis MCC: 0.96766639464

Unnormalized Results for AdaBoostClassifier:

AdaBoostClassifier Accuracy Score: 0.990941021526
AdaBoostClassifier F-Measure: 0.979023886831
AdaBoostClassifier Sensitivity: 0.992044625169
AdaBoostClassifier Specificity: 0.990645171077
AdaBoostClassifier Precision: 0.966554368179
AdaBoostClassifier Area Under ROC Curve: 0.991344898123
AdaBoostClassifier Area Under Precision Recall Curve: 0.960527205399
```

```
AdaBoostClassifier MCC: 0.973473086519


Normalized Results for AdaBoostClassifier:

AdaBoostClassifier Accuracy Score: 0.990941021526
AdaBoostClassifier F-Measure: 0.979023886831
AdaBoostClassifier Sensitivity: 0.992044625169
AdaBoostClassifier Specificity: 0.990645171077
AdaBoostClassifier Precision: 0.966554368179
AdaBoostClassifier Area Under ROC Curve: 0.991344898123
AdaBoostClassifier Area Under Precision Recall Curve: 0.960527205399
AdaBoostClassifier MCC: 0.973473086519
```

6. Use three feature selection methods to select feature subsets on train set 2 and compute accuracy measures in step 5 for all the classifiers. Repeat for normalized version of train set 2. Do you get improvement in accuracy when you perform feature selection or is it better to use all of the features? Which feature selection strategy gives the best accuracy?

[6]:
```
for name, model in models:
    selectors = [
        ("VarianceThreshold", VarianceThreshold()),
        ("SelectKBest with f_classif 1", SelectKBest(f_classif,k=1)),
        ("SelectKBest with chi2 1", SelectKBest(chi2,k=1)),
        ("SelectKBest with f_classif 2", SelectKBest(f_classif,k=2)),
        ("SelectKBest with chi2 2", SelectKBest(chi2,k=2)),
        ("SelectKBest with f_classif 3", SelectKBest(f_classif,k=3)),
        ("SelectKBest with chi2 3", SelectKBest(chi2,k=3)),
        ("SelectKBest with f_classif 4", SelectKBest(f_classif,k=4)),
        ("SelectKBest with chi2 4", SelectKBest(chi2,k=4)),
        ("SelectKBest with f_classif 5", SelectKBest(f_classif,k=5)),
        ("SelectKBest with chi2 5", SelectKBest(chi2,k=5)),
        ("SelectKBest with f_classif 6", SelectKBest(f_classif,k=6)),
        ("SelectKBest with chi2 7", SelectKBest(chi2,k=6)),
        ("SelectKBest with f_classif 7", SelectKBest(f_classif,k=7)),
        ("SelectKBest with chi2 7", SelectKBest(chi2,k=7))
    ]

    for selector_name, selector in selectors:
        selected_x_train_two = selector.fit_transform(X_train_two, y=y_train_two)
        normalized_selected_x_train_two  = selector.
↪fit_transform(normalized_x_train_two, y=y_train_two)

        print("{} unnormalized results:\n".format(selector_name))
        dump_metrics_with_cv(selected_x_train_two, y_train_two, name, model)
```

```
        print("{} normalized results:\n".format(selector_name))
        dump_metrics_with_cv(normalized_selected_x_train_two, y_train_two, name,␣
 ↪model)

print("Yes, I did get improvement when I performed feature selection.")
print("SelectKBest that uses chi2 with 4 features gave the best accuracy.")
```

```
Note: I only reported the results of each feature selection method with most␣
 ↪optimized parameter for each model. To see the results of all feature selection␣
 ↪methods, you can run the code or check the Jupyter Notebook in github.com/
 ↪mabdullahsoyturk/Occupancy-Detection

VarianceThreshold unnormalized results:

Logistic Regression Accuracy Score: 0.982649389021
Logistic Regression F-Measure: 0.96063221029
Logistic Regression Sensitivity: 0.99133041393
Logistic Regression Specificity: 0.980313787062
Logistic Regression Precision: 0.932265962074
Logistic Regression Area Under ROC Curve: 0.985822100496
Logistic Regression Area Under Precision Recall Curve: 0.925995821007
Logistic Regression MCC: 0.950474980812

VarianceThreshold normalized results:

Logistic Regression Accuracy Score: 0.988945271712
Logistic Regression F-Measure: 0.974631060518
Logistic Regression Sensitivity: 0.997826086957
Logistic Regression Specificity: 0.986551603826
Logistic Regression Precision: 0.952632448389
Logistic Regression Area Under ROC Curve: 0.992188845391
Logistic Regression Area Under Precision Recall Curve: 0.951010106793
Logistic Regression MCC: 0.968055905242

SelectKBest with f_classif 5 unnormalized results:

Logistic Regression Accuracy Score: 0.98863805205
Logistic Regression F-Measure: 0.97390423569
Logistic Regression Sensitivity: 0.996376811594
Logistic Regression Specificity: 0.986551603826
Logistic Regression Precision: 0.952569785956
Logistic Regression Area Under ROC Curve: 0.99146420771
Logistic Regression Area Under Precision Recall Curve: 0.949871018481
Logistic Regression MCC: 0.967110414133

SelectKBest with f_classif 5 normalized results:
```

```
Logistic Regression Accuracy Score: 0.988945271712
Logistic Regression F-Measure: 0.974631060518
Logistic Regression Sensitivity: 0.997826086957
Logistic Regression Specificity: 0.986551603826
Logistic Regression Precision: 0.952632448389
Logistic Regression Area Under ROC Curve: 0.992188845391
Logistic Regression Area Under Precision Recall Curve: 0.951010106793
Logistic Regression MCC: 0.968055905242

SelectKBest with chi2 5 unnormalized results:

Logistic Regression Accuracy Score: 0.968371914167
Logistic Regression F-Measure: 0.929181523093
Logistic Regression Sensitivity: 0.975456156814
Logistic Regression Specificity: 0.966475148095
Logistic Regression Precision: 0.887903710452
Logistic Regression Area Under ROC Curve: 0.970965652454
Logistic Regression Area Under Precision Recall Curve: 0.871197565725
Logistic Regression MCC: 0.910929195028

SelectKBest with chi2 5 normalized results:

Logistic Regression Accuracy Score: 0.988945271712
Logistic Regression F-Measure: 0.974631060518
Logistic Regression Sensitivity: 0.997826086957
Logistic Regression Specificity: 0.986551603826
Logistic Regression Precision: 0.952632448389
Logistic Regression Area Under ROC Curve: 0.992188845391
Logistic Regression Area Under Precision Recall Curve: 0.951010106793
Logistic Regression MCC: 0.968055905242

VarianceThreshold unnormalized results:

K-Nearest Neighbour Accuracy Score: 0.993398543946
K-Nearest Neighbour F-Measure: 0.98445991284
K-Nearest Neighbour Sensitivity: 0.985543738922
K-Nearest Neighbour Specificity: 0.995517706935
K-Nearest Neighbour Precision: 0.983501777729
K-Nearest Neighbour Area Under ROC Curve: 0.990530722928
K-Nearest Neighbour Area Under Precision Recall Curve: 0.972319984519
K-Nearest Neighbour MCC: 0.980319749123

VarianceThreshold normalized results:

K-Nearest Neighbour Accuracy Score: 0.993706940876
K-Nearest Neighbour F-Measure: 0.985086637998
K-Nearest Neighbour Sensitivity: 0.981221978939
```

```
K-Nearest Neighbour Specificity: 0.997077161126
K-Nearest Neighbour Precision: 0.989155665262
K-Nearest Neighbour Area Under ROC Curve: 0.989149570032
K-Nearest Neighbour Area Under Precision Recall Curve: 0.974525374825
K-Nearest Neighbour MCC: 0.98118028929

SelectKBest with chi2 7 unnormalized results:

K-Nearest Neighbour Accuracy Score: 0.993398543946
K-Nearest Neighbour F-Measure: 0.98445991284
K-Nearest Neighbour Sensitivity: 0.985543738922
K-Nearest Neighbour Specificity: 0.995517706935
K-Nearest Neighbour Precision: 0.983501777729
K-Nearest Neighbour Area Under ROC Curve: 0.990530722928
K-Nearest Neighbour Area Under Precision Recall Curve: 0.972319984519
K-Nearest Neighbour MCC: 0.980319749123

SelectKBest with chi2 7 normalized results:

K-Nearest Neighbour Accuracy Score: 0.993399721214
K-Nearest Neighbour F-Measure: 0.984383043879
K-Nearest Neighbour Sensitivity: 0.98194661662
K-Nearest Neighbour Specificity: 0.996492365804
K-Nearest Neighbour Precision: 0.987018253013
K-Nearest Neighbour Area Under ROC Curve: 0.989219491212
K-Nearest Neighbour Area Under Precision Recall Curve: 0.972979612679
K-Nearest Neighbour MCC: 0.98027803497

SelectKBest with f_classif 7 unnormalized results:

K-Nearest Neighbour Accuracy Score: 0.993398543946
K-Nearest Neighbour F-Measure: 0.98445991284
K-Nearest Neighbour Sensitivity: 0.985543738922
K-Nearest Neighbour Specificity: 0.995517706935
K-Nearest Neighbour Precision: 0.983501777729
K-Nearest Neighbour Area Under ROC Curve: 0.990530722928
K-Nearest Neighbour Area Under Precision Recall Curve: 0.972319984519
K-Nearest Neighbour MCC: 0.980319749123

SelectKBest with f_classif 7 normalized results:

K-Nearest Neighbour Accuracy Score: 0.993706940876
K-Nearest Neighbour F-Measure: 0.985086637998
K-Nearest Neighbour Sensitivity: 0.981221978939
K-Nearest Neighbour Specificity: 0.997077161126
K-Nearest Neighbour Precision: 0.989155665262
K-Nearest Neighbour Area Under ROC Curve: 0.989149570032
K-Nearest Neighbour Area Under Precision Recall Curve: 0.974525374825
```

```
K-Nearest Neighbour MCC: 0.98118028929

SelectKBest with chi2 7 unnormalized results:

K-Nearest Neighbour Accuracy Score: 0.993398543946
K-Nearest Neighbour F-Measure: 0.98445991284
K-Nearest Neighbour Sensitivity: 0.985543738922
K-Nearest Neighbour Specificity: 0.995517706935
K-Nearest Neighbour Precision: 0.983501777729
K-Nearest Neighbour Area Under ROC Curve: 0.990530722928
K-Nearest Neighbour Area Under Precision Recall Curve: 0.972319984519
K-Nearest Neighbour MCC: 0.980319749123

SelectKBest with chi2 7 normalized results:

K-Nearest Neighbour Accuracy Score: 0.993706940876
K-Nearest Neighbour F-Measure: 0.985086637998
K-Nearest Neighbour Sensitivity: 0.981221978939
K-Nearest Neighbour Specificity: 0.997077161126
K-Nearest Neighbour Precision: 0.989155665262
K-Nearest Neighbour Area Under ROC Curve: 0.989149570032
K-Nearest Neighbour Area Under Precision Recall Curve: 0.974525374825
K-Nearest Neighbour MCC: 0.98118028929

VarianceThreshold unnormalized results:

Naive Bayes Accuracy Score: 0.9788093774
Naive Bayes F-Measure: 0.952521719428
Naive Bayes Sensitivity: 0.997101449275
Naive Bayes Specificity: 0.973881797013
Naive Bayes Precision: 0.912119800735
Naive Bayes Area Under ROC Curve: 0.985491623144
Naive Bayes Area Under Precision Recall Curve: 0.910104028771
Naive Bayes MCC: 0.940630372536

VarianceThreshold normalized results:

Naive Bayes Accuracy Score: 0.989405865607
Naive Bayes F-Measure: 0.975666131904
Naive Bayes Sensitivity: 0.997826086957
Naive Bayes Specificity: 0.987136399147
Naive Bayes Precision: 0.954620247581
Naive Bayes Area Under ROC Curve: 0.992481243052
Naive Bayes Area Under Precision Recall Curve: 0.952988461982
Naive Bayes MCC: 0.969353111156

SelectKBest with f_classif 3 unnormalized results:
```

```
Naive Bayes Accuracy Score: 0.978041563842
Naive Bayes F-Measure: 0.950890243257
Naive Bayes Sensitivity: 0.997101449275
Naive Bayes Specificity: 0.972907138144
Naive Bayes Precision: 0.909146439299
Naive Bayes Area Under ROC Curve: 0.98500429371
Naive Bayes Area Under Precision Recall Curve: 0.907143358172
Naive Bayes MCC: 0.938620991197

SelectKBest with f_classif 3 normalized results:

Naive Bayes Accuracy Score: 0.978041563842
Naive Bayes F-Measure: 0.950890243257
Naive Bayes Sensitivity: 0.997101449275
Naive Bayes Specificity: 0.972907138144
Naive Bayes Precision: 0.909146439299
Naive Bayes Area Under ROC Curve: 0.98500429371
Naive Bayes Area Under Precision Recall Curve: 0.907143358172
Naive Bayes MCC: 0.938620991197

SelectKBest with chi2 3 unnormalized results:

Naive Bayes Accuracy Score: 0.979270206893
Naive Bayes F-Measure: 0.953516226651
Naive Bayes Sensitivity: 0.997101449275
Naive Bayes Specificity: 0.974466592335
Naive Bayes Precision: 0.913958891482
Naive Bayes Area Under ROC Curve: 0.985784020805
Naive Bayes Area Under Precision Recall Curve: 0.911929792772
Naive Bayes MCC: 0.941861394552

SelectKBest with chi2 3 normalized results:

Naive Bayes Accuracy Score: 0.989405865607
Naive Bayes F-Measure: 0.975686297035
Naive Bayes Sensitivity: 0.998550724638
Naive Bayes Specificity: 0.986941467374
Naive Bayes Precision: 0.953988740576
Naive Bayes Area Under ROC Curve: 0.992746096006
Naive Bayes Area Under Precision Recall Curve: 0.952907221983
Naive Bayes MCC: 0.969393451809

VarianceThreshold unnormalized results:

Decision Tree Accuracy Score: 0.991249419899
Decision Tree F-Measure: 0.979375261706
Decision Tree Sensitivity: 0.979777916797
Decision Tree Specificity: 0.994348116292
```

```
Decision Tree Precision: 0.979241945317
Decision Tree Area Under ROC Curve: 0.987063016544
Decision Tree Area Under Precision Recall Curve: 0.963669231603
Decision Tree MCC: 0.973930345142

VarianceThreshold normalized results:

Decision Tree Accuracy Score: 0.991095810068
Decision Tree F-Measure: 0.979009158276
Decision Tree Sensitivity: 0.979053279116
Decision Tree Specificity: 0.994348116292
Decision Tree Precision: 0.979216972787
Decision Tree Area Under ROC Curve: 0.986700697704
Decision Tree Area Under Precision Recall Curve: 0.963099108555
Decision Tree MCC: 0.97345996652

SelectKBest with f_classif 3 unnormalized results:

Decision Tree Accuracy Score: 0.991095102552
Decision Tree F-Measure: 0.978949749052
Decision Tree Sensitivity: 0.976139088729
Decision Tree Specificity: 0.995127464142
Decision Tree Precision: 0.981921099169
Decision Tree Area Under ROC Curve: 0.985633276436
Decision Tree Area Under Precision Recall Curve: 0.963605607955
Decision Tree MCC: 0.973367463857

SelectKBest with f_classif 3 normalized results:

Decision Tree Accuracy Score: 0.990941492721
Decision Tree F-Measure: 0.978575752221
Decision Tree Sensitivity: 0.975414451048
Decision Tree Specificity: 0.995127464142
Decision Tree Precision: 0.981890266692
Decision Tree Area Under ROC Curve: 0.985270957595
Decision Tree Area Under Precision Recall Curve: 0.963035920647
Decision Tree MCC: 0.972893818197

SelectKBest with chi2 3 unnormalized results:

Decision Tree Accuracy Score: 0.993704587062
Decision Tree F-Measure: 0.98513992865
Decision Tree Sensitivity: 0.984803461579
Decision Tree Specificity: 0.996102123012
Decision Tree Precision: 0.985582046666
Decision Tree Area Under ROC Curve: 0.990452792295
Decision Tree Area Under Precision Recall Curve: 0.97383173302
Decision Tree MCC: 0.981188827938
```

```
SelectKBest with chi2 3 normalized results:

Decision Tree Accuracy Score: 0.985721820518
Decision Tree F-Measure: 0.966309126118
Decision Tree Sensitivity: 0.964576165155
Decision Tree Specificity: 0.991423760439
Decision Tree Precision: 0.968151200733
Decision Tree Area Under ROC Curve: 0.977999962797
Decision Tree Area Under Precision Recall Curve: 0.941562319991
Decision Tree MCC: 0.957294468627

VarianceThreshold unnormalized results:

Random Forest Accuracy Score: 0.99416494608
Random Forest F-Measure: 0.986254839634
Random Forest Sensitivity: 0.986263163382
Random Forest Specificity: 0.996296675541
Random Forest Precision: 0.986355589415
Random Forest Area Under ROC Curve: 0.991279919462
Random Forest Area Under Precision Recall Curve: 0.975708013286
Random Forest MCC: 0.982595044853

VarianceThreshold normalized results:

Random Forest Accuracy Score: 0.994471930145
Random Forest F-Measure: 0.986979384422
Random Forest Sensitivity: 0.987707225524
Random Forest Specificity: 0.996296675541
Random Forest Precision: 0.986380706185
Random Forest Area Under ROC Curve: 0.992001950532
Random Forest Area Under Precision Recall Curve: 0.976844475321
Random Forest MCC: 0.983522506563

SelectKBest with f_classif 4 unnormalized results:

Random Forest Accuracy Score: 0.993244934836
Random Forest F-Measure: 0.984080840136
Random Forest Sensitivity: 0.984104890001
Random Forest Specificity: 0.995712259464
Random Forest Precision: 0.98416026882
Random Forest Area Under ROC Curve: 0.989908574733
Random Forest Area Under Precision Recall Curve: 0.971883049162
Random Forest MCC: 0.979835341877

SelectKBest with f_classif 4 normalized results:

Random Forest Accuracy Score: 0.993244934836
```

```
Random Forest F-Measure: 0.984080840136
Random Forest Sensitivity: 0.984104890001
Random Forest Specificity: 0.995712259464
Random Forest Precision: 0.98416026882
Random Forest Area Under ROC Curve: 0.989908574733
Random Forest Area Under Precision Recall Curve: 0.971883049162
Random Forest MCC: 0.979835341877

SelectKBest with chi2 4 unnormalized results:

Random Forest Accuracy Score: 0.995393824007
Random Forest F-Measure: 0.989186958302
Random Forest Sensitivity: 0.991325200709
Random Forest Specificity: 0.99649198656
Random Forest Precision: 0.987195908607
Random Forest Area Under ROC Curve: 0.993908593634
Random Forest Area Under Precision Recall Curve: 0.980456977904
Random Forest MCC: 0.986320150773

SelectKBest with chi2 4 normalized results:

Random Forest Accuracy Score: 0.993244934836
Random Forest F-Measure: 0.984080840136
Random Forest Sensitivity: 0.984104890001
Random Forest Specificity: 0.995712259464
Random Forest Precision: 0.98416026882
Random Forest Area Under ROC Curve: 0.989908574733
Random Forest Area Under Precision Recall Curve: 0.971883049162
Random Forest MCC: 0.979835341877

VarianceThreshold unnormalized results:

Support Vector Machine Accuracy Score: 0.787688495819
Support Vector Machine F-Measure: 0.0
Support Vector Machine Sensitivity: 0.0
Support Vector Machine Specificity: 1.0
Support Vector Machine Precision: 0.0
Support Vector Machine Area Under ROC Curve: 0.5
Support Vector Machine Area Under Precision Recall Curve: 0.212311504181
Support Vector Machine MCC: 0.0

VarianceThreshold normalized results:

Support Vector Machine Accuracy Score: 0.988792132355
Support Vector Machine F-Measure: 0.974310144264
Support Vector Machine Sensitivity: 0.998550724638
Support Vector Machine Specificity: 0.986162119523
Support Vector Machine Precision: 0.951353025314
```

```
Support Vector Machine Area Under ROC Curve: 0.99235642208
Support Vector Machine Area Under Precision Recall Curve: 0.950280821792
Support Vector Machine MCC: 0.967670688953

SelectKBest with f_classif 6 unnormalized results:

Support Vector Machine Accuracy Score: 0.96008357426
Support Vector Machine F-Measure: 0.914083907246
Support Vector Machine Sensitivity: 0.994932749453
Support Vector Machine Specificity: 0.950690604592
Support Vector Machine Precision: 0.846063019084
Support Vector Machine Area Under ROC Curve: 0.972811677022
Support Vector Machine Area Under Precision Recall Curve: 0.842799760223
Support Vector Machine MCC: 0.893620540006

SelectKBest with f_classif 6 normalized results:

Support Vector Machine Accuracy Score: 0.988792132355
Support Vector Machine F-Measure: 0.974310144264
Support Vector Machine Sensitivity: 0.998550724638
Support Vector Machine Specificity: 0.986162119523
Support Vector Machine Precision: 0.951353025314
Support Vector Machine Area Under ROC Curve: 0.99235642208
Support Vector Machine Area Under Precision Recall Curve: 0.950280821792
Support Vector Machine MCC: 0.967670688953

SelectKBest with chi2 6 unnormalized results:

Support Vector Machine Accuracy Score: 0.787688495819
Support Vector Machine F-Measure: 0.0
Support Vector Machine Sensitivity: 0.0
Support Vector Machine Specificity: 1.0
Support Vector Machine Precision: 0.0
Support Vector Machine Area Under ROC Curve: 0.5
Support Vector Machine Area Under Precision Recall Curve: 0.212311504181
Support Vector Machine MCC: 0.0

SelectKBest with chi2 6 normalized results:

Support Vector Machine Accuracy Score: 0.988792132355
Support Vector Machine F-Measure: 0.974310144264
Support Vector Machine Sensitivity: 0.998550724638
Support Vector Machine Specificity: 0.986162119523
Support Vector Machine Precision: 0.951353025314
Support Vector Machine Area Under ROC Curve: 0.99235642208
Support Vector Machine Area Under Precision Recall Curve: 0.950280821792
Support Vector Machine MCC: 0.967670688953
```

```
VarianceThreshold unnormalized results:

Linear Discriminant Analysis Accuracy Score: 0.988791897479
Linear Discriminant Analysis F-Measure: 0.974307691132
Linear Discriminant Analysis Sensitivity: 0.998550724638
Linear Discriminant Analysis Specificity: 0.986161740278
Linear Discriminant Analysis Precision: 0.951343714701
Linear Discriminant Analysis Area Under ROC Curve: 0.992356232458
Linear Discriminant Analysis Area Under Precision Recall Curve: 0.950271511179
Linear Discriminant Analysis MCC: 0.96766639464

VarianceThreshold normalized results:

Linear Discriminant Analysis Accuracy Score: 0.988791897479
Linear Discriminant Analysis F-Measure: 0.974307691132
Linear Discriminant Analysis Sensitivity: 0.998550724638
Linear Discriminant Analysis Specificity: 0.986161740278
Linear Discriminant Analysis Precision: 0.951343714701
Linear Discriminant Analysis Area Under ROC Curve: 0.992356232458
Linear Discriminant Analysis Area Under Precision Recall Curve: 0.950271511179
Linear Discriminant Analysis MCC: 0.96766639464

SelectKBest with f_classif 4 unnormalized results:

Linear Discriminant Analysis Accuracy Score: 0.988791897479
Linear Discriminant Analysis F-Measure: 0.974307691132
Linear Discriminant Analysis Sensitivity: 0.998550724638
Linear Discriminant Analysis Specificity: 0.986161740278
Linear Discriminant Analysis Precision: 0.951343714701
Linear Discriminant Analysis Area Under ROC Curve: 0.992356232458
Linear Discriminant Analysis Area Under Precision Recall Curve: 0.950271511179
Linear Discriminant Analysis MCC: 0.96766639464

SelectKBest with f_classif 4 normalized results:

Linear Discriminant Analysis Accuracy Score: 0.988791897479
Linear Discriminant Analysis F-Measure: 0.974307691132
Linear Discriminant Analysis Sensitivity: 0.998550724638
Linear Discriminant Analysis Specificity: 0.986161740278
Linear Discriminant Analysis Precision: 0.951343714701
Linear Discriminant Analysis Area Under ROC Curve: 0.992356232458
Linear Discriminant Analysis Area Under Precision Recall Curve: 0.950271511179
Linear Discriminant Analysis MCC: 0.96766639464

SelectKBest with chi2 4 unnormalized results:

Linear Discriminant Analysis Accuracy Score: 0.982495071674
Linear Discriminant Analysis F-Measure: 0.960499076636
```

```
Linear Discriminant Analysis Sensitivity: 0.998550724638
Linear Discriminant Analysis Specificity: 0.978169916794
Linear Discriminant Analysis Precision: 0.925512868061
Linear Discriminant Analysis Area Under ROC Curve: 0.988360320716
Linear Discriminant Analysis Area Under Precision Recall Curve: 0.924504951547
Linear Discriminant Analysis MCC: 0.950530524943

SelectKBest with chi2 4 normalized results:

Linear Discriminant Analysis Accuracy Score: 0.988791897479
Linear Discriminant Analysis F-Measure: 0.974307691132
Linear Discriminant Analysis Sensitivity: 0.998550724638
Linear Discriminant Analysis Specificity: 0.986161740278
Linear Discriminant Analysis Precision: 0.951343714701
Linear Discriminant Analysis Area Under ROC Curve: 0.992356232458
Linear Discriminant Analysis Area Under Precision Recall Curve: 0.950271511179
Linear Discriminant Analysis MCC: 0.96766639464

VarianceThreshold unnormalized results:

AdaBoostClassifier Accuracy Score: 0.990941021526
AdaBoostClassifier F-Measure: 0.979023886831
AdaBoostClassifier Sensitivity: 0.992044625169
AdaBoostClassifier Specificity: 0.990645171077
AdaBoostClassifier Precision: 0.966554368179
AdaBoostClassifier Area Under ROC Curve: 0.991344898123
AdaBoostClassifier Area Under Precision Recall Curve: 0.960527205399
AdaBoostClassifier MCC: 0.973473086519

VarianceThreshold normalized results:

AdaBoostClassifier Accuracy Score: 0.990941021526
AdaBoostClassifier F-Measure: 0.979023886831
AdaBoostClassifier Sensitivity: 0.992044625169
AdaBoostClassifier Specificity: 0.990645171077
AdaBoostClassifier Precision: 0.966554368179
AdaBoostClassifier Area Under ROC Curve: 0.991344898123
AdaBoostClassifier Area Under Precision Recall Curve: 0.960527205399
AdaBoostClassifier MCC: 0.973473086519

SelectKBest with chi2 7 unnormalized results:

AdaBoostClassifier Accuracy Score: 0.990633801864
AdaBoostClassifier F-Measure: 0.97830203946
AdaBoostClassifier Sensitivity: 0.991319987488
AdaBoostClassifier Specificity: 0.990450239303
AdaBoostClassifier Precision: 0.96582587332
AdaBoostClassifier Area Under ROC Curve: 0.990885113396
```

```
AdaBoostClassifier Area Under Precision Recall Curve: 0.959268933015
AdaBoostClassifier MCC: 0.97255222215

SelectKBest with chi2 7 normalized results:

AdaBoostClassifier Accuracy Score: 0.988791661881
AdaBoostClassifier F-Measure: 0.974101584023
AdaBoostClassifier Sensitivity: 0.991309561047
AdaBoostClassifier Specificity: 0.988111058017
AdaBoostClassifier Precision: 0.957665440941
AdaBoostClassifier Area Under ROC Curve: 0.989710309532
AdaBoostClassifier Area Under Precision Recall Curve: 0.9511552088
AdaBoostClassifier MCC: 0.967266630833

SelectKBest with f_classif 7 unnormalized results:

AdaBoostClassifier Accuracy Score: 0.990941021526
AdaBoostClassifier F-Measure: 0.979023886831
AdaBoostClassifier Sensitivity: 0.992044625169
AdaBoostClassifier Specificity: 0.990645171077
AdaBoostClassifier Precision: 0.966554368179
AdaBoostClassifier Area Under ROC Curve: 0.991344898123
AdaBoostClassifier Area Under Precision Recall Curve: 0.960527205399
AdaBoostClassifier MCC: 0.973473086519

SelectKBest with f_classif 7 normalized results:

AdaBoostClassifier Accuracy Score: 0.990941021526
AdaBoostClassifier F-Measure: 0.979023886831
AdaBoostClassifier Sensitivity: 0.992044625169
AdaBoostClassifier Specificity: 0.990645171077
AdaBoostClassifier Precision: 0.966554368179
AdaBoostClassifier Area Under ROC Curve: 0.991344898123
AdaBoostClassifier Area Under Precision Recall Curve: 0.960527205399
AdaBoostClassifier MCC: 0.973473086519

SelectKBest with chi2 7 unnormalized results:

AdaBoostClassifier Accuracy Score: 0.990941021526
AdaBoostClassifier F-Measure: 0.979023886831
AdaBoostClassifier Sensitivity: 0.992044625169
AdaBoostClassifier Specificity: 0.990645171077
AdaBoostClassifier Precision: 0.966554368179
AdaBoostClassifier Area Under ROC Curve: 0.991344898123
AdaBoostClassifier Area Under Precision Recall Curve: 0.960527205399
AdaBoostClassifier MCC: 0.973473086519

SelectKBest with chi2 7 normalized results:
```

```
AdaBoostClassifier Accuracy Score: 0.990941021526
AdaBoostClassifier F-Measure: 0.979023886831
AdaBoostClassifier Sensitivity: 0.992044625169
AdaBoostClassifier Specificity: 0.990645171077
AdaBoostClassifier Precision: 0.966554368179
AdaBoostClassifier Area Under ROC Curve: 0.991344898123
AdaBoostClassifier Area Under Precision Recall Curve: 0.960527205399
AdaBoostClassifier MCC: 0.973473086519

Yes, I did get improvement when I performed feature selection.
SelectKBest that uses chi2 with 4 features gave the best accuracy.
```

7. Choose the version of train set 2 that contains the optimum feature set you found in step 6 and the data for the best normalization strategy. Optimize the following hyperparameters:

k parameter in k-NN number of trees in random forest Number of iterations in Adaboost C, gamma parameter in SVM

Try a grid of values and choose the best value(s) that maximize the overall cross validation accuracy.

- For k-NN you can choose 1 5 10 15 . . . 100 (with increments of 5 after k=5)
- For number of trees in random forest you can try 5 10 25 50 75 100 150 200 250 300 350 400 450 500
- For the number of iterations in Adaboost you can try 5 10 15 20 25 30 40 50 75 100 125 150 175 200

To optimize C and gamma parameters of the SVM you can consider the following parameter grid:

C {2^-5, 2^-3, 2^-1, 2^1, 2^3, 2^5, . . . 2^13, 2^15} {2^-15, 2^-13, . . . , 2^-1, 2^1, 2^3, 2^5}

There are a total of 11 values for the C parameter and 11 values for the gamma parameter (a total of 121 values to consider for the (C, gamma) pair).

Report the best cross-validation accuracies and optimum parameter values you found.

Compute predictions on the validation set using the models trained by optimum hyperparameters. Report the same accuracy measures as in step 5.

[7]:
```
best_selector = SelectKBest(chi2,k=4)
optimum_x_train_two = best_selector.fit_transform(X_train_two, y=y_train_two)

hyper_parameters = {
    "K-Nearest Neighbour": [1] + [i*5 for i in range(1,21)],
    "Random Forest": [5,10,25,50,75,100,150,200,250,300,350,400,450,500],
    "AdaBoostClassifier": [5,10,15,20,25,30,40,50,75,100,125,150,175,200],
    "Support Vector Machine": ([2**i for i in range(-5,16,2)], [2**i for i in␣
 ↪range(-15,6,2)])
}
```

```
opt_params = dict()

for model_name, parameters in hyper_parameters.items():
    model = None
    optimum_accuracy = 0.0
    optimum_parameters = ""

    if model_name == "K-Nearest Neighbour":
        for parameter in parameters:
            print("n_neighbors: {}".format(parameter))
            model = KNeighborsClassifier(n_neighbors=parameter)
            accuracy = dump_metrics_with_cv(optimum_x_train_two, y_train_two,␣
 ↪model_name, model)

            if accuracy > optimum_accuracy:
                optimum_accuracy = accuracy
                optimum_parameters = "n_neighbors=" + str(parameter)
                opt_params["n_neighbors"] = parameter
    elif model_name == "Random Forest":
        for parameter in parameters:
            print("n_estimators: {}".format(parameter))
            model = RandomForestClassifier(n_estimators=parameter)
            accuracy = dump_metrics_with_cv(optimum_x_train_two, y_train_two,␣
 ↪model_name, model)

            if accuracy > optimum_accuracy:
                optimum_accuracy = accuracy
                optimum_parameters = "n_estimators=" + str(parameter)
                opt_params["rf_estimators"] = parameter
    elif model_name == "Support Vector Machine":
        C_values = parameters[0]
        gamma_values = parameters[1]

        for C in C_values:
            for gamma in gamma_values:
                print("C: {}, gamma: {}".format(C, gamma))
                model = SVC(kernel="rbf", C=C,gamma=gamma)
                accuracy = dump_metrics_with_cv(optimum_x_train_two,␣
 ↪y_train_two, model_name, model)

                if accuracy > optimum_accuracy:
                    optimum_accuracy = accuracy
                    optimum_parameters = "C:" + str(C) + ", " + "gamma:" +␣
 ↪str(gamma)
                    opt_params["svm_c"] = C
                    opt_params["svm_gamma"] = gamma
    elif model_name == "AdaBoostClassifier":
```

```
        for parameter in parameters:
            print("n_estimators: {}".format(parameter))
            model = AdaBoostClassifier(n_estimators=parameter)
            accuracy = dump_metrics_with_cv(optimum_x_train_two, y_train_two,␣
 ↪model_name, model)

            if accuracy > optimum_accuracy:
                optimum_accuracy = accuracy
                optimum_parameters = "n_estimators=" + str(parameter)
                opt_params["ab_estimators"] = parameter

    print("Optimum Accuracy for {}: {}".format(model_name, optimum_accuracy))
    print("Optimum Parameters:{}\n\n".format(optimum_parameters))
```

```
Optimum Accuracy for AdaBoostClassifier: 0.992477120558
Optimum Parameters:n_estimators=125

Optimum Accuracy for Support Vector Machine: 0.994626480924
Optimum Parameters:C:2, gamma:3.0517578125e-05

Optimum Accuracy for K-Nearest Neighbour: 0.993551918179
Optimum Parameters:n_neighbors=1

Optimum Accuracy for Random Forest: 0.995086839943
Optimum Parameters:n_estimators=100
```

[9]:
```
classifiers = [
    ("K-Nearest␣
 ↪Neighbour",KNeighborsClassifier(n_neighbors=opt_params["n_neighbors"])),
    ("Random␣
 ↪Forest",RandomForestClassifier(n_estimators=opt_params["rf_estimators"],␣
 ↪random_state=42)),
    ("Support Vector Machine",SVC(kernel="rbf",␣
 ↪C=opt_params["svm_c"],gamma=opt_params["svm_gamma"], random_state=42)),
    ␣
 ↪("AdaBoostClassifier",AdaBoostClassifier(n_estimators=opt_params["ab_estimators"],␣
 ↪random_state=42))
]

def dump_metrics_without_cv(features, label, validation, name, model):
    model.fit(features, label)
    preds = model.predict(validation)
```

```
    print("Accuracy of {} with optimal hyperparameters on validation set: {}".
 ↪format(name, accuracy_score(y_validation, preds)))
    print("F-Score of {} with optimal hyperparameters on validation set: {}".
 ↪format(name, f1_score(y_validation, preds)))
    print("Sensitivity of {} with optimal hyperparameters on validation set: {}".
 ↪format(name, recall_score(y_validation, preds)))
    print("Specificity of {} with optimal hyperparameters on validation set: {}".
 ↪format(name, recall_score(y_validation, preds,pos_label=0)))
    print("Precision of {} with optimal hyperparameters on validation set: {}".
 ↪format(name, precision_score(y_validation, preds)))
    print("ROC-Auc Score of {} with optimal hyperparameters on validation set:␣
 ↪{}".format(name, roc_auc_score(y_validation, preds)))
    print("MCC of {} with optimal hyperparameters on validation set: {}\n\n".
 ↪format(name, matthews_corrcoef(y_validation, preds)))

optimum_x_validation = best_selector.fit_transform(X_validation, y=y_validation)

for name,model in classifiers:
    dump_metrics_without_cv(optimum_x_train_two, y_train_two,␣
 ↪optimum_x_validation, name, model)
```

```
Accuracy of K-Nearest Neighbour with optimal hyperparameters on validation set:
0.995089011664
F-Score of K-Nearest Neighbour with optimal hyperparameters on validation set:
0.988505747126
Sensitivity of K-Nearest Neighbour with optimal hyperparameters on validation
set: 0.994219653179
Specificity of K-Nearest Neighbour with optimal hyperparameters on validation
set: 0.995323460639
Precision of K-Nearest Neighbour with optimal hyperparameters on validation set:
0.982857142857
ROC-Auc Score of K-Nearest Neighbour with optimal hyperparameters on validation
set: 0.994771556909
MCC of K-Nearest Neighbour with optimal hyperparameters on validation set:
0.985409634408


Accuracy of Random Forest with optimal hyperparameters on validation set:
0.993247391037
F-Score of Random Forest with optimal hyperparameters on validation set:
0.98426323319
Sensitivity of Random Forest with optimal hyperparameters on validation set:
0.994219653179
Specificity of Random Forest with optimal hyperparameters on validation set:
0.992985190959
Precision of Random Forest with optimal hyperparameters on validation set:
0.974504249292
```

```
ROC-Auc Score of Random Forest with optimal hyperparameters on validation set:
0.993602422069
MCC of Random Forest with optimal hyperparameters on validation set:
0.98004488291


Accuracy of Support Vector Machine with optimal hyperparameters on validation
set: 0.995702885206
F-Score of Support Vector Machine with optimal hyperparameters on validation
set: 0.989928057554
Sensitivity of Support Vector Machine with optimal hyperparameters on validation
set: 0.994219653179
Specificity of Support Vector Machine with optimal hyperparameters on validation
set: 0.996102883866
Precision of Support Vector Machine with optimal hyperparameters on validation
set: 0.985673352436
ROC-Auc Score of Support Vector Machine with optimal hyperparameters on
validation set: 0.995161268523
MCC of Support Vector Machine with optimal hyperparameters on validation set:
0.987211809146


Accuracy of AdaBoostClassifier with optimal hyperparameters on validation set:
0.993247391037
F-Score of AdaBoostClassifier with optimal hyperparameters on validation set:
0.984218077475
Sensitivity of AdaBoostClassifier with optimal hyperparameters on validation
set: 0.991329479769
Specificity of AdaBoostClassifier with optimal hyperparameters on validation
set: 0.993764614186
Precision of AdaBoostClassifier with optimal hyperparameters on validation set:
0.977207977208
ROC-Auc Score of AdaBoostClassifier with optimal hyperparameters on validation
set: 0.992547046977
MCC of AdaBoostClassifier with optimal hyperparameters on validation set:
0.979963968451
```

8. Implement a stacking ensemble, which combines the best performing classifiers obtained in step 7 by a meta-learner (which can be logistic regression). Here you will use the optimum hyper-parameters you found in step 7 to train the models you selected in stacking. You can try different combinations of classifiers for this purpose. Perform cross-validation and report the same accuracy measures as in step 5. Then train the model on the train set 2 and test on validation set. Report the accuracy measures on validation data.

[10]:
```
classifiers_for_stacking = [
    KNeighborsClassifier(n_neighbors=opt_params["n_neighbors"]),
    RandomForestClassifier(n_estimators=opt_params["rf_estimators"],␣
 ↪random_state=42),
    SVC(kernel="rbf", C=opt_params["svm_c"],gamma=opt_params["svm_gamma"],␣
 ↪random_state=42, probability=True),
    AdaBoostClassifier(n_estimators=opt_params["ab_estimators"], random_state=42)
]

lr = LogisticRegression(random_state=42)

print("Scores with optimum data:\n")

sclf = StackingClassifier(classifiers=classifiers_for_stacking,␣
 ↪meta_classifier=lr)
dump_metrics_with_cv(optimum_x_train_two, y_train_two, "Stacking Ensemble", sclf)

print("Scores with validation data:\n")
dump_metrics_without_cv(optimum_x_train_two, y_train_two, optimum_x_validation,␣
 ↪"Stacking Ensemble", sclf)
```

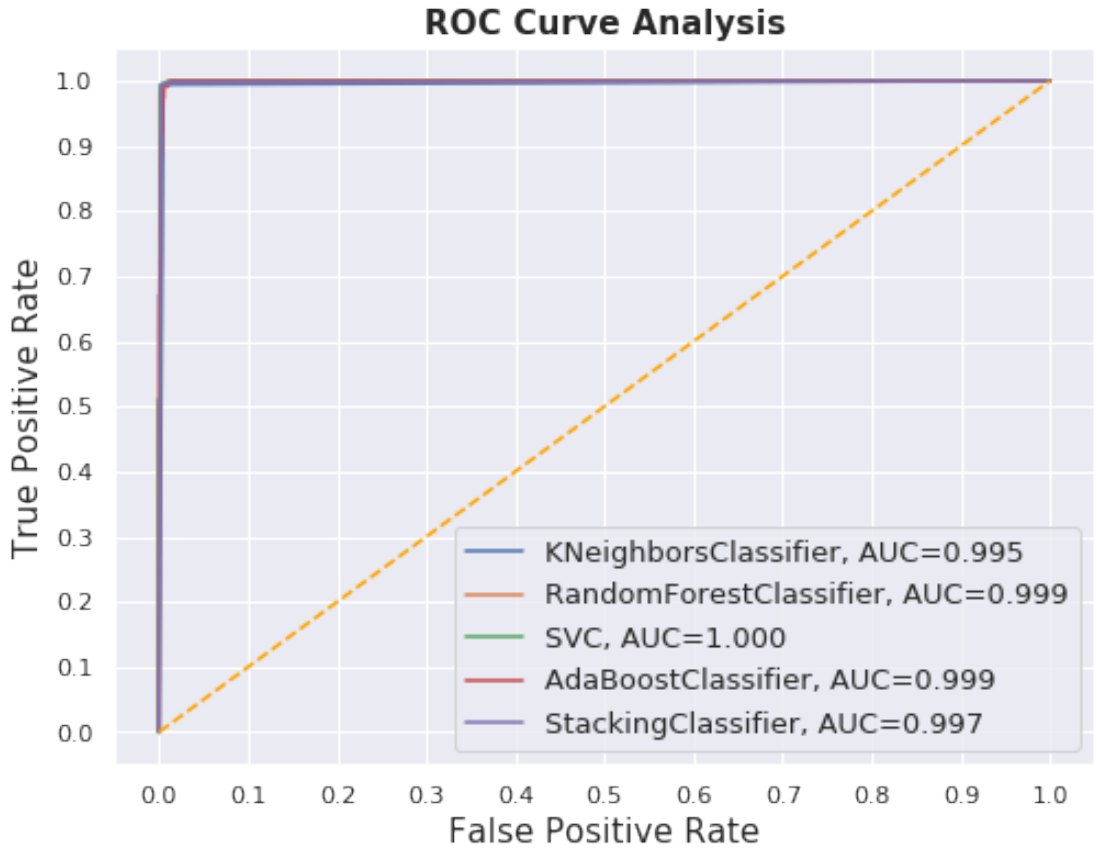
```
Scores with optimum data:

Stacking Ensemble Accuracy Score: 0.995855123974
Stacking Ensemble F-Measure: 0.9902519296
Stacking Ensemble Sensitivity: 0.99133041393
Stacking Ensemble Specificity: 0.997076781881
Stacking Ensemble Precision: 0.989303514095
Stacking Ensemble Area Under ROC Curve: 0.994203597905
Stacking Ensemble Area Under Precision Recall Curve: 0.982528271919
Stacking Ensemble MCC: 0.987672527116

Scores with validation data:

Accuracy of Stacking Ensemble with optimal hyperparameters on validation set:
0.995702885206
F-Score of Stacking Ensemble with optimal hyperparameters on validation set:
0.989928057554
Sensitivity of Stacking Ensemble with optimal hyperparameters on validation set:
0.994219653179
```

```
Specificity of Stacking Ensemble with optimal hyperparameters on validation set:
0.996102883866
Precision of Stacking Ensemble with optimal hyperparameters on validation set:
0.985673352436
ROC-Auc Score of Stacking Ensemble with optimal hyperparameters on validation
set: 0.995161268523
MCC of Stacking Ensemble with optimal hyperparameters on validation set:
0.987211809146
```

9. Generate ROC curves for the methods compared and combine these in a single plot. Comment on the accuracy results. Which methods give the best performance? Can you suggest other methods to further improve the accuracy?

[11]:
```
result_table = pd.DataFrame(columns=['classifiers', 'fpr','tpr','auc'])

optimal_classifiers = [
    KNeighborsClassifier(n_neighbors=opt_params["n_neighbors"]),
    RandomForestClassifier(n_estimators=opt_params["rf_estimators"],␣
 ↪random_state=42),
    SVC(kernel="rbf", C=opt_params["svm_c"],gamma=opt_params["svm_gamma"],␣
 ↪random_state=42, probability=True),
    AdaBoostClassifier(n_estimators=opt_params["ab_estimators"],␣
 ↪random_state=42),
    StackingClassifier(classifiers=classifiers_for_stacking, meta_classifier=lr)
]

for cls in optimal_classifiers:
    model = cls.fit(optimum_x_train_two, y_train_two)
    yproba = model.predict_proba(optimum_x_validation)[::,1]

    fpr, tpr, _ = roc_curve(y_validation,  yproba)
    auc = roc_auc_score(y_validation, yproba)

    result_table = result_table.append({'classifiers':cls.__class__.__name__,
                                        'fpr':fpr,
                                        'tpr':tpr,
                                        'auc':auc}, ignore_index=True)

result_table.set_index('classifiers', inplace=True)

fig = plt.figure(figsize=(8,6))

for i in result_table.index:
    plt.plot(result_table.loc[i]['fpr'],
             result_table.loc[i]['tpr'],
```

```
            label="{}, AUC={:.3f}".format(i, result_table.loc[i]['auc']))

plt.plot([0,1], [0,1], color='orange', linestyle='--')

plt.xticks(np.arange(0.0, 1.1, step=0.1))
plt.xlabel("False Positive Rate", fontsize=15)

plt.yticks(np.arange(0.0, 1.1, step=0.1))
plt.ylabel("True Positive Rate", fontsize=15)

plt.title('ROC Curve Analysis', fontweight='bold', fontsize=15)
plt.legend(prop={'size':13}, loc='lower right')

plt.show()
```



Random Forest, Support Vector Machine and AdaBoost are giving the best performances. Since the accuracy is already pretty high, I think there is no point of a suggestion.

10. Train the method that gives the most accurate predictions so far (i.e. the highest overall accuracy) on the original train set after applying the best feature selection and normalization strategy and compute predictions on the samples of the test set(s) for which the true labels are available. Report the same accuracy measures as in step 5.

[14]:
```
clf = RandomForestClassifier(n_estimators=opt_params["rf_estimators"],␣
↪random_state=42)

optimum_x_train = best_selector.fit_transform(X_train, y=y_train)
optimum_x_test = best_selector.fit_transform(X_test, y=y_test)
optimum_x_test2 = best_selector.fit_transform(X_test2, y=y_test2)

clf.fit(optimum_x_train, y_train)

preds = clf.predict(optimum_x_test)

print("Accuracy of RandomForest with optimal hyperparameters on test set: {}".
↪format(accuracy_score(y_test, preds)))
print("F-Score of RandomForest with optimal hyperparameters on test set: {}".
↪format(f1_score(y_test, preds)))
print("Sensitivity of RandomForest with optimal hyperparameters on test set: {}".
↪format(recall_score(y_test, preds)))
print("Specificity of RandomForest with optimal hyperparameters on test set: {}".
↪format(recall_score(y_test, preds,pos_label=0)))
print("Precision of RandomForest with optimal hyperparameters on test set: {}".
↪format(precision_score(y_test, preds)))
print("ROC-Auc Score of RandomForest with optimal hyperparameters on test set:␣
↪{}".format(roc_auc_score(y_test, preds)))
print("MCC of RandomForest with optimal hyperparameters on test set: {}\n\n".
↪format(matthews_corrcoef(y_test, preds)))

preds = clf.predict(optimum_x_test2)

print("Accuracy of RandomForest with optimal hyperparameters on test set2: {}".
↪format(accuracy_score(y_test2, preds)))
print("F-Score of RandomForest with optimal hyperparameters on test set2: {}".
↪format(f1_score(y_test2, preds)))
print("Sensitivity of RandomForest with optimal hyperparameters on test set2:␣
↪{}".format(recall_score(y_test2, preds)))
print("Specificity of RandomForest with optimal hyperparameters on test set2:␣
↪{}".format(recall_score(y_test2, preds,pos_label=0)))
print("Precision of RandomForest with optimal hyperparameters on test set2: {}".
↪format(precision_score(y_test2, preds)))
print("ROC-Auc Score of RandomForest with optimal hyperparameters on test set2:␣
↪{}".format(roc_auc_score(y_test2, preds)))
```

```
print("MCC of RandomForest with optimal hyperparameters on test set2: {}\n\n".
 ↪format(matthews_corrcoef(y_test2, preds)))
```

```
Accuracy of RandomForest with optimal hyperparameters on test set: 0.978986866792
F-Score of RandomForest with optimal hyperparameters on test set: 0.971971971972
Sensitivity of RandomForest with optimal hyperparameters on test set: 0.
 ↪998971193416
Specificity of RandomForest with optimal hyperparameters on test set: 0.
 ↪967513290018
Precision of RandomForest with optimal hyperparameters on test set: 0.946393762183
ROC-Auc Score of RandomForest with optimal hyperparameters on test set: 0.
 ↪983242241717
MCC of RandomForest with optimal hyperparameters on test set: 0.956078033949


Accuracy of RandomForest with optimal hyperparameters on test set2: 0.993334700574
F-Score of RandomForest with optimal hyperparameters on test set2: 0.984326018809
Sensitivity of RandomForest with optimal hyperparameters on test set2: 0.
 ↪996095656418
Specificity of RandomForest with optimal hyperparameters on test set2: 0.
 ↪992600285603
Precision of RandomForest with optimal hyperparameters on test set2: 0.972831267874
ROC-Auc Score of RandomForest with optimal hyperparameters on test set2: 0.
 ↪99434797101
MCC of RandomForest with optimal hyperparameters on test set2: 0.980204538224
```

11. Do literature review and find publications on the same topic. Which methods performed the best? Compare them with the methods you developed in this project. Can you improve your methods using the techniques implemented in the literature? Suggest ideas for improvement.

Publications

1. Accurate occupancy detection of an office room from light, temperature, humidity and CO2 measurements using statistical learning models. Luis M. Candanedo, VÃ©ronique Feldheim. Energy and Buildings. Volume 112, 15 January 2016, Pages 28-39.
2. Richardson, Ian & Thomson, Murray & Infield, David. (2008). A high-resolution domestic building occupancy model for energy demand simulations. Energy and Buildings. 40. 1560-1566. 10.1016/j.enbuild.2008.02.006.
3. S. Meyn, A. Surana, Y. Lin, S.M. Oggianu, S. Narayanan, T.A. Frewen, A sensor-utility-network method for estimation of occupancy in buildings, in: Decision and Control, 2009 held jointly with the 2009 28th Chinese Control Conference. CDC/CCC 2009. Proceedings of the 48th IEEE Conference on, IEEE, Shanghai, P.R. China, 2009, pp. 1494–1500.
4.] V.L. Erickson, Y. Lin, A. Kamthe, R. Brahme, A. Surana, A.E. Cerpa, M.D. Sohn, S. Narayanan, Energy efficient building environment control strategies using real-time occu-

pancy measurements, in: Proceedings of the first ACM workshop on embedded sensing systems for energy-efficiency in buildings, ACM, Berkeley, California, 2009, pp. 19–24.

5.] C. Liao, P. Barooah, An integrated approach to occupancy modeling and estimation in commercial buildings, in: American Control Conference (ACC), IEEE, Baltimore, MD, 2010, pp. 3130–3135.

Methods implemented in the literature

Linear Discrimant Analysis performed the best in the publications. Since this is a very well established problem, almost all methods that I have performed was already performed by other researchers. So, the results are very similiar because of that. I can definitely improve my methods using the tecniques implemented in the literature. For instance, I can use bootstrap sampling for evaluating just like in the first article or I could take advantage of principle component analysis. I could also optimize the parameters of linear discriminant analysis which gives a fairly good accuracy.