Project 2 Veronika Ebenal, Raghavendra Tikare, Stanislav Klein

Scientific background of the project

This project is centered around classifying data based on bio-medical microscopy images using the Breast Cancer Wisconsin (Diagnostic) Data Set. The diagnosis of breast tumors has traditionally been performed by a full biopsy. However, many different features are thought to be correlated with malignancy, and the process remains highly subjective. We set to increase the speed, correctness, and objectivity of the diagnosis process by using machine learning and deep learning techniques.

Goal of the project

The goal of task 1 was first to analyse and classify the given dataset to gather statistical information about the data, such as summary statistics, outliers, effect sizes, correlation about some available attributes, second to determine the three most important features with regards to diagnostic relevance, and finally to train two classifiers (random forest and support vector machine) to determine the tissue type (benign or malignant). The goal of task 2 was finding a data source with images and training a deep learning classifier that works directly on the images.

Approach 1: Traditional Machine Learning

- **Description of the data**
  The data we used was obtained from the Breast Cancer Wisconsin (Diagnostic) Data Set. This dataset contains extracted biomedical attributes from digitized images of fine needle aspirate tissues for more than 500 patients, as well as a sample diagnosis done by an expert of malignant or benign.

- **Summary of the data statistics**
  The three best features which we performed data statistics on were worst radius, worst area, and worst concave points. All three had roughly normal distributions (of malignant and benign tumors) on their histograms. They all had two outliers. They all had effect sizes of over two, with worst concave points having the highest and worst area having the lowest. Moderate positive correlation existed between all three features.

- **Description of the best features and how they have been determined**
  The three best features were worst radius, worst area, and worst concave points, with the worst meaning highest. This was determined using recursive feature elimination (RFE) with random forest. In RFE, features are recursively weighted (using random forest) and pruned (by smallest absolute weight) until the desired number of features remain.

- **Description and comparison of the classifier methods (RF, SVM)**
  In a random forest classifier, multiple decision trees vote on classification. Each decision tree decides its vote based on a random subset of attributes and samples. The final classification is the one voted on by the maximum number of trees.

  In a support vector machine classifier, each data point is plotted according to its k attributes in k-dimensional space. Next, a line is found to split the data of the two classification as well as possible. The line should be chosen so that the closest points in both of the groups is furthest away.

  In our case, random forest performed better than SVM. In general, whether SVM or random forest is preferable to use is dependant on the data it is used on. For example, SVM works on two-class classifications, while random forest works for more multiclass classifications. Additionally, random forest works on a range of feature types, such as numerical or categorical, whereas SVM must be pre-processed one-hot encoded for any categorical features. Random forest is also preferable for larger data sets, whereas SVM tends to work better on sparser data. Finally, SVM performs best of linear dependencies, and random forest on non-linear.

- **Results**

```
Random Forest Accuracy: 91.23%
SVM Accuracy: 72.81%
```

  Both random forest and SVM gave fairly high accuracies ranging from 80%-100% and 70%-90%, respectively. We believe that the better results from random forest is because of the non-linear dependency between the best attributes and the diagnosis.

Approach 2: Deep Learning

- **Description of the data**
  Our original dataset was obtained from The Broad Bioimage Benchmark Collection (BBBC). The dataset is of annotated biological image sets of testing and validation of C.elegans live/dead assay wherein 100 images are from a 384-well plate of positive and negative controls. However, since we had issues with this dataset, we eventually had to resort to using a "cats and dogs" dataset obtained from www.microsoft.com/en-us/download/details.aspx?id=54765

- **Description of the used method**
  The method which we used was to build a CNN model in Tensorflow 2.0. The first part is importing our dataset into Google Colab and accessing the dataset by referencing the path in the drive. We loaded the testing and the training data and used PIL to load the images from the directory and since the images are of different

dimensions, we used im.resize() to resize each image to a standard dimension of 100x100. We now converted train_images, train_labels and test_images from lists to numpy arrays. We reshaped our training labels and converted them into a one-hot encoding and normalized it to reduce the range of pixel values in each image to 0.0–1.0, to obtain better results from our model. Next we build our model using Tensor flow 2.0. Here we need to assign two tensors for train_ds that are randomly sampled and batched. These tensors represent the training images and labels. After defining our model architecture we created an object for our model and moved on to define our loss functions, optimizer and metrics. Finally we used categorical cross entropy as our loss function for obtaining single correct result and Adam as our optimizer to update network weights iterative based in training data. We trained our model for 10 epochs and saved the weights of our model after each epoch. We also reset our training loss and accuracy values for each epoch. To load the weights of the model, we created an instance of the MyModel class and used load_weights(path) function. We obtained the model predictions by providing the test_images as a parameter and since the model returns a probability distribution we use np.argmax() to get the highest value.

- **Results**

```
- ETA: 33:06 - loss: 0.6798 - acc: 0.5312WARNING:tensorflow:Method (on_train_batch_end)
- 13s 753us/sample - loss: 0.6525 - acc: 0.6029 - val_loss: 0.5993 - val_acc: 0.6672

- 9s 524us/sample - loss: 0.5477 - acc: 0.7200 - val_loss: 0.5033 - val_acc: 0.7492

- 9s 531us/sample - loss: 0.4722 - acc: 0.7721 - val_loss: 0.4793 - val_acc: 0.7699

- 9s 526us/sample - loss: 0.4246 - acc: 0.8029 - val_loss: 0.4026 - val_acc: 0.8164

- 9s 526us/sample - loss: 0.3980 - acc: 0.8187 - val_loss: 0.3940 - val_acc: 0.8177

- 9s 521us/sample - loss: 0.3565 - acc: 0.8373 - val_loss: 0.3856 - val_acc: 0.8240

- 9s 518us/sample - loss: 0.3284 - acc: 0.8530 - val_loss: 0.3482 - val_acc: 0.8426

- 9s 519us/sample - loss: 0.3007 - acc: 0.8696 - val_loss: 0.3502 - val_acc: 0.8447

- 9s 520us/sample - loss: 0.2789 - acc: 0.8768 - val_loss: 0.3759 - val_acc: 0.8373

- 9s 520us/sample - loss: 0.2473 - acc: 0.8922 - val_loss: 0.3507 - val_acc: 0.8534
```

```
13 #Uploaded cat image
14 prediction=model.predict(prepare("/content/cat.jpg"))
15 print(prediction)
```

```
[[0.02217297 0.977827  ]]
```

As can be seen above, our accuracies and losses varied quite widely. Val_loss (the value of cost for cross-validation data) was between roughly 0.34 to 0.6 and loss (the value of cost for training data) was between 0.24 to 0.66. Val_acc (accuracy in cross-validation) was between 0.74 to 0.85. And finally, accuracy (in test data)

was 0.6 to 0.9. We ran a small test at the end with a cat image, which predicted it to be a cat with 97.78% certainty, leaving 2.22% certainty that it may be a dog.

Discussion of the main difference of the two approaches

Traditional machine learning works on explicit data, with parameters which have often been pre-processed in some way and are easily digestible as input by the program. Deep learning, however, does not need data explicitly extracted before classification. It can do classification work directly on input images, using a very complicated neural network system. The best method to choose is dependent on many factors, including input type and size, time and CPU resources, type of output desired, and ability to pre-process data. Although traditional classifiers seem to be a more convenient and quicker way to implement, they are more likely to have a bigger uncertainty due to outliers or missing data. In the end, both deep learning algorithms and machine learning classifiers fulfill their role when implemented in their most fitting environments.

Discussion: why is this a typical project for a data-scientist? (Or why not?)

This could definitely be considered a typical project for a data scientist. All of the characteristics of data science discussed in lecture were used in one way or another in this project.

Statistics and mathematics: The correct and incorrect results of training and testing the classifiers were used to calculate the accuracy of the classifier.

Machine learning: Task 1 is an example of typical machine learning: training and testing a classifier with some input data and classifications. Task 2 goes beyond simple machine learning into deep learning but many underlying principles still apply.

Programming: the project was actualized using programming in Python.

Data processing: one-hot encoding of the data is an example of data processing.

Data visualization: in order to better understand the data, we first visualized it with different plots and graphs.

Feature selection & extraction: in order to choose the best features we tried using univariate feature selection and recursive feature elimination.

Data knowledge: understanding redundancy and correlation between features, understanding the data statistics and visual aids outputted by the program, and being able to verify realisticness of results all required knowledge of the underlying data.