Malignant Breast Cancer Prediction Through Use of a Support Vector Machine

Data Description

This dataset is from the UCI machine learning repository. It outlines 9 attributes (excluding sample ID) of breast cancer cases and then classifies the tumors as benign or malignant.

In [1]:
```python
import pandas as pd
import numpy as np
```

In [2]:
```python
# read in data
cancer = pd.read_csv("breast_cancer")
df = cancer.copy()
display(df.head())
print()
print("Shape: (699 rows total by 11 columns)")
print()
print("There are", df['Class'].value_counts()[2],"counts of benign tumors,", round(100 * (df['Class'].value_counts()
[2] / 699),2) ,"% of the samples, and", df['Class'].value_counts()[4],"counts of malignant tumors,",round(100 * (df
['Class'].value_counts()[4] / 699),2),"% of the samples.")
```

	ID	Thickness	UnifSize	UnifShape	MargAd	SglEpSize	BareNuc	BlandChrom	NormNuc	Mitosis	Class
0	1000025	5	1	1	1	2	1	3	1	1	2
1	1002945	5	4	4	5	7	10	3	2	1	2
2	1015425	3	1	1	1	2	2	3	1	1	2
3	1016277	6	8	8	1	3	4	3	7	1	2
4	1017023	4	1	1	3	2	1	3	1	1	2

```
Shape: (699 rows total by 11 columns)

There are 458 counts of benign tumors, 65.52 % of the samples, and 241 counts of malignant tumors, 34.48 % of the sam
ples.
```

Attribute(Domain)

The columns have shortened names for ease in coding, but here is the explanation of all independent variables:

1. Sample code number(id number)
2. Clump Thickness(1 - 10), cancer cells grouping together, which is why doctors check for lumps. A higher thickness is more indicative of malignancy.
3. Uniformity of Cell Size(1 - 10), cancer cells generally grow in unexpected shapes and sizes, so a lower value of uniformity is more indicative of malignancy.
4. Uniformity of Cell Shape(1 - 10)
5. Marginal Adhesion(1 - 10), malignant cells are less adhesive than regular cells, which is how you get metastasis, so a lower marginal adhesion indicates malignancy.
6. Single Epithelial Cell Size(1 - 10), epithelial cells are surface cells, whether external or internal. Since they are generally uniform, a higher SEC size is more indicative of malignancy.
7. Bare Nuclei(1 - 10), nuclei without cytoplasm. Lower bare nuclei indicates malignancy.
8. Bland Chromatin(1 - 10), uniformity in texture of the nucleus. Less uniform tends to be malignant.
9. Normal Nucleoli(1 - 10), the nucleolus is what copies the DNA, so unusual cells, such as cancer, typically have less normal nucleoli
10. Mitoses(1 - 10), an estimate on number of times mitosis has taken place. Since malignant cells reproduce faster, a higher number is indicative of malignancy.
11. Class:(2 for benign, 4 for malignant), classification, or, diagnosis.

Project

Question: Can we use a relatively small data set to accurately predict which biopsies will be malignant and which will be benign?

I will be using the dataset to create a Support Vector Machine to classify future biopsy samples as benign or malignant based on the 9 attributes. This is an example of supervised learning. If we didn't know how many classifications there were, then an unsupervised K-Means approach would be best.

My steps:

1. Clean the data, make sure the data types are usable, check for missing data, and either impute data or drop Na rows, whichever is more appropriate at that stage.
2. Split the data into training and testing sets
3. Build the SVM and apply the data
4. Check and interpret results

Cleaning

Cleaning steps:

1. Update the Class (benign:2, malignant:4) to (benign:0, malignant:1)
2. Check for missing values
3. Impute missing values if appropriate, otherwise, drop rows

In [3]:
```python
# Set classification from (benign:2, malignant:4) to (benign:0, malignant:1)
df['Class'] = np.where(df['Class'] == 2, 0, 1)
print(df['Class'].value_counts())
```

```
0    458
1    241
Name: Class, dtype: int64
```

We have confirmed that what was 2 is now 0, and what was 4 is now 1. I am not sure why 2 and 4 were the defaults when 0 and 1 are generally accepted as standard binary.

In [4]:
```python
# Check data types
print(df.dtypes)
```

```
ID             int64
Thickness      int64
UnifSize       int64
UnifShape      int64
MargAd         int64
SglEpSize      int64
BareNuc       object
BlandChrom     int64
NormNuc        int64
Mitosis        int64
Class          int64
dtype: object
```

Looks like there is something wrong with the BareNuc Column

In [5]:
```python
df['BareNuc'].isnull().values.any()
```

Out[5]: False

There are no Null values so I will cast to int

In [6]:
```python
try:
    df['BareNuc'] = df['BareNuc'].astype('int')

except Exception as e:
    print("The error is: ",e)
```

```
The error is:  invalid literal for int() with base 10: '?'
```

I tried to cast to int, but it says there are '?'s in the data instead of a regular integer or Null value, so a brief spot check into the data showed there was at least one '?' in the data. I will turn that into a Null value and see how many there are.

In [7]:
```python
# replace '?'s with None values
df['BareNuc'] = df['BareNuc'].replace({'?':None})

# count the Nones
na_sum = df['BareNuc'].isna().sum()
na_sum_malig = len(df.loc[(df['BareNuc'].isna()) & (df['Class'] == 1)])
na_sum_ben = len(df.loc[(df['BareNuc'].isna()) & (df['Class'] == 0)])
percent_na = (na_sum/df.shape[0]) * 100

print("Number of Na values in BareNuc:",na_sum)
print("Percentage of rows with Na values in data: {0:.2f}%".format(percent_na))
print("Number of Malignant Na values:", na_sum_malig)
print("Number of Benign Na values:", na_sum_ben)
```

```
Number of Na values in BareNuc: 16
Percentage of rows with Na values in data: 2.29%
Number of Malignant Na values: 2
Number of Benign Na values: 14
```

Since less than 5% of the data is None, I will impute the missing values, based on the mean BareNuc count of the other data points in their class. I don't want to take the mean of the whole dataset, since we are trying to distinguish between malignant and benign, and an average of those two together may not be as accurate as the two averages separately.

In [8]:
```python
# drop na rows and convert to int
df_no_nas = df.copy().dropna()
df_no_nas['BareNuc'] = df_no_nas['BareNuc'].astype('int')
print("After dropping the Na rows, the data frame shape is", df_no_nas.shape)

# get mean BareNuc for Malignant and Benign tumors
df_malig = df_no_nas[df_no_nas['Class'] == 1]
mean_BN_malig = round(df_malig['BareNuc'].mean(),2)
# print(df_malig.shape)

df_ben = df_no_nas[df_no_nas['Class'] == 0]
mean_BN_ben = round(df_ben['BareNuc'].mean(),2)
# print(df_ben.shape)

print("The mean bare nucleus count for malignant cells is:", mean_BN_malig)
print("The mean bare nucleus count for benign cells is:", mean_BN_ben)
print()

# check before
print("Before imputation")
display(df.loc[[23,40]])

# perform imputations
df['BareNuc'] = np.where((df['BareNuc'].isnull() & df['Class'] == 1), mean_BN_malig, df['BareNuc'])
df['BareNuc'] = np.where((df['BareNuc'].isnull()), mean_BN_ben, df['BareNuc'])


# check after
print("After imputation")
display(df.loc[[23,40]])

# cast as float
df['BareNuc'] = df['BareNuc'].astype('float')
```

```
After dropping the Na rows, the data frame shape is (683, 11)
The mean bare nucleus count for malignant cells is: 7.63
The mean bare nucleus count for benign cells is: 1.35

Before imputation
```

	ID	Thickness	UnifSize	UnifShape	MargAd	SglEpSize	BareNuc	BlandChrom	NormNuc	Mitosis	Class
23	1057013	8	4	5	1	2	None	7	3	1	1
40	1096800	6	6	6	9	6	None	7	8	1	0

```
After imputation
```

	ID	Thickness	UnifSize	UnifShape	MargAd	SglEpSize	BareNuc	BlandChrom	NormNuc	Mitosis	Class
23	1057013	8	4	5	1	2	7.63	7	3	1	1
40	1096800	6	6	6	9	6	1.35	7	8	1	0

Above we can see line 23, a malignant tumor, has been imputed to 7.63. Line 40, a benign tumor, has been imputed to 1.35

Analysis

Now that the data has been cleaned, we can begin building the SVM

Steps:

1. Split independent and dependent variables
2. Split 75/25 training/testing
3. Build the Support Vector Machine
4. Apply the training and testing sets
5. Run results

I will start by splitting the data 75/25 training/testing.

In [9]:
```python
# Remove ID since it isn't a cell attribute, and split between independent and dependent variables
df.columns
ind_var = np.asarray(df[['Thickness', 'UnifSize', 'UnifShape', 'MargAd', 'SglEpSize', 'BareNuc', 'BlandChrom', 'Norm
Nuc', 'Mitosis']])
dep_var = np.asarray(df['Class'])
```

In [10]:
```python
# split into training and testing sets
from sklearn.model_selection import train_test_split

ind_var_train, ind_var_test, dep_var_train, dep_var_test = train_test_split(ind_var, dep_var, test_size = 0.25, rand
om_state = 4)
# https://www.geeksforgeeks.org/how-to-do-train-test-split-using-sklearn-in-python/

print('Independent variable training set:')
print(ind_var_train.shape)
print()
print('Dependent variable training set:')
print(dep_var_train.shape)
print()
print('Independent variable testing set:')
print(ind_var_test.shape)
print()
print('Dependent variable testing set:')
print(dep_var_test.shape)
```

```
Independent variable training set:
(524, 9)

Dependent variable training set:
(524,)

Independent variable testing set:
(175, 9)

Dependent variable testing set:
(175,)
```

As expected, the dependent variable arrays are one dimensional. And we can see that 524 rows is 75% of 699, with the remaining 175 as test rows.

In [11]:
```python
from sklearn import svm

# set up the actual machine
classifier = svm.SVC(kernel = 'rbf', gamma = 'scale', C = 2)

# put data into the machine and get a prediction to compare to the dependent variable test set
classification_predict= classifier.fit(ind_var_train, dep_var_train).predict(ind_var_test)
```

A support vector machine is a form of supervised classification machine learning. It separates data with a hyperplane, and, in the case of a hard classifier, designates everything on one side as type 0, and everything on the other side as type 1, like so:

image.png

Different kernels do different things. For example, if data cannot be separated by a linear hyperplane, changing the kernel may allow for a polynomial hyperplane. In this instance, after trying a few different kernels, the Radial Basis Function (rbf) kernel worked best. Using the poly kernel made the model much less precise and less accurate, and missed 13 Malignant tumors, as opposed to the linear kernel's 1. The rbf kernel found one more true malignant tumor than the linear, but found an extra false positive as well.

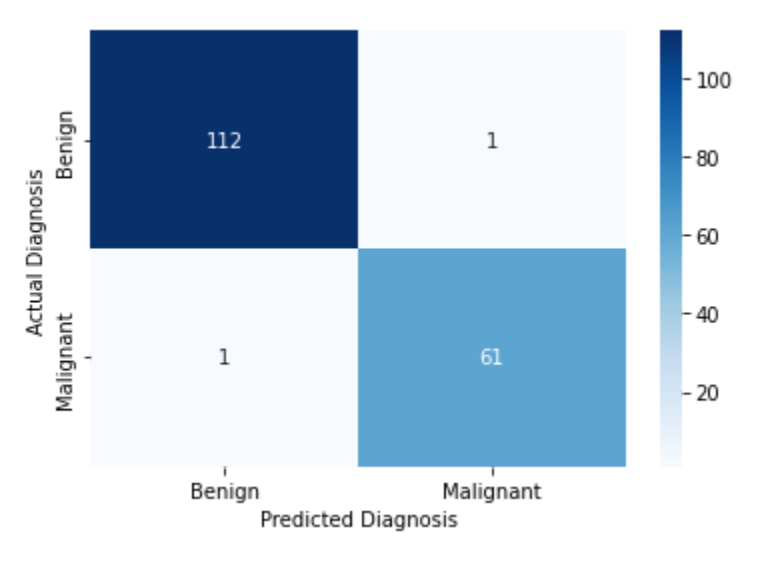
The gamma parameter determines the weight or influence of the training rows. The default gamma value gets the best results for this data set.

The C parameter determines the cost of misclassifying data. A lower value of C will maintain a large margin, or distance between sides of the hyperplane. The C value will be adjusted when determining if false positives are more dangerous, or false negatives are. Lowering the C value by a factor of 1000 tanks the results as margins become too wide. All predictions become benign, obviously not ideal. Raising the C parameter even by a factor of 10,000 does not change the result, meaning that there is enough separation between the data that smaller margins do not affect classification. It appears the ideal C value is no less than 1.5.

Results

In [12]:
```python
from sklearn.metrics import classification_report, confusion_matrix
import seaborn as sns

print("Classification Report:")
print(classification_report(dep_var_test, classification_predict))
print()
print("Confusion Matrix:")

# confusion matrix
cf_matrix = confusion_matrix(dep_var_test, classification_predict)
cf_formatted = sns.heatmap(cf_matrix, annot = True, fmt = '.0f', cmap = 'Blues')

# x labels
cf_formatted.set_xlabel("Predicted Diagnosis")
cf_formatted.xaxis.set_ticklabels(['Benign', 'Malignant'])

# y labels
cf_formatted.set_ylabel("Actual Diagnosis")
cf_formatted.yaxis.set_ticklabels(['Benign', 'Malignant'])


print("True Benigns:", cf_matrix[0][0])
print("False Malignants:", cf_matrix[0][1])
print("False Benigns:", cf_matrix[1][0])
print("True Malignants:", cf_matrix[1][1])
```

```
Classification Report:
              precision    recall  f1-score   support

           0       0.99      0.99      0.99       113
           1       0.98      0.98      0.98        62

    accuracy                           0.99       175
   macro avg       0.99      0.99      0.99       175
weighted avg       0.99      0.99      0.99       175


Confusion Matrix:
True Benigns: 112
False Malignants: 1
False Benigns: 1
True Malignants: 61
```



The classification report goes over a few different metrics.

Precision = True Positive / (True Positive + False Positive)

This is the machine's ability to correctly label negative samples. The closer to 1, the better.

Recall = True Positives / (True Positive + False Negative)

This is the machine's ability to correctly label positive samples. The closer to 1, the better.

F1 Score = 2 * (Precision * Recall) / (Precision + Recall)

The harmonic mean of precision and recall. It is a measure of accuracy rather than precision. The closer to 1, the better.

Support = actuals for the test variables, or, the "horizontal sum" in the confusion matrix. The negative, 0, or benign support value is 113, derived from the horizontal addition of 112 and 1.

The confusion matrix results are interpreted as True meaning the prediction and the actual result match, False meaning they did not match, Benign, denoted by a 0 in the data, and referred to medically as "Negative", and Malignant, denoted by a 1 in the data, or "Positive", medically. There are 112 True Benign results and 1 False Benign result. That means the machine said there are 113 Benign results, and there are actually 112, and the 1 incorrect prediction was actually Malignant. There are 61 True Malignants, and 1 False Malignant, meaning the prediction was 62 Malignant tumors, and there were only 61.

Conclusion

Question: Can we apply a relatively small data set to a Support Vector Machine in order to accurately predict which breast cancer biopsies will be malignant and which will be benign?

Answer: Yes, applying the provided data set from UCI to a Support Vector Machine, we are able to predict which breast cancer biopsies will be malignant and which will be benign with an accuracy of 99%.

Of course, it would be great for the machine to get it 100% right of the time; no one would be misdiagnosed. However, we want to avoid overfitting the model to this limited training data. If we had 6 million samples from all over the world and a dozen more unique attributes, then overfitting may not be a concern, as that would be a far more representative sample. The current dataset could be representative, or it could be all be from 1 clinic. I cannot make that determination here.

Regarding the SVM parameters, the C parameter is what will be altered to change the number of false positives and false negatives. It is up to the diagnosing party which is more dangerous, over-diagnosing a healthy person, or under-diagnosing a sick person.

With more time, I would run a logistic regression model on the data and see if variable selection would be appropriate. It may be that not all 9 independent variables are necessary for determining likelihood of a tumor becoming malignant in the future.

Overall, a 98% true positive rate is fantastic. According to the American Cancer Society, most mammograms show a 7/8, or 87.5% true positive rate. The issue isn't with the models generally, but with data collection. Tumor data is harder to collect in denser tissue because MRIs are not the default test method. That being said, all data in this set was collected by fine needle aspiration (biopsy). The ability for the machine to classify with such precision and accuracy means the two types of cells are generally distinct from one another.

Going forward, I believe monitoring tumors as early as possible and tracking what becomes malignant and what doesn't will be most effective for prevention and treatment. Using historical data on tumors, we can apply logistic regression to predict what factors make tumors more likely to become malignant at earlier stages, allowing more time to remove the tumors, and hopefully reduce the need for chemotherapy, and save lives.