How can Machine learning be used in different ways to predict bipolar disorder?

Joakim I. Frogner

Thesis submitted for the degree of
Master in Programming and Networks
60 credits

Department of Informatics
Faculty of mathematics and natural sciences

UNIVERSITY OF OSLO

Spring 2019

How can Machine learning be used in different ways to predict bipolar disorder?

Joakim I. Frogner

How can Machine learning be used in different ways to predict bipolar disorder?

http://www.duo.uio.no/

Printed: Reprosentralen, University of Oslo

Abstract

Contents

List of Figures

List of Tables

Preface

Part I

Introduction

Chapter 1

Introduction

1.1 Motivation

Statistics
- Data shows that 5,890,000 adults are diagnosed with bipolar disorder in the USA (2,65% of the adult population) [find better source].

Ways to use the results of this study?

1.2 Thesis overview

[Fill in later]

Chapter 2

Background

2.1 Bipolar disorder

Bipolar disorder is the disorder where you experience extreme mood swings. One day you can feel amazing and everything is fine, but the next day you feel like you don't belong anywhere in this universe. Mood swings in general is not something that you should be concerned about. It is however the extreme cases where your mind turns 180 degrees from day to day that is the main symptom of bipolar disorder. There is not really a specific type of people that get this; they can be of any age and any gender, but most people that suffer from it find out (by having an experience or episode) around age 25 [3].

When talking about bipolar disorder, we often separate between the states *normal*, *mania* and *depression*. The last two are the states we usually talk about, since a normal state isn't that interesting. These two states are very different, but they have some similarities, for example sleeping problems.

When a bipolar person is in a manic state, he/she may do things that they never would have intended doing, like spending a lot of money on items they really don't need, or abusing drugs/alcohol. They may also feel really excited or powerful [2].

A bipolar patient is in a depressive state when he or she is in a bad mood swing. They can stop doing everything they usually like to do, and lie down in bed all day with no motivation to do anything useful. They may feel useless and that they don't belong here, or being guilty of something they may or may not have done. In some cases, a depression may even end up with suicidality, where the person either just thinks of death, or actually attempt suicide (actually 20% of people diagnosed with bipolarity commit suicide [3]).

The frequency of these symptoms can vary. One year they can have these mood swings every day for several weeks at the time, and the next they get them less frequent, like once every month.

We also separate between bipolar disorder type I and II, with the main difference being that the manic episodes are way more aggressive in type I [1].

Statistics say that bipolarity is genetically inheritable, with 23% chance of getting a child with bipolar disorder if one parent is bipolar, and 66% if both parents are [3].

2.2 Machine learning

Machine learning is the field of computer science where you basically throw a lot of data into an algorithm and expect it to give you answers to whatever you prefer, with as little work as possible. This was not the case in the early days of the technology, but nowadays it is a lot easier with all the diffent frameworks and tools available.

Machine learning is a great and almost 'magical' technology, but only if you do it right. First you need to have enough data to feed into the algorithm, and to be efficient when training the model on a large dataset, which you need to be if you want your result quickly, you need good hardware. You can get away with a decent CPU if you just want to test it out on a small dataset, but if you really want to do machine learning, then you need a good GPU. The reason why GPUs are so much better than CPUs on this specific task, is because the CPUs are designed for flexibility and general computing workloads. The GPUs on the other hand, are designed to do the same instructions over and over again in parallel. This makes GPUs a lot more efficient for machine learning, and especially for deep neural networks [18].

Now how do you do the actual machine learning? Well there are many diffent approaches to this, which I will discuss in the next sections, but say you want to use a neural network for your task. Then your next step should be to choose a framework. You can of course do everything yourself, but why reinvent the wheel when there are so many good frameworks and tools already out there?

The programming language **Python** is great for machine learning in my (and many other peoples) opinion. It is structured in a way such that it really looks like pseudo-code, and this is perfect because we don't want to spend time on weird syntax rules in another language. For Python, you have a popular framework called **TensorFlow** which is developed by Google. This allows you to build models easily, and also execute the training and testing. Before you get started with TensorFlow, do a quick google search to see if someone else has already done something similiar to what you are trying to acheive, and if you find something, odds are that your neural network model can be similar, if not identical to it. If not, then you have to sit down and actually make the model yourself.

For the model implementation part, whether you found a model online or want to build it yourself, you can of course do it in TensorFlow, but there is an easier way. **Keras** is also a popular framework that is most commonly used together with TensorFlow. On their documentation website [9], they see their framework as 'A high-level neural networks API, written in Python and capable of running on top of TensorFlow, CNTK or Teano'.

Follwing their '30 seconds to Keras' guide [9], you can create a 'sequential' model with 'dense' layers, configure its learning process (compile), then fit, train, evaluate and predict with just a few lines of code:

Source Code 2.1: 30 Seconds to Keras

```
from keras.models import Sequential
from keras.layers import Dense

model = Sequential()

model.add(Dense(units=64, activation='relu', input_dim=100))
model.add(Dense(units=10, activation='softmax'))

model.compile(loss='categorical_crossentropy',
              optimizer='sgd',
              metrics=['accuracy'])

model.fit(x_train, y_train, epochs=5, batch_size=32)
loss_and_metrics = model.evaluate(x_test, y_test, batch_size=128)
classes = model.predict(x_test, batch_size=128)
```

So, as long as you know your theory, and can decide which model to use (and either find a good implementation of that model or create it yourself), you can easily do machine learning. One important task you have to do, is to make the dataset ready. This is the boring and tedious part of machine learning, but it has to be done in order for making it possible to train the model on it.

2.3 Machine learning strategies

Picking the right machine learning model can be quite difficult, especially if you don't have any experience from earlier. There are a couple of diffent *strategies* you can choose from when deciding on a machine learning model. These are called *Supervised and unupervised learning*, and you need to look at your dataset and how it is structured to find out which one to use. The following sections will be a description of the strategies, to make the decision easier.

2.3.1 Supervised learning

This is the machine learning strategy where both input and desired output data are provided [17]. You can use this if you want to train a model to classify letters in the alphabet, or anything else where you have a dataset with both input and output data (for the alphabet, images of letters are input data and the actual letters are the output data). If you train this alphabet model, you will be able to input a competely new image of a letter, and the model will classify it to the letter it most likely fits. This kind

of supervised learning is called *Classification*, and is *the problem of assigning new observations to the class they most likely belong, based on a classification model built from labeled training data* [12].

Another kind of supervised learning is called *Regression*, and is all about predicting (or estimating) a value. A classic example for regression learning is predicting income, using *features* like home location, job title, field of study, years of education and years of experience. We call these features *categorical* (first three) and *numerical* (last two) [11].

2.3.2 Unsupervised learning

Another strategy is *Unsupervised learning*. This is what you want to use if you have a dataset without the same meaning as in a dataset for supervised learning. The items may not have a fixed answer, like the letters in the alphabet are. It is useful when you have *unlabeled* data, and want to for instance group data together in what we call a *cluster*. It may not be as commonly used as supervised learning, but unsupervised learning can also be very useful in some cases; like grouping addresses together in neighborhoods if you have a unsorted list of addresses as a dataset.

2.3.3 Semi-supervised learning

Now, you may not always want to use one of the strategies above. Looking at your dataset you may want something in between; a combination of labeled and unlabeled data. This is when semi-supervised learning comes in handy. For example if you have a lot of data to give labels to in your dataset, it can be simply too much work. We won't go deep into details about how this works, but it is important to mention it because of its usefulness.

2.4 Machine learning approaches

When you know whether you want to use supervised learning, unsupervised learning or something in between, you need to select an approach. We call them approaches because you use these regardless of the strategy you end up using; most of them (with the exception of *reinforcement learning*, which we will come back to) work in both supervised and unsupervised learning. There are a lot of diffent approaches available, and we will describe some of them, namely those we will use in the main parts of this thesis.

2.4.1 Decision tree learning

In computer science, trees are data structures commonly used to describe something that *branches out* based on different input. For example a tree can be a representation of how the frequency of letters in the alphabet are distributed in a text file, so that the text file can be compressed optimally.

Day	Temperature	Outlook	Humidity	Wind	Run
1	15 C	Sun	Low	Strong	Yes
2	6 C	Rain	High	Weak	No
3	15 C	Rain	Medium	Strong	Yes
4	6 C	Overcast	High	Medium	Yes
5	15 C	Sun	Low	Weak	No
6	12 C	Overcast	Medium	Weak	No
7	12 C	Sun	Medium	Medium	Yes

Table 2.1: Training data set: days a person went out for a run

I won't go into details about how this works, but my point is that tree-structures are very common in most fields of computer science.

In machine learning, we can apply the tree-structures as *decision tree learning*. And in this approach, we set up all the different outcomes (with the training data set) of a specific question in a tree. Let's say you want to predict whether or not a person will run outside on a specific day. Then it makes sense that the training set contains weather information. The different data in the training set is called attributes, and picking these correctly is important for the quality of the prediction.

Table 2.1 contains data about whether a person went outside for a run or not for a week (just an example, not real data). Here the first 4 (excluding "Day") columns (Temperature, Outlook, Humidity and Wind) is the "predictors" and the last column (Run) is the "target". To use this table in decision tree learning, we need to view it as a tree, with one of the predictors as root node and the targets as leaf nodes.

How we choose the tree structure is critical to the performance of the machine learning, and we need to use a good tree building algorithm. The most common algorithm to use in this situation, is the *ID3* algorithm made by J. R. Quinlan. It is a top-down, greedy search through the space of possible branches with no backtracking [6]. The way this happens is by calculating *Entropy* and *Information Gain*. The idea is to recursively choose the predictor that has the highest information gain and generate a tree structure. With an optimal tree, you can create decision rules by simply following the tree with new data.

Random Forest

One known problem with decision tree models, is that they often include a lot of *Variance*. This means that an algorithm is sensitive to small changes in the training set. One method to reduce the variance, is to use Random Forest.

Random Forest is a supervised machine learning strategy, which can be used for both classification and regression learning [16]. It essentially works by combining decision trees, where the tree building algorithm is heavily randomized for all trees. For example, if you want to get movie recommendations using machine learning, using one decision tree will

most likely be insufficient. Just think what happens when you ask one friend for movies to watch. What that friend recommends is purely based on movies you like and what he has already watched. You might be lucky and find your next favorite movie, but most likely, asking multiple people for recommendations is going to give a better result. The same goes for machine learning, and decision trees will most likely give a better answer if they are combined in a Random Forest.

2.4.2 Neural networks

General idea

The general idea of machine learning with neural networks, is to make the computer think like a human; it is inspired by they way biological neural networks in the human brain process information [13]. There are a lot of diffent neural networks, but all of them share the same underlying layer based architecture, where data is passed between layers where computation is done. The first layer is the input layer, which simply passes the data to the next layer, which is the hidden layers. The number of hidden layers is competely up to the model and the programmer, and this is where the intermediate processing/computation is done, before the data is passed to the output layer where we perform an activation function to define the output [14].

If you have a lot of hidden layers in a neural network, we call it a *deep* neural network. This is commonly used, and there is a lot of different deep networks with a lot of diffent use cases. Two of the most common neural network models are *Recurrent Neural Networks* and *Convolutional Neural Networks*. These two have their own use cases, which I will describe below.

Figure 2.1 is a visualisation of a basic neural network with the input layer on the left, one hidden layer in the middle and the putput layer on the right hand side. Every node in the input layer is connected with every node in the hidden layer, and every node in the hidden layer is connected to every node in the output layer.

Recurrent Neural Networks (RNN)

This type of neural network is good for predicting something based on a sequence of data, like for example predicting words in a sentence, which can be especially useful for typing on a phone. Also doing predictions based on historical data, like a forecast, is something an RNN can do effectively, which is something the dataset that I'm going to use in this thesis consist of.

One downside to regular Recurrent Neural Networks, is that if the sequence of data is long, the prediction will most likely be off if something that was for example typed in the beginning of a long text is a dependancy for a prediction four chapters later, like the location of the main charater. The workaround for this is something called *Long Short-Term Memory*

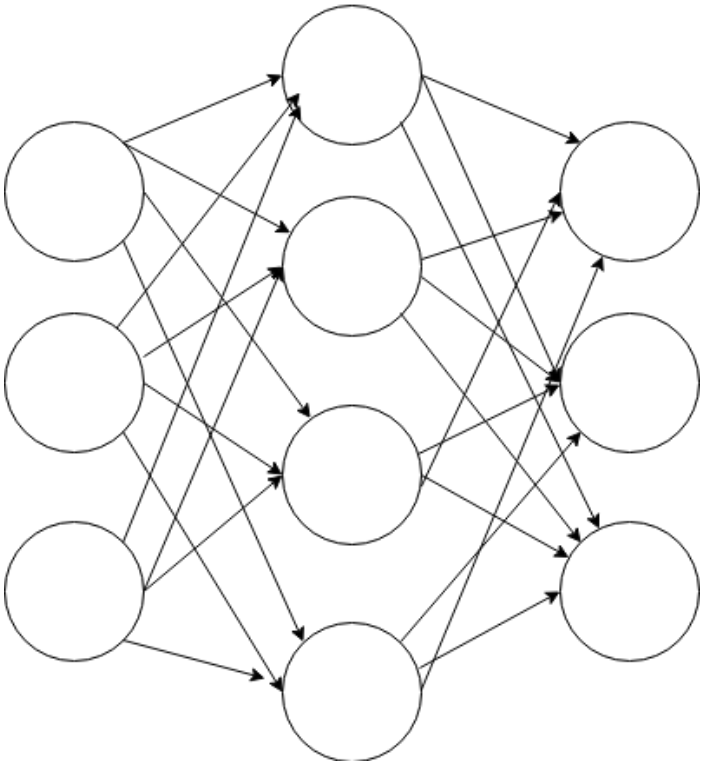

Figure 2.1: Neural network

Recurrent Neural Network (LSTM RNN), and is the idea of having additional logic to avoid the prediction model forgetting important facts.

Convolutional Neural Networks (CNN)

A Convolutional Neural Network, or *CNN* for short, can be used for identifying patterns in data, which can be used for predictions. A common use case for a CNN is image recognition. This is where you train your model to be really good at identifying objects in images, for example the difference between cats and dogs. Then you can input a competely different image to the model, and it will output whether the image is of a cat or a dog. This type of CNN is two-dimentional because an input image is really a two-dimentional array of pixel values, and it is most common to have a 2D CNN. Another way of constructing a CNN, is one-dimentionally, which can be useful for *one-dimentional* data, for example sensor data from gyroscopes or accelerometers [8].

	number	days	gender	age	afftype	melanch	inpatient	edu	marriage	work	madrs1	madrs2
0	condition_1	11	2	35-39	2.0	2.0	2.0	6-10	1.0	2.0	19.0	19.0
1	condition_2	18	2	40-44	1.0	2.0	2.0	6-10	2.0	2.0	24.0	11.0
2	condition_3	13	1	45-49	2.0	2.0	2.0	6-10	2.0	2.0	24.0	25.0
3	condition_4	13	2	25-29	2.0	2.0	2.0	11-15	1.0	1.0	20.0	16.0
4	condition_5	13	2	50-54	2.0	2.0	2.0	11-15	2.0	2.0	26.0	26.0

Figure 2.2: Demographics about participants (5 first rows)

	timestamp	date	activity
0	2003-05-07 12:00:00	2003-05-07	0
1	2003-05-07 12:01:00	2003-05-07	143
2	2003-05-07 12:02:00	2003-05-07	0
3	2003-05-07 12:03:00	2003-05-07	20
4	2003-05-07 12:04:00	2003-05-07	166

Figure 2.3: Activity measurements (5 first rows)

2.5 How can machine learning help people with bipolar disorder?

The usage of machine learning in the medical fields is growing exponentially these days. There are so many use cases of machine learning, and of course it can help in the bipolar field too! Let's say bipolar patients had a device that measured their heart rate among other things 24 hours a day could feed the data into a machine learning model that could give the user live feedback on which bipolar state they are currently in. I think that would be very useful, for both the patients and doctors/nurses. Another use case coudl be if medical institutions could know in advance how many new bipolar patients to expect the next day.

I believe that using machine learning in this field of study could help a lot of people get through their depression or mania, and potentially get rid of the condition competely.

2.6 The dataset

The dataset we will use in this project [7] was collected for another study for motor activity in schizophrenia and major depression. With the data about schizophrenia stripped out, this dataset is sufficient for my thesis. It contains activity level data for 23 bipolar and unipolar patients, and 32 non-depressed contributors. From now on, I will refer to the bipolar/unipolar group as the *condition group*, and the non-depressed group as the *control group*. This is also done in the dataset details [5].

The dataset is in two parts. One part includes the demographics of each participant 2.2. For the control group this only includes the number of days they were collecting data, their gender (1 for female and 2 for male) and age. For the condition group, it also includes their affliction type (1 for bipolar type II, 2 for unipolar depressive, 3 for bipolar type I), melanch (1 for melancholia, 2 for no melancholia), inpatient (1 for inpatient, 2 for outpatient), edu (education in years), marriage (1 for married / cohabiting, 2 for single), work (1 for working / studying, 2 for unemployed/sick/pension), madrs1 (MADRS score when measurement started) and madrs2 (MADRS score when measurement stopped) [5]. MADRS score (Montgomery-Asberg Depression Rating Scale) is used to grade the current severity of an ongoing depression [5].

The second part includes sensor data about the condition group and control group, as one file for each person 2.3. These files are in two folders: "condition" and "control" (for the two groups respectively), and one file for each person is inside the folders (filename is "GROUP_X.csv" where X is their id and GROUP is either condition or control. Inside the files, there is a list of activity measurements for every minute of the data collection period.

2.7 What to do in this project

As stated in the problem statement, we want to perform different types of machine learning on the dataset. There is a couple of things we have in mind that the dataset can be used for:

- Predict whether a given participant belongs to the *control group* or the *condition group*.
- Predictions on the patients MADRS score and sleep patterns.

More details on how we will acheive our goal will come in the next part of this thesis.

2.8 Challenges and ethical concerns

In most projects in the medical fields, there are going to be ethical concerns and challenges with privacy. What happens if someone that are not clearified for the data gets access to it? What if the database gets hacked? With new regulations (GDPR), which basically means that users have the right to be "forgotten". However, in this project all data is anonymized (only referenced by an id), so there will be no persons mentioned. If the dataset were not to be anonymized, and the patient's names were in it, things could get problematic if it got into the wrong hands.

Part II

The project

Chapter 3

Planning and preparing data

3.1 Goals

Our goal is to build a machine learning model that can find out which group a participant belongs to (*control group* or the *condition group*). There are several ways this can be achieved, and two ideas came to our minds. The first one was a bit naive, and was to simply throw in the columns from the demographics dataset 2.2. We did not expect much from this, as there is only 55 rows in the table. Anyone having a little bit experience with machine learning will know that this is not nearly enough data. But we wanted to do it regardless, and see how a simple and stupid model performed. Doing this, we would also establish some sort of benchmark for performance; the second model must do *at least* better than this one.

The other idea was to use the activity measurements as time series data, and create a one-dimentional Convolutional Neural Network (1D-CNN). We knew that CNNs are used a lot today on image recognition, but is a bit different as images are two-dimentional data, and our measurements are in *one* dimention.

3.1.1 Regression

We decided to use a regression classifier for the first idea. However the dataset is not structured in a way such that we can just throw it into Keras. First and foremost, most values in this table are blank for participants in the *control group*, so it does only make sense to predict within the *condition group*.

The columns **number** and **days** should be dropped, as they probably have nothing to do with the result. The column **afftype** is be the column the regression model should guess, so it needed to be in a separate table (X = input table, y = output table).

Both the *age* and *education* columns are strings with values that the actual value is within. For example the participant **condition_1** has age = **35-39** and edu = **6-10**. It would be better to change this to one value, and we decided to use the median of the two values. Also, we changed some other values to make more sense. *Melanch, inpatient, marriage, work and gender* are **binary** values, so we changed them to be either 0 or 1. We also changed *afftype* (terniary value) to between 0 and 2 instead of between 1 and 3.

After the changes, the tables should be read the following way:
X:

- **gender**: 0 = male, 1 = female
- **age**: Median value of age range
- **melanch**: 0 = does not have melancholia, 1 = has melancholia
- **inpatient**: 0 = outpatient, 1 = inpatient
- **edu**: Median value of education range
- **marriage**: 0 = single, 1 = married (or cohabiting)
- **work**: 0 = not working, 1 = working
- **madrs1**: MADRS score before activity measurements
- **madrs2**: MADRS score after activity measurements

y:

- **afftype**: 0 = bipolar II, 1 = unipolar depressive, 2 = bipolar I

3.1.2 Convolutional Neural Network

As we said before, we wanted to use a Convolutional Neural Network to classify which group a given participant most likely belongs to.

Learning experiments

We needed to learn more about CNNs. CNNs are used in image recognition, so we proceeded to implement one. We found a tutorial on how to make a 2D CNN for classifying cats and dogs from images [10], and thought it would be a good way to learn.

It was both a fun and informative experience implementing this. Especially when we extended the script to allow an image url to predict on. Then we could browse for any image of a cat or a dog, and find out if the model could handle it (in most cases it did!). We even tried inputting images of humans to the model for fun. This experiment resulted in a lot of motivation for our task.

However as mentioned before, our data is one-dimentional, so a two-dimentional CNN would not be useful.

> *A 1D CNN is very effective when you expect to derive interesting features from shorter (fixed-length) segments of the overall data set and where the location of the feature within the segment is not of high relevance. This applies well to the analysis of time sequences of sensor data (such as gyroscope or accelerometer data).* [8]

To learn more about 1D CNNs, we followed a tutorial [8], which used a dataset containing time-sliced accelerometer data from a smartphone on the participants waists. The goal for this CNN is to predict what a given person is doing at the time, given the accelerometer data for that time slice. What the given person is doing is one of the following:

- Standing
- Walking
- Jogging
- Sitting
- Upstairs
- Downstairs

As we followed the tutorial and implemented the model, we learned a lot about how 1D CNNs work and how we should structure our own data. In machine learning it is common to normalize data values, which is important to for example make up for differences in features. There are built in ways to automatically do this, but you can also make your own *normalizer*, which was done in this tutorial in the following way, and is the one we ended up using in our activity measurement classifier as well:

```
1     def feature_normalize(dataset):
2         mu = np.mean(dataset, axis=0)
3         sigma = np.std(dataset, axis=0)
4         return (dataset - mu)/sigma
```

Source Code 3.1: Feature Normalizer

We also learned where our dataset could provide more data. What if the dataset contained the current mental state of the bipolar patient? Then someone could make some automated system that always can tell a patient whether they are normal, manic or depressive. However data collection for this kind of task would be difficult because we can't always know what the patient thinks, nor does the patient themselves. The "tutorial" dataset is different because it is easy to differientiate physical states of the body like standing or walking.

Classification based on activity measurements

The input data was quite different for this kind task. We wanted the input data to the model to be a list containing activity measurements for all participants, and somehow label them **[NOT BIPOLAR]** or **[BIPOLAR]**. We ended up using time-slices of measurements, as previously learned in the tutorial [8].

We created a list where for each participant in the demographics table 2.2, measurements for four hours were grouped. Another choice we

	Actual: Negative	**Actual: Positive**
Predicted: Negative	True Negative (TN)	False Negative (FN)
Predicted: Positive	False Positive (FP)	True Positive (TP)

Table 3.1: Confusion Matrix

	Actual: Not Bipolar	**Actual: Bipolar**
Predicted: Not Bipolar	**95**	5
Predicted: Bipolar	7	**93**

Table 3.2: Confusion Matrix Example: Bipolar vs Not Bipolar

learned from the tutorial was to overlap the sequences, so we made the next group of four hours start *one* hour after, and not *four* hours after the group before, as one might think. When this list was complete with sequences from all participants, we had to **reshape** it so that it could fit into a neural network. We ended up with a feature list (which we called **segments**), where each element was a list of activity measurements for 4 hours:

segments[0] = [[0], [143], [0], [20], [166], [160], [306], [277], [439], ...]

A second list was created simultaneously, with the value **0** or **1** for the labels **[NOT BIPOLAR]** and **[BIPOLAR]**. This value was chosen according to the group the participants were in. Using a helper function from Keras, **to_categorical**, we transformed this list of labels into to columns, **[NOT BIPOLAR]** and **[BIPOLAR]**, where only one of the columns have the value 1 for each row. Transforming the values to a categorical table is required for the neural network that we ended up building, to be able to select a *category* for the result. This list, which we called **labels**, looked like this:

labels[0] = [0, 1]

Meaning that the first segment is labeled as **[BIPOLAR]**.

3.2 Performance Metrics

3.2.1 Confusion Matrix

Table 3.1 shows a *confusion matrix*. It is a very common and easy to understand metric for classification models in machine learning, and is the basis for other of common performance metrics, which we will describe later in this section. It can tell you how well your model is performing, by having correlation values for the different predicted classes. In our planned model for predicting bipolar vs not bipolar, let's say we have 200 samples in our test data, a *confusion matrix* for a good model would look like table 3.2, with highest numbers in **True Positive** and **True Negative** and as low numbers as possible in **False Positive** and **False Negative**. Having a high number in **True Positive** means that the model is able to predict that a participant is bipolar if he or she actually is bipolar, and having a high number in **True Negative** means that the model is able to predict that a

participant is not bipolar if he or she actually isn't. The other cases, **False Positive** and **False Negative**, is where the model made a wrong prediction, and therefore as close these numbers are to zero the better our model is.

3.2.2 Accuracy

'Accuracy is a good measure when the target variable classes in the data are nearly balanced.' ([15])

When calculating the *accuracy*, we sum up the correct predictions and divide that with the total number of predictions ($\frac{TP+TN}{TP+TN+FN+FP}$). For our example (3.2), the *accuracy* would be $\frac{93+95}{93+95+5+7} = 94\%$. It is a good metric to use for our example because the number of samples for each variable class is well balanced ($93 + 5 = 98$ samples where **bipolar** was the correct option, and $7 + 95 = 102$ samples where **not bipolar** was correct).

Terrible use of the *accuracy* metric would be when one of the classes strongly dominates the samples. For example, if a model predicts **cancer** vs **no cancer**, and the samples contain 5 people with cancer and the 95 remaining people do not. The model would be terrible at predicting cancer and have an *accuracy* score of 95% [15].

3.2.3 Precision

This performance metric operates entirely on the predicted positives, and it tells us how many **true positives** there is among **predicted positives** ($\frac{TP}{TP+FP}$) [15].

Precision is better to use on *unbalanced* classes than accuracy. The *cancer vs no cancer* example, assuming it predicts no-one to have **cancer**, would yield a precision score of $\frac{5}{5+95} = 5\%$. And our *bipolar vs not bipolar* example would result in a precision score of $\frac{93}{93+7} = 93\%$.

3.2.4 Recall

Recall is another useful performance metric. It tells us the relationship between **true positives** and **actual positives**, for example how many predicted bipolar participants there were among total bipolar participants.

The calculation of recall is done by dividing **true positives** by **true positives + false negatives** ($\frac{TP}{TP+FN}$), which translates into $\frac{93}{93+5} \approx 95\%$ for table 3.2.

Choosing a metric to use from *precision* or *recall* depends on your goal. Try to achieve close to 100% *recall* if you want to reduce **false negatives**, and likewise with *precision* if you want to reduce **false positives** [15].

3.2.5 Specificity

As *recall* operates on **actual positives**, *specificity* is the exact opposite metric. It tells us the relationship between **true negatives** and **actual negatives**. So if your goal is to reduce **false positives**, specificity is a valid choice.

Specificity is calculated by dividing **true negatives** by **true negatives + false positives** ($\frac{TN}{TN+FP}$). For table 3.2, the *specificity* score equals $\frac{95}{95+7} \approx 93\%$.

3.2.6 F1 Score

The metrics that we have described in this section are all useful when determining whether your classification model is good enough. But the relationship between *recall* and *precision* and knowing when to use which can be confusing, at least if the different classes are somewhere between completely unbalanced and perfectly balanced (for example a 35% split).

Therefore another metric called *F1 Score* was created, which gives us a balanced value combining *recall (R)* and *precision (P)*. The basic idea is to return the *mean* value of the two scores (F1 = $\frac{P+R}{2}$), but that would not be balanced if one score is much lower than the other. F1 score actually uses something called *harmonic mean* instead of the standard *arithmetic mean*, and is calculated as $2 \cdot \frac{P \cdot R}{P+R}$ [15].

Following this formula, the F1 score for confusion matrix 3.2 becomes:

$$F1 = 2 \cdot \frac{P \cdot R}{P + R} = 2 \cdot \frac{0,93 \cdot 0,95}{0,93 + 0,95} \approx 94\%$$

3.2.7 Classification Report

A machine learning framework for Python, *sklearn*, includes a package with functions to calculate most of these scores. Simply import and use them like this:

```
1   from sklearn.metrics import confusion_matrix,
2                               precision_score,
3                               recall_score,
4                               f1_score,
5                               classification_report
6
7   # ... set up model, make predictions ...
8
9   print(confusion_matrix(y, y_class_predicted))
10  print(precision_score(y, y_class_predicted))
11  print(recall_score(y, y_class_predicted))
12  print(f1_score(y, y_class_predicted))
13  print(classification_report(y, y_class_predicted))
```

Source Code 3.2: sklearn metrics

The classification report returns a matrix (see table 3.3) with *precision*, *recall*, *F1 score* and another column called *support* (simply how many samples of data there are for this class). The rows describe each class used for prediction [4].

	precision	recall	f1-score	support
class 0	0.05	1.00	0.67	1
class 1	0.00	0.00	0.00	1
class 2	1.00	0.67	0.80	3
micro avg	0.60	0.60	0.60	5
macro avg	0.50	0.56	0.49	5
weighted avg	0.60	0.60	0.60	5

Table 3.3: Classification Report

3.3 Benchmarks

Chapter 4

Implementing the project

4.1 Regression

4.2 1D Convolutional Neural Network

4.2.1 Creating the Model

Following the tutorial on 1D CNNs [8], we came up with this model 4.2. We used most of the same parameters, but we also had to tweak some of them to make the model work on our dataset. To make a convolutional neural network, you always need some *convolutional* and *pooling* layers. Which layers you add in between and the ordering of them, together with the parameters you pass to the layers, is what makes the model perform differently. This, of course, requires your input data to be well structured.

- We start by defining a **Sequential** model. This is an easy to understand and readable way of defining a model. Alternatively we could use a **functional** model, which gives more control of inputs and outputs of the layers. A **functional** model would be useful if we wanted to debug and optimize each layer within the model.

- **Reshape**: In the first layer we need to reshape the input data so that it becomes an X by 1 matrix, where X is the length of each segment. The reason for this reshape step is because the next layer (**Conv1D**) requires the input to contain (*batch, steps, channels*). *Batch* will be set to **None**, *steps* will be the segments, and *channels* will be 1 (because we only have one measurement value for each minute).

- **Conv1D**: This is the first *convolutional* layer, where the required parameters are how many *filters* you want, and how big the *kernel* should be. As in the tutorial, we also use 100 filters and a kernel size of 10. We don't know if these are the perfect values to use, but having 100 filters means that the network can learn 100 features in this layer. Having less or more might impact the performance, which we will experiment with in the section about model optimization. There are many different parameters that you can use on a layer like this, for example *padding* and *strides*, but using the default values is good for now. This is also something that we can experiment with later. However using the default parameters, the output of this layer will be a $X - 9 \times 100$ matrix, where X is the same as in the first layer.

- **Conv1D**: The second convolutional layer will look exactly like the first one, and the output will be a $X - 9 - 9 \times 100$ matrix.

- **MaxPooling1D**: Pooling is important in a convolutional neural network to reduce complexity [8]. The basic idea of *max pooling* is to reduce to only the maximum value for each *window* of size $N \times N$. We are using 2 as window size (N), resulting in matrix that is half the size of the input ($\frac{X-9-9}{2} \times 100$). Pooling may also help reduce *overfitting*.

Not bipolar	Bipolar
0	1
1	0
0	1
1	0
1	0
0	1
0	1
1	0

Table 4.1: Categorical Labels

- **Conv1D**: Two more convolutional layers are added, and after these the input to the next layer will be a matrix of size $(\frac{X-9-9}{2} - 9 - 9) \times 160$.
- **GlobalAveragePooling1D**: This is another pooling layer, and this one takes the average of weights within the network instead of the maximum. After doing this, the matrix will be of size 1×160.
- **Dense**: The final layer in the model is a dense layer (fully connected) which reduces the matrix from 160 values to 2 with the activation function **softmax**. Then output (a 1 in either of the two neurons), is mapped to the corresponding label (bipolar/not).

4.2.2 Creating Input Data

One function (*create_segments_and_labels()* 4.1) is responsible of creating the data that is sent into the neural network. It does the following:

- First we read the *global* dataset, where we find each participant and whether they are bipolar or not. As there is no *afftype* value for non-bipolars participants, we simply set this to 0. This is fine because the other possible values are 1, 2 and 3.
- Then we iterate over all participant activity data files:
  - Append a **segment** that is $8 * 60 = 480$ minutes long to the list of segments.
  - Append a 1 if the participant is bipolar, or a 0 if not to the list of labels.
  - Skip to the next hour, and repeat until we have added all segments.
- Make the list of labels into a *categorical* 2D matrix 4.1 with a **1** in only one of the columns, instead of a single-dimentional list. This is needed for the **softmax** activation function, which we will describe later.

- Also we need the list of segments to be restructured. We do this with the **reshape** function, and after this the data is ready to be passed into the neural network.

However, one last step is done before we start training the model. We need to split the data into training and test data, so that we can test how good the model is after training it. The **train_test_split** from the *sklearn* package does this, and you input the segments and labels, and also specify how large you want the training and test sets to be. The function also randomizes the data, so that the model won't accidentally learn something that is correct for two segments in a row that also happen to be chronological.

4.3 Optimizing the model

4.4 Source Code

4.4.1 Create Input Data

Source Code 4.1: Reading Dataset

```
def create_segments_and_labels():
  scores = pd.read_csv('scores.csv')
  scores['afftype'].fillna(0, inplace=True)

  segments = []
  labels = []

  for person in scores['number']:
    p = scores[scores['number'] == person]
    df_activity = pd.read_csv(f'{person}.csv')

    for i in range(0, len(df_activity) - 8*60, 60):
      segment = df_activity['activity'].values[i : i + 8*60]
      segments.append([segment])

      if p['afftype'].values[0] == 0:
        labels.append(0)
      else:
        labels.append(1)

  segments = np.asarray(segments).reshape(-1, 8*60, 1)
  segments = segments.reshape(segments.shape[0], 8*60)

  labels = to_categorical(np.asarray(labels), 2)

  return segments, labels
```

4.4.2 The Model

Source Code 4.2: 1D Convolutional Neural Network Model

```
  def create_model(segment_length, input_shape):
    model = Sequential()

    model.add(Reshape((8*60, 1), input_shape=(8*60,)))
    model.add(Conv1D(100, 10, activation='relu', input_shape=(8*60, 1)))
    model.add(Conv1D(100, 10, activation='relu'))
    model.add(MaxPooling1D(2))
    model.add(Conv1D(160, 10, activation='relu'))
    model.add(Conv1D(160, 10, activation='relu'))
    model.add(GlobalAveragePooling1D())
    model.add(Dropout(0.5))
    model.add(Dense(2, activation='softmax'))
```

```
13
14          return model
```

Part III

Conclusion

Chapter 5

Results

Bibliography

[1] *Bipolar 1 Disorder and Bipolar 2 Disorder: What Are the Differences?* Accessed: 14.03.2018. URL: https://www.healthline.com/health/bipolar-disorder/bipolar-1-vs-bipolar-2.

[2] *Bipolar disorder*. Accessed: 14.03.2018. URL: https://familydoctor.org/condition/bipolar-disorder/.

[3] *Bipolar disorder statistics*. Accessed: 14.03.2018. URL: https://www.statisticbrain.com/bipolar-disorder-statistics/.

[4] *Classification Report*. Accessed: 07.02.2019. URL: https://scikit-learn.org/stable/modules/generated/sklearn.metrics.classification_report.html.

[5] *Dataset details*. Accessed: 08.05.2018. URL: http://datasets.simula.no/depresjon/.

[6] *Decision Tree*. Accessed: 08.05.2018. URL: http://www.saedsayad.com/decision_tree.htm.

[7] Enrique Garcia-Ceja et al. 'Depresjon: A Motor Activity Database of Depression Episodes in Unipolar and Bipolar Patients'. In: *Proceedings of the 9th ACM on Multimedia Systems Conference*. MMSys'18. Amsterdam, The Netherlands: ACM, 2018. DOI: 10.1145/3204949.3208125. URL: http://doi.acm.org/10.1145/3204949.3208125.

[8] *Introduction to 1D Convolutional Neural Networks*. Accessed: 07.12.2018. URL: https://blog.goodaudience.com/introduction-to-1d-convolutional-neural-networks-in-keras-for-time-sequences-3a7ff801a2cf.

[9] *Keras documentation*. Accessed: 15.03.2018. URL: https://keras.io.

[10] *Loading in your own data - Deep Learning basics with Python, TensorFlow and Keras p.2*. Accessed: 07.12.2018. URL: https://pythonprogramming.net/loading-custom-data-deep-learning-python-tensorflow-keras/?completed=/introduction-deep-learning-python-tensorflow-keras/.

[11] *Machine Learning for Humans, Part 2.1: Supervised Learning*. Accessed: 11.04.2018. URL: https://medium.com/machine-learning-for-humans/supervised-learning-740383a2feab.

[12] *Machine Learning for Humans, Part 2.2: Supervised Learning II*. Accessed: 11.04.2018. URL: https://medium.com/machine-learning-for-humans/supervised-learning-2-5c1c23f3560d.

[13] *Neural Networks*. Accessed: 21.05.2018. URL: https://ujjwalkarn.me/2016/08/09/quick-intro-neural-networks/.

[14] *Neural Networks Introduction*. Accessed: 21.05.2018. URL: https://towardsdatascience.com/a-gentle-introduction-to-neural-networks-series-part-1-2b90b87795bc.

[15] *Performance Metrics for Classification problems in Machine Learning*. Accessed: 07.02.2019. URL: https://medium.com/greyatom/performance-metrics-for-classification-problems-in-machine-learning-part-i-b085d432082b.

[16] *Random forest*. Accessed: 21.05.2018. URL: https://medium.com/Synced/how-random-forest-algorithm-works-in-machine-learning-3c0fe15b6674.

[17] *What is supervised learning?* Accessed: 11.04.2018. URL: https://searchenterpriseai.techtarget.com/definition/supervised-learning.

[18] *Why are GPUs well-suited to deep learning?* Accessed: 15.03.2018. URL: https://www.quora.com/Why-are-GPUs-well-suited-to-deep-learning.