Comparing classification methods to correctly identify cases of melanoma

1 Introduction:

Melanoma is the most dangerous type of skin cancer which affects the cells that produce melanin. It commonly develops as new pigmentated growth on the skin or from an existing nevus, commonly known as a mole. Melanoma is among the most diagnosed tumors for all age groups below 30 and it is highly dangerous if left untreated for a long period of time. [1]

A trained dermatologist can determine if odd looking nevi is dangerous or not, however the time of medical doctors is limited and not everyone is able to afford good healthcare. Thus, having machine learning models which can reliably diagnose melanoma would prevents countless deaths every year.

This model will rely solely on textual information about the nevus and not images, thus in a real-world application there would still be the need for someone to extract the data from photos. However, this is substantially easier and it does not require the need for health professional.

2 Problem Formulation

The main problem I am trying to solve is classification of nevi into a normal nevus and melanoma. The data that will be used is information about nevi that have been diagnosed by hospital staff and has already been classified.

The features that will be used are the features of the nevus. They are categorical data.

Asymmetry – symmetric, symmetric on 1-axis, asymmetric

Presence of dots – absent, atypical, typical

Presence of pigment networks – atypical, typical

Presence of streaks – absent, present

Presence of regression areas – absent, present

Presence of blue-whitish veil – absent, present

Colors present – one or more of white, red, light brown, dark brown, blue-gray, black

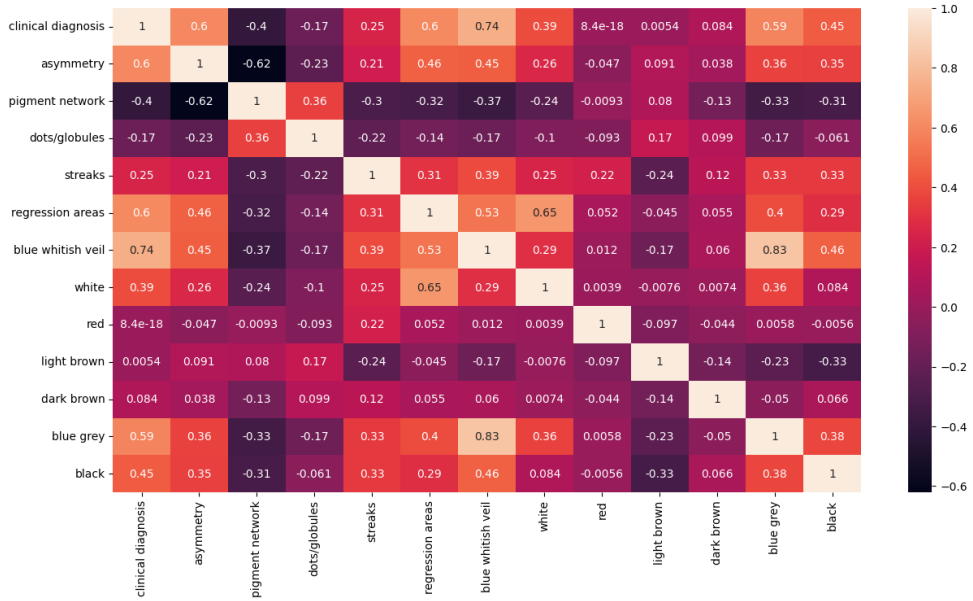
The label is binary, either the nevus is normal (0) or it is melanoma. (1)

The dataset can be found on https://www.fc.up.pt/addi/ph2%20database.html. It includes 200 datapoints each with 13 columns with no missing values. [2]

3.1 Data preprocessing

The label in the dataset has been classified into the categories: typical nevus, atypical nevus and melanoma. For the purpose of this project, we will combine typical and atypical nevi in one category (0) since they are both normal and not a medical condition like melanoma (1).

Instead of using the A, AT, T, P categories for absent, atypical, typical and present features we will use the 0,1,2,3 categories. The reason why we are not one-hot encoding, is that the data is



ordinal, so atypical features are more present than absent features, typical are more present than atypical ones etc. For the presence of different colors we will use binary data, 1 if present and 0 otherwise. Then we plot the correlation matrix between the features and the label, in order to see which of the columns present in the dataset are useful and which are not.

From the correlation matrix we can see that features such as: presence of red, light brown and dark brown color have small (less than 0.1) absolute correlation with the label and thus using them might result in overfitting without substantially improving the model. We will use all the other labels, thus in total 9 features.

3.2 Model

Since there is a strong linear correlation between most of the features and the label, using linear models would work sufficiently well. We will be using Logistic Regression and Support Vector Machines, as they are simple classifying methods that use linear maps.[3][4] Another method we will use is Decision Trees, since there is a large amount of categorical data present.[5]

For Logistic Regression we will use logistic loss and for SVM we will use hinge loss which are the default loss functions of each sklearn class.

$$Hinge\ loss \Rightarrow \ \mathrm{l(y)} = \max(0, 1 - t * y)$$

For Decision Trees we will use gini impurity.

The data points will be split randomly using the test_train_split function in the following way: 50% will be used for training, 30% for testing, and 20% for validation. Training is the most important step thus it requires the most amount of data. Validation data will be used to optimize the methods and testing will be used to find the optimal method.

In disease diagnostics it is most important to maximize the number of correct positives, thus aside from the accuracy of the model, an important metric we are going to use is the recall.

$$Recall = \frac{True\ Postive}{True\ Positive+False\ Negative} \qquad Accuracy\ = \frac{True\ Positve+True\ Negative}{TP+FN+TN+FP}$$

4 Results

4.1 Logistic Regression

After running the code we can see that the values for accuracy and recall are as follows:

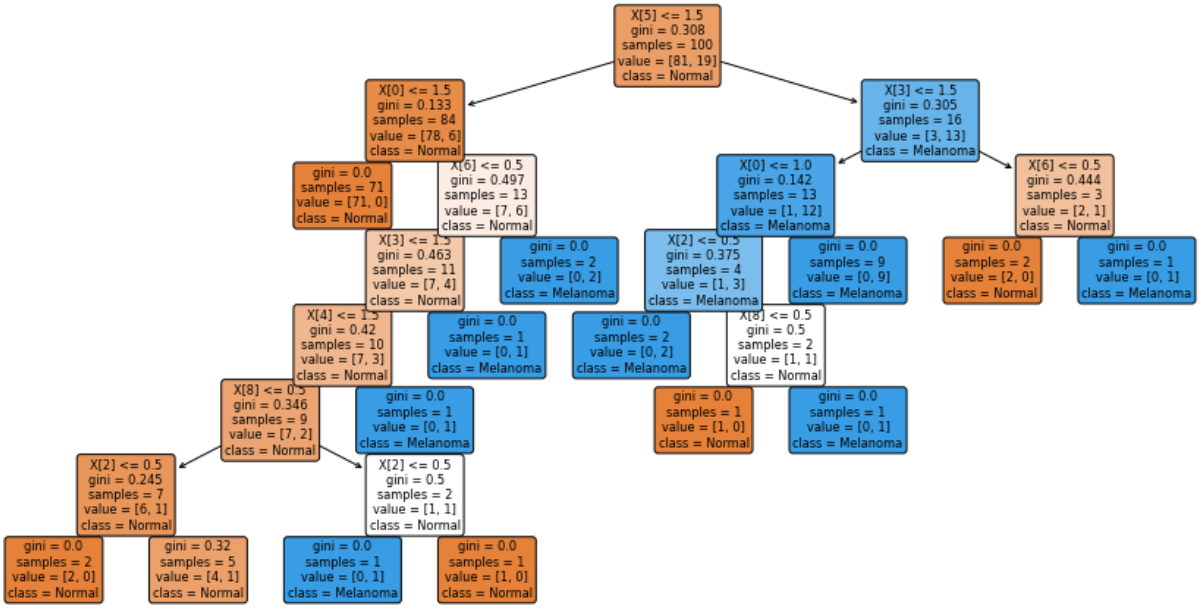
```
Training Score
Accuracy: 0.94
========================================
Validation Score
Accuracy: 0.92
Recall: 0.6666666666666666
========================================
Testing Score
Accuracy: 0.92
Recall: 0.8333333333333334
========================================
```

4.2 Support vector machines

The values we get by using support vector machines are as follows:

```
Training Score
Accuracy: 0.94
========================================
Validation Score
Accuracy: 0.9333333333333333
Recall: 0.75
========================================
Testing Score
Accuracy: 0.925
Recall: 0.7777777777777778
========================================
```

4.3 Decission tree




```
Training Score
Accuracy: 0.99
========================================
Validation Score
Accuracy: 0.9166666666666666
Recall: 0.75
========================================
Testing Score
Accuracy: 0.8
Recall: 0.3333333333333333
========================================
```

4.4 Chosen method

Finally we can compare our methods. Decision trees are doing the worst in both accuracy and recall. This might be since we haven't optimized the tree to reduce its size which might increase the accuracy of the tree. The 99% accuracy in the training score also tells us that there might be an overfitting problem.

Then we can see that Support vector machines and Linear regression have similar accuracies with SVM having slightly better accuracy. However in the testing set we can see that Linear regression has higher recall and since there is only a slight difference in accuracy, Linear Regression is the best model.

5 Conclusion

In the end we have reached 92% accuracy and 83.3% recall. There is still a lot of improvement to do as this means 1 in 5 persons with melanoma doesn't get a correct diagnosis. In order to increase accuracy we can further optimize the methods. Another thing that would reduce the error would be more data, specifically more data with a positive diagnosis, as only 40 of our 200 data points have melanoma and thus it is a relatively small sample.

6 Biblography

[1] https://www.mayoclinic.org/diseases-conditions/melanoma/symptoms-causes/syc-20374884

[2] https://www.fc.up.pt/addi/ph2%20database.html

[3] https://scikit-learn.org/stable/modules/svm.html

[4]https://scikitlearn.org/stable/modules/generated/sklearn.linear_model.LogisticRegression.html

[5] https://scikit-learn.org/stable/modules/tree.html

7 Appendic

The changed dataset which is used in the code can be found here

https://drive.google.com/uc?id=1Nhk2eXQNOae54QXNrEcCHsiOjJAraBuJ&export=download

The code can be found on the next page

Untitled1

March 31, 2022

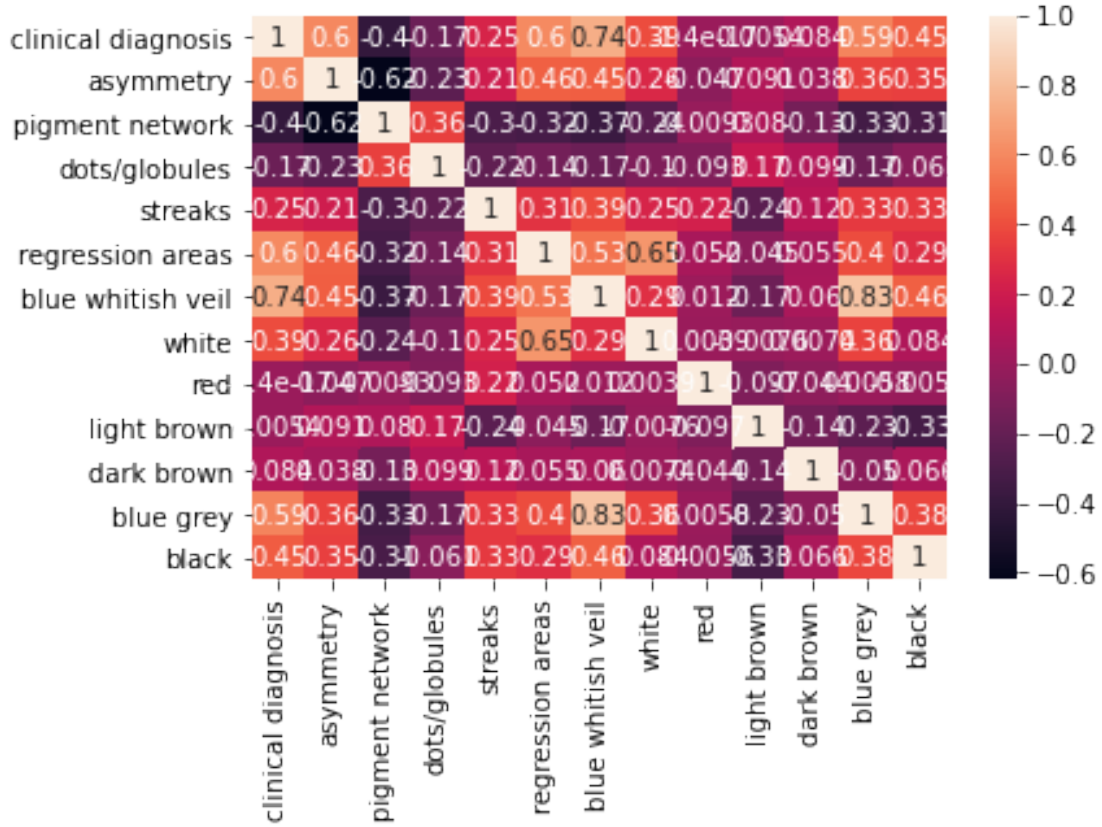
[62]:
```
import numpy as np
import pandas as pd
import seaborn as sn
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression

from sklearn.svm import SVC
from sklearn.tree import DecisionTreeClassifier, plot_tree
```

[91]:
```
df = pd.read_csv('document.csv')
X = df[["asymmetry","pigment network","dots/globules","streaks","regression␣
 ↪areas","blue whitish veil","white","blue grey","black"]].to_numpy()
y = df["clinical diagnosis"].to_numpy()
df.head

X_train, X_rem, y_train, y_rem = train_test_split(X, y, test_size=0.5,␣
 ↪random_state=13)
X_val, X_test, y_val, y_test = train_test_split(X_rem, y_rem, test_size=0.4,␣
 ↪random_state=23)
```

[92]:
```
def results(name, clf, X_train, X_val, X_test, y_train, y_val, y_test):
    y_train_pred = clf.predict(X_train)
    y_val_pred = clf.predict(X_val)
    y_test_pred = clf.predict(X_test)

    train_acc = accuracy_score(y_train, y_train_pred)
    val_acc = accuracy_score(y_val, y_val_pred)
    test_acc = accuracy_score(y_test, y_test_pred)

    test_mat = confusion_matrix(y_test, y_test_pred)
    tn, fp, fn, tp = test_mat.ravel()
    test_rec = tp / (tp+fn)

    val_mat = confusion_matrix(y_val, y_val_pred)
    tn, fp, fn, tp = val_mat.ravel()
    val_rec = tp / (tp+fn)
```

```
    print(name)
    print("="*40)
    print("Training Score")
    print("Accuracy: " + str(train_acc))
    print("="*40)
    print("Validation Score")
    print("Accuracy: " + str(val_acc))
    print("Recall: " + str(val_rec))
    print("="*40)
    print("Testing Score")
    print("Accuracy: " + str(test_acc))
    print("Recall: " + str(test_rec))
    print("="*40)
    print("\n")
```

[93]:
```
#Correlation Matrix
corrMatrix = df.corr()
sn.heatmap(corrMatrix, annot=True)
plt.show()
```



[94]:
```
#Logistic Regression
clf_lr = LogisticRegression().fit(X_train, y_train)  #fit the training data to␣
 ↪the model


results("Logistic Regression",clf_lr, X_train, X_val, X_test, y_train, y_val,␣
 ↪y_test)
```

```
Logistic Regression
========================================
Training Score
Accuracy: 0.94
========================================
Validation Score
Accuracy: 0.9333333333333333
Recall: 0.75
========================================
Testing Score
Accuracy: 0.9
Recall: 0.7777777777777778
========================================
```

[95]:
```
#Suppot vector machines
clf_svc = SVC()
clf_svc.fit(X_train,y_train)
results("Support Vector Machines",clf_svc, X_train, X_val, X_test, y_train,␣
 ↪y_val, y_test)
```

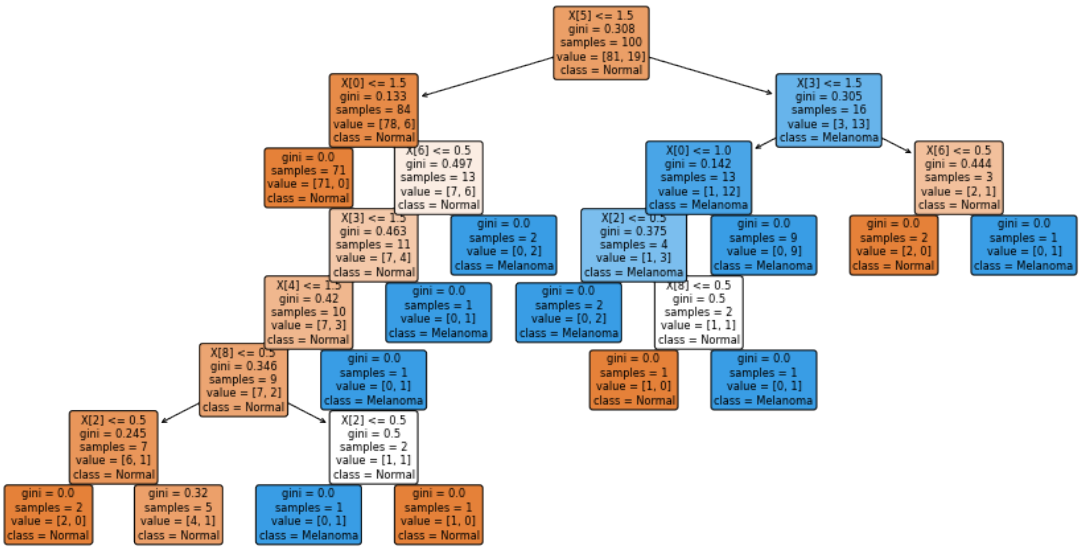
```
Support Vector Machines
========================================
Training Score
Accuracy: 0.94
========================================
Validation Score
Accuracy: 0.9333333333333333
Recall: 0.75
========================================
Testing Score
Accuracy: 0.925
Recall: 0.7777777777777778
========================================
```

[98]:
```
clf_tr = DecisionTreeClassifier()
```

[99]:
```
plt.figure(figsize = (15, 7.5))
plot_tree(tree, filled = True, rounded = True, class_names=["Normal",␣
 ↪"Melanoma"])
```

[99]:
```
[Text(467.7352941176471, 382.21875, 'X[5] <= 1.5\ngini = 0.308\nsamples =
100\nvalue = [81, 19]\nclass = Normal'),
 Text(295.4117647058823, 331.25625, 'X[0] <= 1.5\ngini = 0.133\nsamples =
84\nvalue = [78, 6]\nclass = Normal'),
 Text(246.1764705882353, 280.29375, 'gini = 0.0\nsamples = 71\nvalue = [71,
0]\nclass = Normal'),
 Text(344.6470588235294, 280.29375, 'X[6] <= 0.5\ngini = 0.497\nsamples =
13\nvalue = [7, 6]\nclass = Normal'),
 Text(295.4117647058823, 229.33124999999998, 'X[3] <= 1.5\ngini = 0.463\nsamples
= 11\nvalue = [7, 4]\nclass = Normal'),
 Text(246.1764705882353, 178.36875, 'X[4] <= 1.5\ngini = 0.42\nsamples =
10\nvalue = [7, 3]\nclass = Normal'),
 Text(196.94117647058823, 127.40625, 'X[8] <= 0.5\ngini = 0.346\nsamples =
9\nvalue = [7, 2]\nclass = Normal'),
 Text(98.47058823529412, 76.44375000000002, 'X[2] <= 0.5\ngini = 0.245\nsamples
= 7\nvalue = [6, 1]\nclass = Normal'),
 Text(49.23529411764706, 25.48124999999999, 'gini = 0.0\nsamples = 2\nvalue =
[2, 0]\nclass = Normal'),
 Text(147.70588235294116, 25.48124999999999, 'gini = 0.32\nsamples = 5\nvalue =
[4, 1]\nclass = Normal'),
 Text(295.4117647058823, 76.44375000000002, 'X[2] <= 0.5\ngini = 0.5\nsamples =
2\nvalue = [1, 1]\nclass = Normal'),
 Text(246.1764705882353, 25.48124999999999, 'gini = 0.0\nsamples = 1\nvalue =
[0, 1]\nclass = Melanoma'),
 Text(344.6470588235294, 25.48124999999999, 'gini = 0.0\nsamples = 1\nvalue =
[1, 0]\nclass = Normal'),
 Text(295.4117647058823, 127.40625, 'gini = 0.0\nsamples = 1\nvalue = [0,
1]\nclass = Melanoma'),
 Text(344.6470588235294, 178.36875, 'gini = 0.0\nsamples = 1\nvalue = [0,
1]\nclass = Melanoma'),
 Text(393.88235294117646, 229.33124999999998, 'gini = 0.0\nsamples = 2\nvalue =
[0, 2]\nclass = Melanoma'),
 Text(640.0588235294117, 331.25625, 'X[3] <= 1.5\ngini = 0.305\nsamples =
16\nvalue = [3, 13]\nclass = Melanoma'),
 Text(541.5882352941177, 280.29375, 'X[0] <= 1.0\ngini = 0.142\nsamples =
13\nvalue = [1, 12]\nclass = Melanoma'),
 Text(492.3529411764706, 229.33124999999998, 'X[2] <= 0.5\ngini = 0.375\nsamples
= 4\nvalue = [1, 3]\nclass = Melanoma'),
 Text(443.11764705882354, 178.36875, 'gini = 0.0\nsamples = 2\nvalue = [0,
2]\nclass = Melanoma'),
 Text(541.5882352941177, 178.36875, 'X[8] <= 0.5\ngini = 0.5\nsamples = 2\nvalue
```

```
= [1, 1]\nclass = Normal'),
 Text(492.3529411764706, 127.40625, 'gini = 0.0\nsamples = 1\nvalue = [1,
0]\nclass = Normal'),
 Text(590.8235294117646, 127.40625, 'gini = 0.0\nsamples = 1\nvalue = [0,
1]\nclass = Melanoma'),
 Text(590.8235294117646, 229.33124999999998, 'gini = 0.0\nsamples = 9\nvalue =
[0, 9]\nclass = Melanoma'),
 Text(738.5294117647059, 280.29375, 'X[6] <= 0.5\ngini = 0.444\nsamples =
3\nvalue = [2, 1]\nclass = Normal'),
 Text(689.2941176470588, 229.33124999999998, 'gini = 0.0\nsamples = 2\nvalue =
[2, 0]\nclass = Normal'),
 Text(787.7647058823529, 229.33124999999998, 'gini = 0.0\nsamples = 1\nvalue =
[0, 1]\nclass = Melanoma')]
```



[100]:
```
clf_tr.fit(X_train,y_train)
results("Decission tree",clf_tr, X_train, X_val, X_test, y_train, y_val, y_test)
```

```
Decission tree
========================================
Training Score
Accuracy: 0.99
========================================
Validation Score
Accuracy: 0.9166666666666666
Recall: 0.75
========================================
Testing Score
Accuracy: 0.8
Recall: 0.3333333333333333
```

==========