Machine learning Group Exam: Markus, Nikita og Bo

## Task 1 Machine learning on tabular mushrooms

The goal of this task is to determine if a mushroom is edible or poisonous. The first thing we chose to do with the handling of data is to add a section above to give a visual description to the attributes of a given mushroom. We looked over the data and noticed that the column with the name “stalk root” had missing values marked with “?”. This seemed interesting for the handling of data and we wanted to remove this. Since it is unknown we thought we could just remove the unknown attributes, but we reconsidered. We decided to split the data into a binary table to see if a mushroom included a certain attribute or not.
Started by looking at “LabelEncoder” to substitute the data from characters into numbers going from 0 to the highest different attribute, but this seemed unclear to us while researching, so we switched to OneHotEncoder to later split up the data into as an example: From just “cap_shape” into “cap shape_b, cap shape_c, cap shape_f, cap shape_k, cap shape_s, cap shape_x” and assign a 1 or 0 value to them if the mushroom has or hasn't had the certain attribute as mentioned below. As we will use one-hot encoding on each of the features, having an unknown stalk root can be considered a feature that can be used for prediction.

Write about the train_test_split

Knn (Overfit, underfit)

Logistical regression

Neural network

## Task 2 Sentiment analysis

We started working on task 2 since the most recent lab that was out was lab3 which covered sentiment analysis. We started by implementing the BernoulliNB module by using the training data from train.json and creating our own test data by taking 20% from the train.json which gave us accuracy results with the accuracy being: Accuracy: 0.5981191222570533. This was before we realized that there is test.json.

Using the test.json and train.json, we then get a result of Accuracy: 0.7375105842506351. This is a significant increase, just by using the correct data. This happens because it allows the model to analyze more data to train itself to recognize the sentiment of words.

The next, and quite simple step, is to look at the data to see if there are any changes that can be made to help the model. For example, this would include the removal of “periods”, “commas”, “forward-slashes” and even unnecessary spaces. We would deem these characters unnecessary for the model because of the goal we have with it. In this task, we want to find and count similar words within sentences to then set them with a defined sentiment. Removing these characters allows similar words to be counted together. As an example, before the removal, “good” and “good.” are considered two different words. This hinders the accuracy of the model quite less than expected. because this adds redundancy to our data.
Another thing we thought was important is to remove all capitalization so that words, as an example, “Good” and “good” become equal for the model.

After removing some unnecessary characters and removing capitalizations, the model then performed with Accuracy: 0.7383573243014394. Is it then even worth implementing?

I am unsure what the next step would be. Perhaps the idea is to continue looking for unnecessary characters to try and remove them. There must be a better way to improve the model. Perhaps another model may work better?

While testing our current model we found out that feeding the model correct data gave us higher results and removing punctuations and capitalizations adjusted it in a non-significant way.

We found out that by switching the model to multinomial model we got a far greater accuracy to the model. *Accuracy here*

The reason for multinomial model being better is because it takes into considers the frequency of the words compared to Bernoulli that look at if the word is present or absent

***NEW, rewritten task 2. Not sure if we should have our original mistakes in, like using our training data also as the test data, as well as fitting both the test and training data***

Looking over task two, we realized it seemed to be a very similar sentiment analysis task like in the third lab exercise. We took this into account and tried using the same methods as in the third lab. The two methods shown in the lab are using the models Bernoulli Naive Bayes and Multinomial Naive Bayes. This is done by taking each sentence and the positive, negative, or neutral attributes and connecting it using two separate lists, lining up the indexes. This is done to the training and testing data. The next step here is to vectorize the sentences list, meaning it turns the sentences into a numerical representation, so that the model can take the texts and sentiments as inputs. Initially, with Bernoulli Naive Bayes, without any text alteration, we got an accuracy score of 59.36%. This isn’t great. There must be a way to get a higher accuracy score.

The first idea was to preprocess the training and testing data before predicting again with the test data. Removing some characters such as periods, commas, question marks, and unnecessary spaces, as well as making everything lowercase, yielded an accuracy score of, surprisingly, the same 59.36%.

We then thought that perhaps the other model might perform much better. We then tried the Multinomial Naive Bayes model with the same text preprocessing, getting an accuracy score of 61.39%. We were expecting better results.

The next thought was to continue looking at the third lab assignment. There it mentions TF-IDF, meaning Term Frequency - Inverse Document Frequency. TF measures how frequently a word is in the document. IDF measures how important a word is in comparison to the rest of the document. Implementing this only lowered the Multinomial model accuracy score slightly. It’s all a little confusing. It seems that there must be some way to increase the accuracy score, but perhaps there just might not be enough data.

We cannot lemmatize, as it is in norwegian.

### Task 3 Convolutional neural networks

In this task, the assignment is about training a convolutional neural network (CNN) as a binary classifier from the dataset that we have been provided with. This is a CIFAR-10 dataset that consists of 60000 images that are 32x32 colored images and will identify one of the following categories; airplane, automobile, bird, cat, deer, dog, frog, horse, ship, truck. Each of these has 6000 images that are divided into 50000 for the training model and 10000 for the testing model.

This is the last task we started working on since we have not yet covered the material in the curriculum until later. While researching for this assignment we found a couple of different pre-trained CNN models we could use for this project. Some of the choices were ResNet-50/101/152, Inception-v3/v4, MobileNetV1/V2/V3. A decision was made not to use these because of the computing power that would be unsatisfactory for some systems with the amount of layers that the models have. We landed upon the VGG16 model that is a deep CNN model that has 16 layers. It is a less complex model compared to the previous one we have mentioned, but it is more complex then a 3-5 layer CNN. We chose VGG16 at first because we ran into some errors while trying to create our own 3-5 layer CNN using a pre-trained CNN model that has strong performance in particular image classification.

As we implemented the model and chose our own pictures to test on the model we got an initial accuracy score of around 60.89% this wasn't just noticeable when looking at the accuracy. When feeding the model new images for it to classify it miss categorized them in some instances, but was correct in other instances most of the time. Apparently, this might be a coincidence but worth mentioning that it was less accurate when we started mixing the files like using a .png and a .jpg. When we only used .png then it was accurate most of the time. The accuracy of the model is not only affected by the difference in the format of the input images but in the model's architecture, the training data, and the fine-tuning.

To Improve the model's performance we increase the number of training epochs we are using from 10 which gives the initial accuracy to 15. This gave us an accuracy of 67.13 but we have to be aware that adding more epochs makes it so that it trains longer and requires more computing power and we have to be aware not to overfit our model either. So finding the appropriate amount is important. We could also adjust the hyperparameter tuning of the  learning rate or batch size. We also could fine tune more layers of the model to increase its score

When using the model to classify new images that we have taken or decided to use, it is important to preprocess them in the same way as we did with the training data. This would include resizing the images, normalizing their pixel values, and ensuring that they are in the same format.  Doing this leads to the model doing their job great of generalizing its learning to new data and maintaining a consistent performance.

In conclusion, this assignment demonstrates the usage and potential of a deep convolutional neural network, such as VGG16, for image classification tasks. Although our initial accuracy was not very high, adjusting different parameters led to improving the model.