Comparison of classification algorithms on predicting breast cancer malignancy of cell clumps

Timo van Donselaar (s1019095) & Floris Rossel (s1010794)
12-01-2020

Abstract

This paper will investigate the relative performance of several classification algorithms on a breast cancer dataset, where the goal of the algorithm is to classify a patient as having a benign or malignant tumor. The classifiers tested are a decision tree, three types of naive Bayes classifiers and a multilayer perceptron (MLP).
The algorithms are trained and tested using k-fold cross validation, after which the performance is calculated using a cost function. The goal when finding the optimal model is to minimize the cost function. Ultimately the best algorithm will be selected based on the performance.
We also provide learning rates for each of the algorithms to give an idea how more training data affects the performance.
The performance of the Gaussian naive Bayes classifier and MLP are near-perfect, getting accuracies of 99.7% and 99.4% respectively.

Our approach

Before choosing the algorithms, we will visualize the data using histograms to get a sense of how the data is spread out. After that, we will choose the algorithms to use for classification, optimize the hyperparameters of those algorithms and finally calculate the accuracy and cost of each classification using a cost matrix. We think the use of cross-validation gives the most valid results, as this gives the generalized performance of the classifiers.
For the cost calculation, we use a much higher cost for false negatives than for false positives, because a patient that is diagnosed having a benign tumor when it is truly malignant is much worse than a wrongly diagnosed benign tumor. The values of the cost matrix remain open for adjustment for real-world use cases.
The algorithm with the lowest accuracy and/or cost has performed the best and will be recommended for future diagnostication of patients if it outperforms the models of previous research.

Application domain

Although we are analyzing the efficacy of the mentioned classification algorithms for medical diagnosis with respect to breast cancer, the scope of research extends beyond this particular type of cancer, and potentially other types of medical diagnoses as well. The data we used for our research is composed of cell clump characteristics, which can also be measured in other types of tumors in the body. This means our results can be taken into account for decision making when implementing data-mining-algorithm-based cancer diagnosis.
Our research could also stretch further outside of the cancer-domain, into other types of body abnormalities where pathological diagnosis is desirable. The performance of the algorithms will probably vary more outside of the domain of cancer, and it’s obvious that

these problems will require new hyperparameter tuning and learning for each specific type of abnormality or disease.

Data set

The data we will use is from the Wisconsin Breast Cancer Database. It can be found on the UCI Machine Learning Repository. The database was obtained from the University of Wisconsin Hospitals, in Madison, from Dr. William H. Wolberg. The data consists of 699 objects, that represent (the characteristics of) breast FNA's (fine needle aspirates), which contained epithelial cell clumps. The following nine characteristics were assessed and are represented as ordinal attributes: clump thickness, uniformity of cell size, uniformity of cell shape, cohesion of peripheral cells of the clump (marginal adhesion), single epithelial cell size, the proportion of single epithelial nuclei had no surrounding cytoplasm anymore (bare nuclei), blandness of nuclear chromatin (bland chromatin), normal nucleoli, and infrequent mitosis (mitosis). These characteristics form discrete numbers and have a range of 1 to 10. The FNA samples are either benign or malignant, which forms the class label of an object. Benign is represented with the number 2 and malignant with the number 4 in the database. All objects also have an id number.

Related previous work

Two reports, that used the Wisconsin breast cancer data, both published in 1990, are: 'Multisurface method of pattern separation for medical diagnosis applied to breast cytology'[1] and 'cancer diagnosis via linear programming'[2], both by W. H. Wolberg and O. L. Mangasarian. The second report refers to the first. They both describe a system for diagnosing breast cancer. This system was originally built with about 360 objects, which were the first objects in the Wisconsin breast cancer data, used in this project. For the diagnosis, multisurface pattern separation was used, which is a 'mathematical method for distinguishing between elements of two pattern sets' and which uses linear programming. [1] It is related to neural networks [2].

The system mentioned in 'cancer diagnosis via linear programming', correctly diagnosed 165 out of 166 cases, in a test period of 17 months. These test objects were later added to the data. According to the same report, larger numbers for the attributes generally indicate a higher probability of malignancy, but no single measurement can determine whether the object is malignant or not. [2]

In 'Multisurface method of pattern separation for medical diagnosis applied to breast cytology' the discussed system is compared to a decision tree, that gave 19 wrong classification on a total of 352 objects. In this report are also tests mentioned with models (or systems), trained on different amounts of samples. When all the data (369 samples) were used as training set, complete classification was achieved. When half of the data was as training set and the other half as test set, the error rate was 6.5%. With two third as training set and the other third as test set, this was 4.1%. From this is concluded that the accuracy increases with the size of the training set. [1]

In 1992 a report was published by K. P. Bennett & O. L. Mangasarian, 'Robust linear programming discrimination of two linearly inseparable sets', in which three linear programs

(of which two are multisurface methods) are compared, among others on the Wisconsin breast cancer data. The dataset then consisted of 566 points. When the presented model in this report that outperformed the other two, was trained on two third of the data and testes on the other third, it gave an error of 3.01% on the training data and 2.56% on the test data. [3]

Preparing the data / preprocessing

Missing data

The data is almost complete; only one attribute has missing values in the data. Of the 698 records, 16 data points contain missing attribute values for this attribute. We decided to remove these data points because we don't expect 16 missing records to have a significant impact on the classification performances.

Outlier detection & removal

Our data should be inspected for outliers that could potentially worsen the quality of the generated classification models. To detect outliers, we will use several data visualization techniques, and manually inspect the data this way.
After inspecting the visualization of the data set, we decided to not consider any value an outlier, since the data is all within a scale of 1-10 and has thus already been pre-processed. Since the data is from tumor cells, we expect there to be several data points that correspond to malignant cells with very different values than benign cells. This often binary separation will become clear in the visualization.

Visualization

Before starting to think about visualizing the data, we need to take into account the type of data. Since all attributes are ordinal values between 1 and 10, it makes sense to use histograms to indicate how often a value occurs. We also constructed 2D histograms of pairwise combinations of attributes to see if there exist interesting relationships between attributes. The 2D histograms show the distribution of the data on the basis of two attributes, by depicting a color grid. The number of records with certain values are represented by a color on a color scale. In the case of our data, a 2D histogram is always a 10 by 10 grid, because all attributes have a range from 1 to 10. Because the data is often concentrated on a small range of values, we used a logarithmic color scale to make the records outside of these concentrated data clumps more visible.
To indicate the difference between benign and malignant data, we split up the data and plotted the two groups on separately as well. This gives a view of how the classes are separated. It showed that objects with higher values are in general malignant, as stated by W. H. Wolberg and O. L. Mangasarian [2].
Since there are many combinations, we decided to only include some of the more interesting histograms in the appendix.
We also plotted the first two principal components after PCA analysis. When the data is colored according to the classes, it shows that the benign objects are close together, with some exceptions, while the malignant objects are more widespread.

Classification algorithm selection

Based on the type of data and the distribution on values, we thought a decision tree could work well, as there seem to be clear boundaries in many variables that separate the two classes benign and malignant. We also tested three types of a Naive Bayes classifiers and neural network, more specifically a multilayer perceptron (MLP). These latter models are renowned and can be used for complex data, so we believe they will give good results as well.
For all algorithms we use the Scikit-learn library.

Decision tree implementation

For this method we used the DecisionTreeClassifier module and the ShuffleSplit module for cross-validating the hyperparameters. We use a loop to test classification accuracies using different maximum tree depths between 2 and 15. We found that increasing the tree depth above 15 does not improve the accuracy on the test set. For each tree depth that is tested, ShuffleSplit will split the data in 10 random sets where one set is used for testing. The average accuracy after all 10 shuffles is assigned to the depth that has been used. To smooth out the accuracy graph with respect to the tree depth, we re-run the entire process 50 times, which means every depth is tested 50*10 (number of splits) times. The results are plotted in the section ‘Results and evaluation’.
We also tested different numbers of minimum samples required to split a node, and although we did not plot these, we concluded that a value of 30 was semi-optimal. The differences in accuracy after adding or subtracting ~10 to/from this value changes within the margin in which the accuracy changes between runs. Because of the randomness of the shufflesplit method, we can’t optimize this value to one point.

Naive Bayes implementation

We tested three different types of naive Bayes (NB) classifiers, namely the multinomial, gaussian and complement NB classifiers. The multinomial NB classifier is a suitable candidate because it works well with discrete features, which all of the variables are in our data set. The gaussian NB is not chosen for any particular reason, but is often used and works well in general. And finally, we included the complement NB classifier because it is suited for imbalanced data sets, which could be said about our data set where 239 of 683 records belong to the class ‘malignant’. However, other data sets could be more unbalanced, so we decided to test this classifier for a general idea of performance.

Multilayer perceptron implementation

We implemented the MLP using the MLPClassifier module. The program calculates the accuracy for a MLP with 1 and 2 layers, and for each of these between 2 and 15 hidden units per layer. After testing all three available solvers for weight optimization, we concluded that ‘lbfgs’ is far superior for this dataset, which is in correspondence with the following statement in the documentation; “For small datasets, however, ‘lbfgs’ can converge faster and perform better.”
For performance (read accuracy) measurements, we again use the ShuffleSplit method with 10 splits.

Learning rate calculation

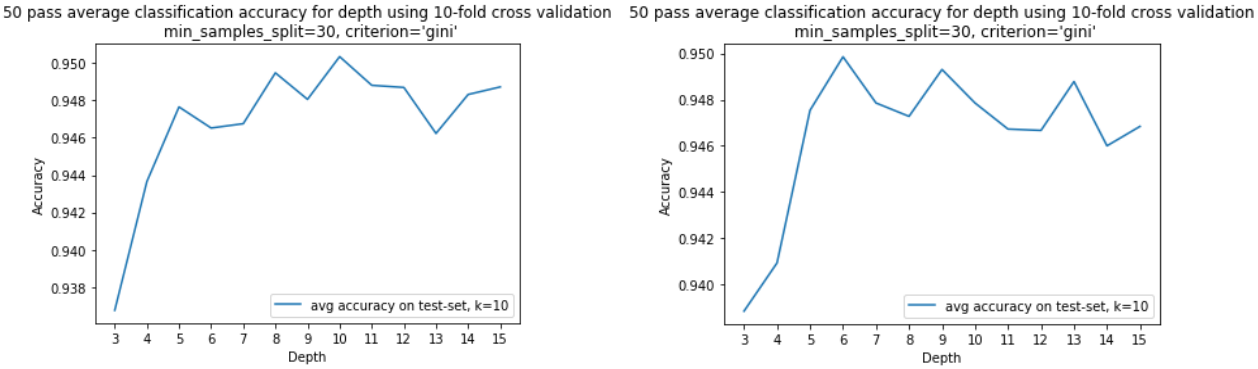
To obtain the learning rates for each classifier, we use increasingly sized portions of the dataset to feed into the classifiers. We decided to use ShuffleSplit for this, where we give the method the flag ‘test_size’ an increasing ratio from 0.05 to 0.99, (1 doesn’t work since the ShuffleSplit does not allow an empty training set). This number indicates the portion of the data normally used for testing, but we use it for the input of cross-validating the classifiers. The accuracy of each portion size and each classifier is plotted in one graph.

Results and evaluation

Hyperparameter tuning

Decision tree tuning:

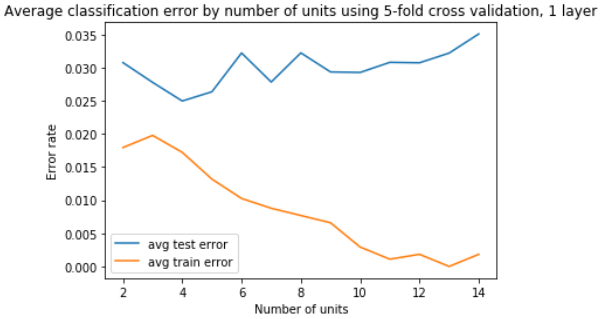
Tuning the maximum depth of DecisionTreeClassifier gives the following results (two runs):

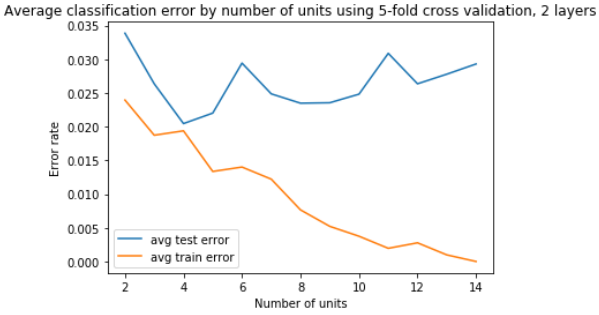


All runs of the same program gives a similar line, but the accuracy varies in a range of ~0.04. We conclude that the accuracy doesn’t change between a depth of 6 and 15. We picked 10 as the optimal maximum tree depth for the final comparison.

MLP tuning:

The most important hyperparameters of the MLPClassifier are the number of layers and units per layer. Because of time constraints we only tested one and two layers, because adding more layers needs more computing time. Based on the results, it doesn’t appear that adding more than two layers will help, since the accuracy scores barely improve going from one to two layers.
We tested all values between 2 and 15 as the number of units per layer. After 15 units, the accuracy on the training set did not seem to improve, as it was already at near-hundred percent accurate. What matters more is the accuracy on the test set, which doesn’t improve using more than 5 units.





The MLP appears to work best with 2 layers and 4 units per layer. We use these parameters in the final comparison.

Results of NB algorithms:

Error of multinomialNB on test set: 0.095
Error of complementNB on test set: 0.147
Error of GaussianNB on test set: 0.040

We can conclude that GaussianNB outperforms the other NB algorithms, so we will only use this one for the final comparison.

Using the optimal hyperparameters for each algorithm, we calculated the average accuracy and cost after 10 runs with different train-test splits. The cost matrix below contains the cost values of wrongly predicted records, where ‘true’ represents the class ‘malignant’, and ‘false’ the class ‘benign’.

Final comparison:

Cost matrix	Predicted class		
Actual class		True	False
	True	0	10
	False	1	0

Using 10-fold cross validation, we compute the cost (method in appendix) and accuracy for each fold, and then average the results.

Decision tree (d=10) results:
accuracy = 0.9260869565217391
average cost = 29.4, all costs: [2, 40, 25, 33, 20, 31, 33, 23, 23, 64]

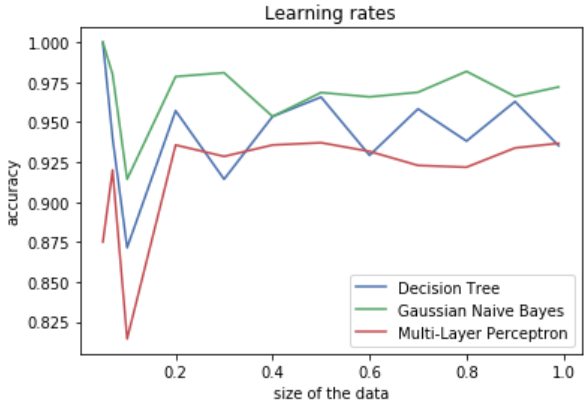
GaussianNB results:
accuracy = 0.9971014492753624
average cost = 2, all costs: [0, 0, 0, 0, 0, 0, 10, 10, 0, 0]

MLP results:
accuracy = 0.9942028985507246
average cost = 3.1, all costs: [0, 0, 0, 11, 0, 10, 0, 10, 0, 0]

Based on these results, we can conclude that both GaussianNB and MLP work equally well. After some runs, the GaussianNB gives a better score than MLP, but on average they barely differ in performance. the Decision tree classifier consistently underperforms compared to the other two. For future diagnosis we strongly recommend either of the two algorithms, but the preference goes to the gaussian naive Bayes classifier for its speed.
We did not expect all our algorithms to perform so well, especially the Gaussian naive-Bayes classifier, since it is extremely fast to train. The performance of the MLP is also unexpected, because in this classifier generally needs a lot more data than most classifiers, and our data set is not very large. It also performs well considering how it contains only 2 hidden layers with 4 units each. The high performance of the classifiers can be explained by the clear separation between benign and malignant data points, also mentioned in [1].

Learning rates

Learning rates show the performance of a classifier with respect to the amount of training data. The learning rates in our report are depicted in the figure on the right.
These results are the biggest surprise to us, because we did not expect only 10% of the data set (68 records) to be even close to sufficient for training any classifier.



Conclusions and future directions

According to Wolberg and Mangasarian, more data will increase the accuracy for the classifier in that report [1]. The learning rates in our research, however, do not show this for the classifiers we used after using more than ~100 samples.
We do not have any tips to improve performance of the tested classifiers, since the best ones perform near perfect.

References

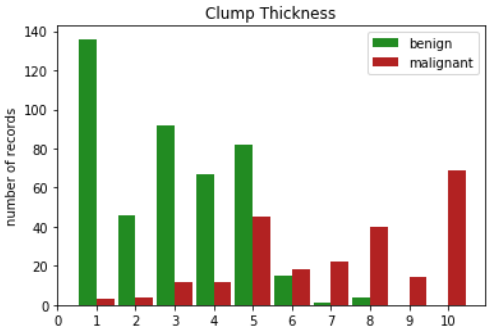
[1] W. H. Wolberg and O. L. Mangasarian, „Multisurface method of pattern separation for medical diagnosis," *Proceedings of the National Academy of Sciences of the United States of America,* vol. Vol. 87, pp. 9193-9196, 1990.

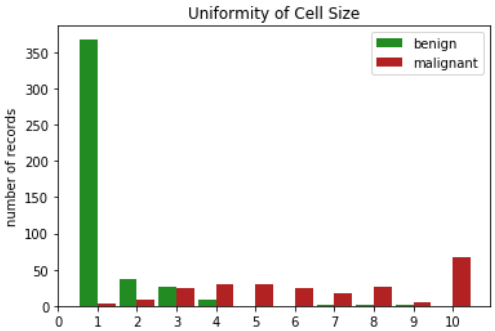
[2] O. Mangasarian and W. Wolberg, *Cancer Diagnosis via Linear Programming,* 1990.

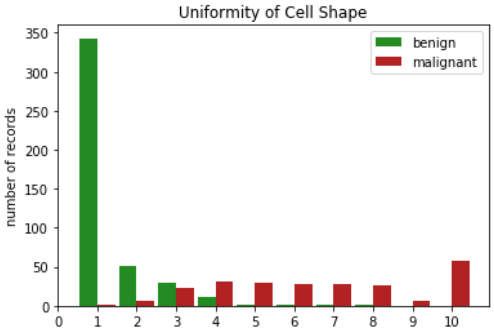
[3] K. P. Bennett and O. Mangasarian, „Robust linear programming discrimination of two linearly inseparable sets," *Optimization Methods and Software,* vol. Volume 1, pp. 23-34, 1992.

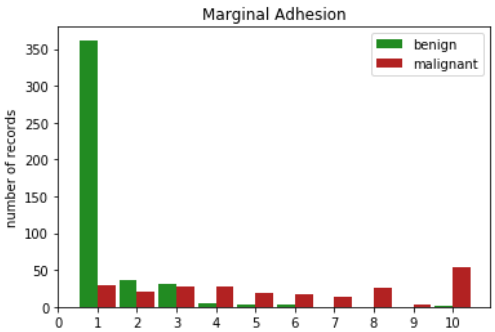
Appendix

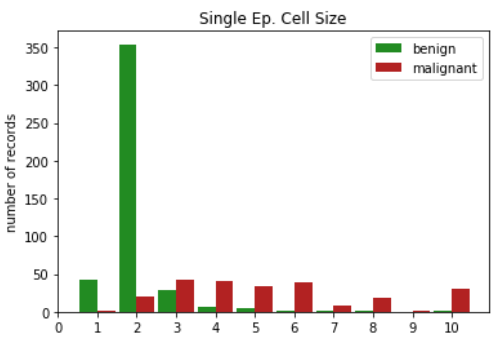
Histograms:

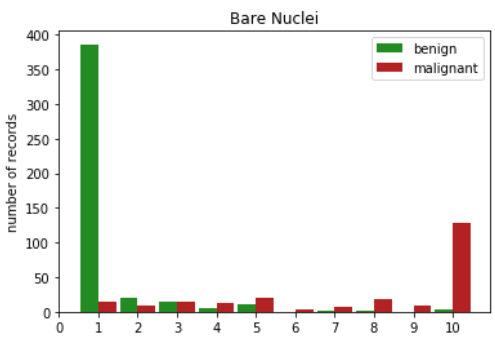


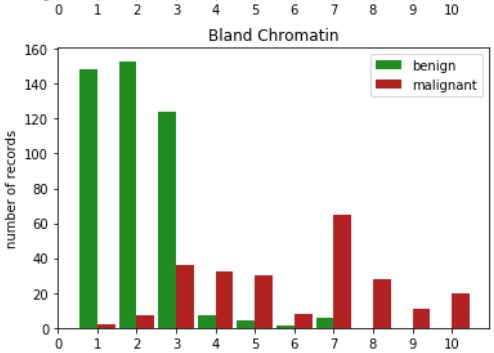


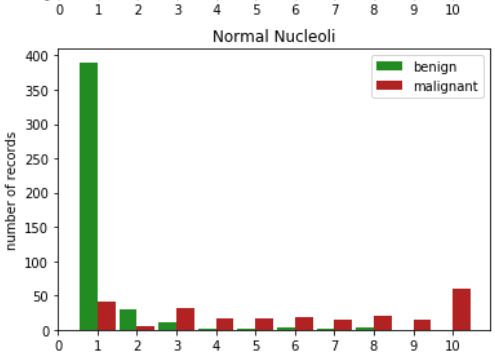


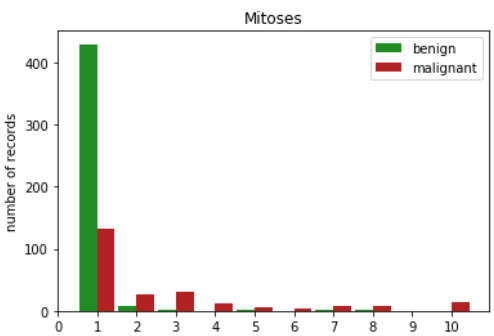


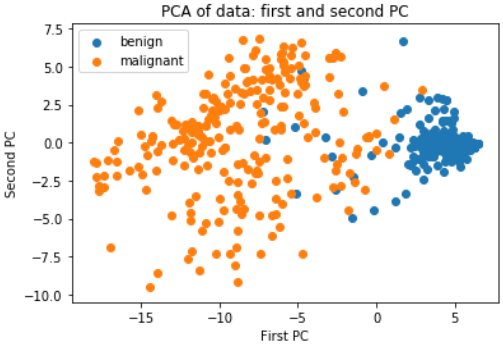




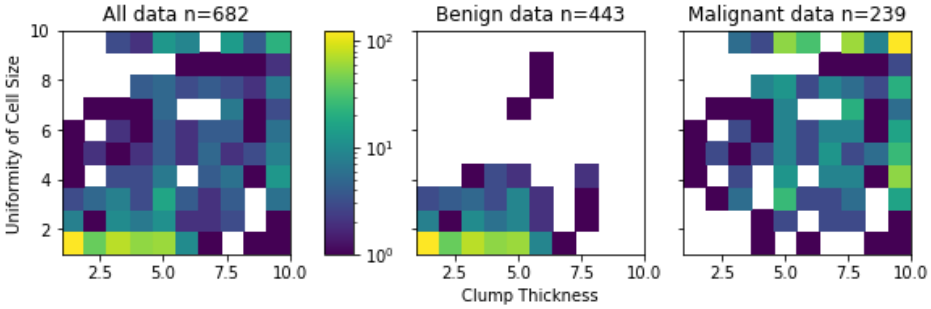




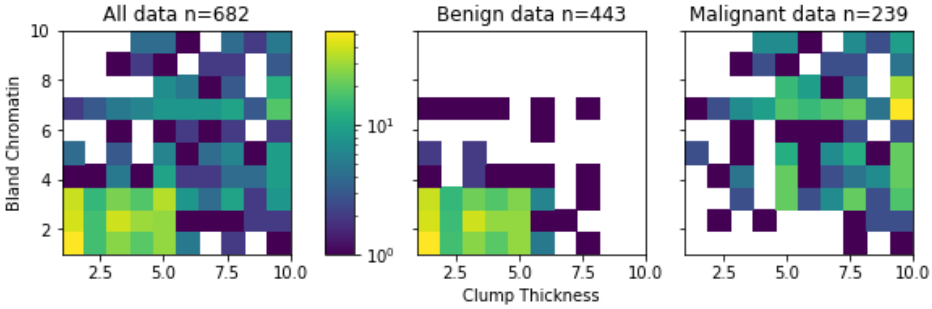




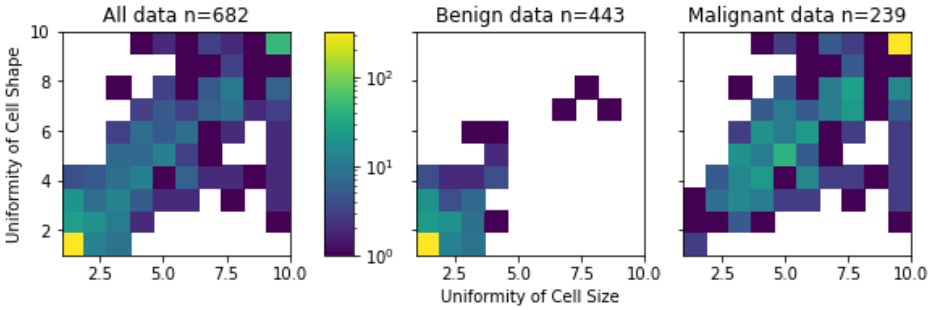
Clump Thickness vs Uniformity of Cell Size



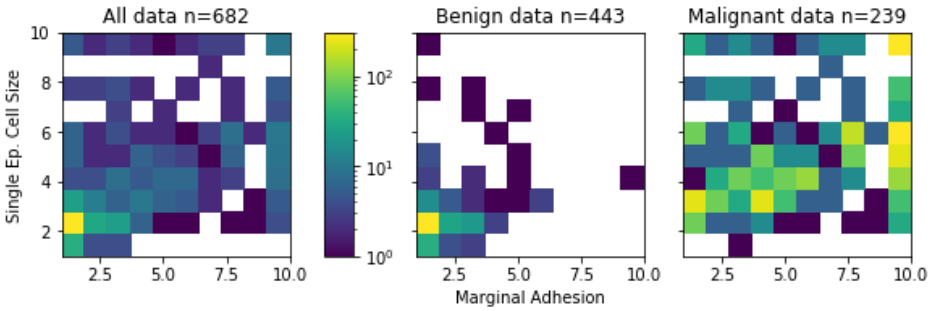
Clump Thickness vs Bland Chromatin



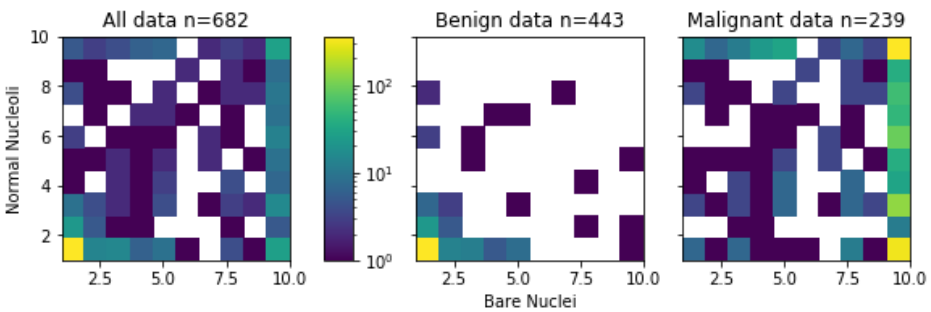
Uniformity of Cell Size vs Uniformity of Cell Shape



Marginal Adhesion vs Single Ep. Cell Size



Bare Nuclei vs Normal Nucleoli



Code

Importing data set and removing incomplete data points.

```
attr = ["ID", "Clump Thickness", "Uniformity of Cell Size", "Uniformity of Cell Shape", "Marginal Adhesion", "Single Ep. Cell Size", "Bare Nuclei", "Bland Chromatin", "Normal Nucleoli", "Mitoses", "Class"]
data = pd.read_csv("Data/breast-cancer-wisconsin.data")
data.columns = attr
data = data.apply(pd.to_numeric, errors='coerce')
data = data.dropna()
data = data.apply(pd.to_numeric, downcast='integer', errors='coerce')
X = data.iloc[:,1:10].values # Attributes
y = data.iloc[:,10].values   # Classes
```

Separating benign and malignant data.

```
benign_data = data[data.Class==False]
benign = data['Class']==False
malignant_data = data[data.Class==True]
malignant = data['Class']==True
```

Cost function

```
# Cost function for classification in binary class (true/false)
# parameters: list of true class labels, list of predicted labels,
# positive label, negative label, true positive cost,
# false positive cost, true negative cost, false negative cost.
def cost_score(y_true, y_pred, positive, negative, tpc=0, fpc=1, tnc=0, fnc=1):
    cost = 0
    for i, j in zip(y_true, y_pred):
        if (i!=j and i==positive):
            cost+=fnc
        if (i!=j and i==negative):
            cost+=fpc
        if (i==j and i==positive):
            cost+=tpc
        if (i==j and i==negative):
            cost+=tnc
    return cost
```

Principal Component Analysis

```
benign = data['Class']==False
malignant = data['Class']==True

μ = np.mean(X, axis=0)
Y = np.subtract(X,μ)
U, S, Vt = np.linalg.svd(Y)
Z = np.dot(Y, np.transpose(Vt))
plt.figure()
plt.scatter((Z[benign])[:,0], (Z[benign])[:,1], label=("benign"))
```

```
plt.scatter((Z[malignant])[:,0], (Z[malignant])[:,1], label=("malignant"))
plt.xlabel('First PC')
plt.ylabel('Second PC')
plt.title('PCA of data: first and second PC')
plt.legend()
plt.show()
```

Testing depths for decision tree classifier using cross-validation

```
avg_depth_acc = [] # stores avg. accuracy for each depth after cross validating
n_depth_acc = []
runs = 50
splits = 10
for n in range(runs): # run 4 times and get an average error graph
    depth_acc_test = [] # stores avg. accuracy for each depth after cross
validating
    ss = ShuffleSplit(n_splits=splits)
    for d in range(2, 15): # try different depths
        accuracies_test = []
        for train, test in ss.split(X):
            X_train, X_test = X[train], X[test] # RECORDS
            y_train, y_test = y[train], y[test] # LABELS
            dtc = tree.DecisionTreeClassifier(criterion='gini',
min_samples_split=30, max_depth=d)
            dtree = dtc.fit(X_train, y_train)
            pred_test = dtree.predict(X_test)
            acc_test = accuracy_score(pred_test, y_test)
            accuracies_test.append(acc_test)
        avg_accuracy_test = sum(accuracies_test) / len(accuracies_test)
        depth_acc_test.append(avg_accuracy_test)
    n_depth_acc.append(depth_acc_test)

for d in range(len(n_depth_acc[0])):
    accsForD = []
    for n in range(runs):
        accsForD.append(n_depth_acc[n][d])
    avg_depth_acc.append(np.mean(accsForD))

depth_axis = list(range(0,13)) # all depth values for the plot
plt.plot(depth_axis, avg_depth_acc, label='avg accuracy on test-set, k=10')
plt.xticks(np.arange(13), ('3', '4',
'5','6','7','8','9','10','11','12','13','14','15'))
plt.ylabel("Accuracy")
plt.xlabel("Depth")
plt.title(str(runs) + " pass average classification accuracy for depth using 10-
fold cross validation\n min_samples_split=30, criterion='gini'")
plt.legend()
plt.show()
```

Part of code for testing learning rates

```
data_sizes = [0.05, 0.07, 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 0.99]
```

```
scores_tree = []
scores_NB = []
scores_MLP = []
for size in data_sizes:
    ss = ShuffleSplit(n_splits=1, test_size=size)
    for train,test in ss.split(X):
        scores_tree.append(decisiontree(X[test], y[test]))
        scores_NB.append(GaussianNB(X[test], y[test]))
        scores_MLP.append(MLP(X[test], y[test]))
```

Naive Bayesian classifiers

```
#10-fold cross validation for three Bayesian classifiers
NBs = [bay.MultinomialNB(), bay.ComplementNB(), bay.GaussianNB()]
avg_accuracy_test = [] # one accuracy for each classifier
for i in NBs:
    skf = StratifiedKFold(n_splits=10, shuffle = True)
    accuracies_test = []
    for train, test in skf.split(X, y):
        X_train, X_test = X[train], X[test] # RECORDS
        y_train, y_test = y[train], y[test] # LABELS
        nbc = i
        nbc.fit(X_train, y_train)
        pred_test = nbc.predict(X_test)
        accuracies_test.append(accuracy_score(pred_test, y_test))
    avg_accuracy_test.append(sum(accuracies_test) / len(accuracies_test))
```

Testing hidden layer sizes in MLP

```
# Training MLP with 2 layers
acc_units_test = []
acc_units_train = []
err_units_test = []
err_units_train = []
for u in range(2,15):
    skf = StratifiedKFold(n_splits=10, shuffle = True)
    accuracies_test = []
    accuracies_train = []
    for train, test in skf.split(X, y):
        X_train, X_test = X[train], X[test] # RECORDS
        y_train, y_test = y[train], y[test] # LABELS
        accuracies_per_fold_test = []
        accuracies_per_fold_train = []
        for i in range(5):
            clf = MLPClassifier(solver='lbfgs', hidden_layer_sizes=(u,u)) # 2
layers with u units
            clf.fit(X_train, y_train)
            pred_test = clf.predict(X_test)
            pred_train = clf.predict(X_train)
            accuracies_per_fold_test.append(accuracy_score(pred_test, y_test))
            accuracies_per_fold_train.append(accuracy_score(pred_train,
```

```
y_train))
        accuracies_test.append(max(accuracies_per_fold_test))
        accuracies_train.append(max(accuracies_per_fold_train))
    avg_accuracy_test = sum(accuracies_test) / len(accuracies_test)
    avg_accuracy_train = sum(accuracies_train) / len(accuracies_train)
    acc_units_test.append(avg_accuracy_test)
    acc_units_train.append(avg_accuracy_train)
    err_units_test.append(1 - avg_accuracy_test)
    err_units_train.append(1 - avg_accuracy_train)
plt.plot(range(2, 15), err_units_test, label='avg test error')
plt.plot(range(2, 15), err_units_train, label='avg train error')
plt.ylabel("Error rate")
plt.xlabel("Number of units")
plt.title("Average classification error by number of units using 5-fold cross
validation, 2 layers")
plt.legend()
plt.show()
```

Final comparison of classifiers using optimal hyperparameters

```
# Using the best hyperparameters of MLP to classify our data, again using
ShuffleSplit
# (we want to know the generalized performance)

ss = ShuffleSplit(n_splits=10)
accuracies = []
costs = []
for train, test in ss.split(X):
    X_train, X_test = X[train], X[test] # RECORDS
    y_train, y_test = y[train], y[test] # LABELS
    dtc = tree.DecisionTreeClassifier(criterion='gini', min_samples_split=30,
max_depth=10)
    dtree = dtc.fit(X_train, y_train)
    pred_test = dtree.predict(X_test)
    accuracies.append(accuracy_score(pred_test, y_test))
    costs.append(cost_score(y_test, pred_test, True, False, 0, 1, 0, 10))
    # y_true, y_pred, positive, negative, tpc=0, fpc=1, tnc=0, fnc=1
avg_acc = stat.mean(accuracies)
avg_cost = stat.mean(costs)
print('Decision tree (d=10) results:')
print('acc = ' + str(avg_acc))
print('avg cost = ' + str(avg_cost) + ' all costs: ' + str(costs))

ss = ShuffleSplit(n_splits=10)
accuracies = []
costs = []
for train, test in ss.split(X):
    X_train, X_test = X[train], X[test] # RECORDS
    y_train, y_test = y[train], y[test] # LABELS
    bay.GaussianNB().fit(X_train, y_train)
    pred_test = clf.predict(X_test)
```

```
    accuracies.append(accuracy_score(pred_test, y_test))
    costs.append(cost_score(y_test, pred_test, True, False, 0, 1, 0, 10))
    # y_true, y_pred, positive, negative, tpc=0, fpc=1, tnc=0, fnc=1
avg_acc = stat.mean(accuracies)
avg_cost = stat.mean(costs)
print('GaussianNB results:')
print('acc = ' + str(avg_acc))
print('avg cost = ' + str(avg_cost) + ' all costs: ' + str(costs))

ss = ShuffleSplit(n_splits=10)
accuracies = []
costs = []
for train, test in ss.split(X):
    X_train, X_test = X[train], X[test] # RECORDS
    y_train, y_test = y[train], y[test] # LABELS
    best_mlp = MLPClassifier(solver='lbfgs', hidden_layer_sizes=(5)) # 2 layers
with u units
    best_mlp.fit(X_train, y_train)
    pred_test = clf.predict(X_test)
    accuracies.append(accuracy_score(pred_test, y_test))
    costs.append(cost_score(y_test, pred_test, True, False, 0, 1, 0, 10))
    # y_true, y_pred, positive, negative, tpc=0, fpc=1, tnc=0, fnc=1
avg_acc = stat.mean(accuracies)
avg_cost = stat.mean(costs)
print('MLP results:')
print('acc = ' + str(avg_acc))
print('avg cost = ' + str(avg_cost) + ' all costs: ' + str(costs))
```