Machine Learning Projects Guide

By Jens Van Gelder – Stage IntoData

A. Introduction

This document will guide you through the working ML projects made during my internship at IntoData. These projects were made in a time span of 3 months by a regular data scientist with no previous knowledge of machine learning or artificial intelligence. These projects are meant as an example of what a regular data scientist/engineer would be capable of doing in a very short time.

The code for all these projects can be found in the following Git repository: https://github.com/JensVanGelder/ML_Projects

B. Software installation & debugging

The code is written in Python. If you do not yet have a Python environment I would recommend installing the Spyder IDE through Anaconda, a good Python distributor. If you install this software you will mostly have access to everything you need to execute these projects.

These projects, and mostly every ML project made in python makes use of pre-written libraries and packages which we always import at the start of a project.

e.g. :

```python
# Importing the libraries
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
```

⚠ If you ever get an error message saying these modules are not available after initiating this code it means you will have to download them first. This is very easily done by opening up your Anaconda prompt and typing:

```
conda install NameOfMissingModule
```

Just go through this process if you ever get this error message and Anaconda will do the rest.

NOTE: The 8_DQN_self_driving_car project makes use of PyTorch which officially does not yet have support for Windows. If you are a Windows user you can follow the instructions in the following link to get PyTorch working in your environment: https://www.superdatascience.com/pytorch/

C. Project guide

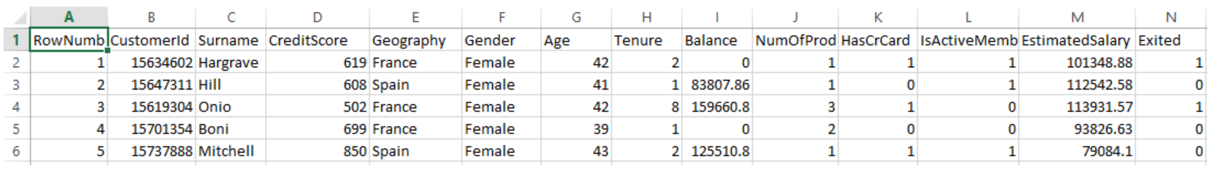
1_ANN_churn_modeling

1. Business Problem Description

A certain bank wants to know when a customer might leave them. Based on a large dataset gathered over many months they want a way to predict if: 1, the customer has left the bank or 0, the customer is still with the bank.

2. Dataset + Goal

The project folder contains the file "Churn_Modelling.csv". This is our dataset provided by the bank.



	A	B	C	D	E	F	G	H	I	J	K	L	M	N
1	RowNumb	CustomerId	Surname	CreditScore	Geography	Gender	Age	Tenure	Balance	NumOfProd	HasCrCard	IsActiveMemb	EstimatedSalary	Exited
2	1	15634602	Hargrave	619	France	Female	42	2	0	1	1	1	101348.88	1
3	2	15647311	Hill	608	Spain	Female	41	1	83807.86	1	0	1	112542.58	0
4	3	15619304	Onio	502	France	Female	42	8	159660.8	3	1	0	113931.57	1
5	4	15701354	Boni	699	France	Female	39	1	0	2	0	0	93826.63	0
6	5	15737888	Mitchell	850	Spain	Female	43	2	125510.8	1	1	1	79084.1	0

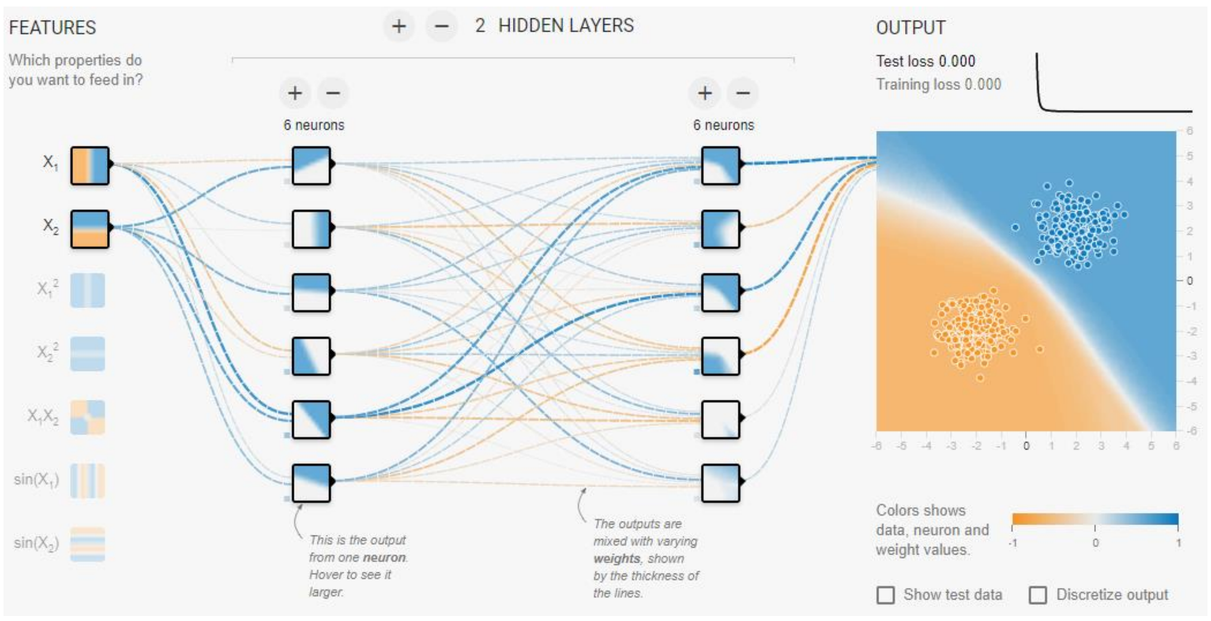
The dataset features a list of customers as rows with their respective info as columns. The last column 'Exited' shows us if a customer has left the bank = 1 or is still with the bank = 0. We will use this data to learn our ML model to predict when a customer might leave the bank based on their other information.

So in the end our goal is to predict either a 1 or a 0 for a specific CustomerId.

3. Model

For this specific case we decided to use a Classification Artificial Neural Network with 2 Hidden layers using the ReLu activation function and the Output layer using the Sigmoid function.

Here is a rough representation of what this neural network looks like:



FEATURES
Which properties do you want to feed in?
X_1
X_2
X_1^2
X_2^2
X_1X_2
$\sin(X_1)$
$\sin(X_2)$
+ − 2 HIDDEN LAYERS
+ − 6 neurons
+ − 6 neurons
This is the output from one neuron. Hover to see it larger.
The outputs are mixed with varying weights, shown by the thickness of the lines.
OUTPUT
Test loss 0.000
Training loss 0.000
Colors shows data, neuron and weight values.
-1 0 1
Show test data
Discretize output

The code used for making this ANN is very simple:

```
# Initialising the ANN
classifier = Sequential()

# Adding the input layer and the first hidden layer with dropout
classifier.add(Dense(output_dim = 6, kernel_initializer = 'uniform',
activation = 'relu', input_dim = 11))
classifier.add(Dropout(p = 0.1))

# Adding the second hidden layer
classifier.add(Dense(output_dim = 6, kernel_initializer = 'uniform',
activation = 'relu'))
classifier.add(Dropout(p = 0.1))

# Adding the output layer
classifier.add(Dense(output_dim = 1, kernel_initializer = 'uniform',
activation = 'sigmoid'))

# Compiling the ANN
classifier.compile(optimizer='SGD', loss='binary_crossentropy',
metrics=['accuracy'])

# Fitting the ANN to the Training set
classifier.fit(X_train, y_train, batch_size=25, epochs=250,
validation_split=0.1)
```

How this ANN works is that it takes the features of a customer as the input and pushes these through a lot of different possible paths, these being the 2 Hidden layers. Then based on previous customer experiences it will attempt to predict a good output for the selected customer, for this project the output will be either a 1 or a 0.

4. How to execute project

If you want to run this project yourself you can simply do this by running the whole code in your Python IDE.

(Always make sure your working directory is in the correct folder or you wont be able to acces the dataset!)

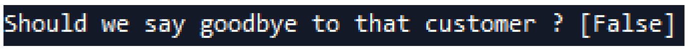
The last 3 lines of code can be adjusted to predict the outcome for your own customer. Simply change the fields in the first line to the ones for your customer. The code shows in comments what every field represents.

```
new_prediction = classifier.predict(sc.transform(np.array([[0.0, 0, 600, 1,
40, 3, 60000, 2, 1, 1, 50000]])))
new_prediction = (new_prediction > 0.5)

print("Should we say goodbye to that customer ?", *new_prediction)
```

After executing the code the ANN will run over many epochs. These can be reduced in the code to save some time.

5. Results

Once the code is finished the console will display if the customer is about to leave the bank.



```
Should we say goodbye to that customer ? [False]
```

2_CNN_cat_or_dog

1. Business Problem Description

We want to make an application that is able to predict if a photo contains either a cat or a dog.

2. Dataset + Goal

The dataset used to train this model can be found via this link:
http://www.superdatascience.com/wp-content/uploads/2017/04/Convolutional_Neural_Networks.zip

This zip file contains the dataset folder. The dataset is made up of 10.000 images of cats and dogs split into a training and test set. Downloading this folder is only required if you want to train your own network. You can use the pre-trained model included in the 2_CNN_cat_or_dog folder to make your own predictions based on the images in the single_prediction folder. You can add your own pictures to this folder if you want to.

3. Model

The model used here is a Convolutional Neural Network also known as a CNN or ConvNet. CNN's are primarily used for image and video recognition and natural language processing that's why they are perfect for this case.

For this CNN we used 2 Convolutional layers with the ReLu activation function, 1 Hidden layer also with the ReLu activation and the Output layer using the sigmoid function.

For a visual representation of what CNN's look like you can go to this link:
http://scs.ryerson.ca/~aharley/vis/conv/

After drawing your number the site will show you how a CNN is able to predict what is written on the image. The principle is the same for our project but instead of the numbers 0-9 we use cats and dogs.

4. How to execute project

First of all always make sure you are in the correct working directory for this project. Next you can run the code step by step as stated in the comments.

Part 1 is used to create the CNN. This is essential.

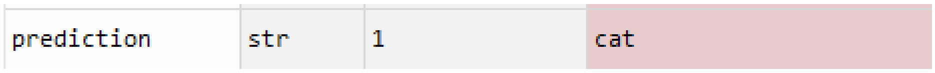
Part 2 is used to train the CNN on the training data. You can skip this part if you dont want to train your own model and you just want to use the pre-trained model.

Part 3 is where we start making the predictions. The first few lines are to load the pre-trained model the others to actually start predicting an outcome. If you want to run a prediction on one of your own pictures you just need to change the path in the following line:

```
test_image = image.load_img('dataset/single_prediction/cat_or_dog_1.jpg',
target_size = (64, 64))
```

5. Results

Once the code is done it will create a new variable containing the prediction:



prediction	str	1	cat

3_Car_class_prediction

1. Business Problem Description

A car dealership wants to be able to predict a car evaluation based on a set of given scores.

2. Dataset + Goal

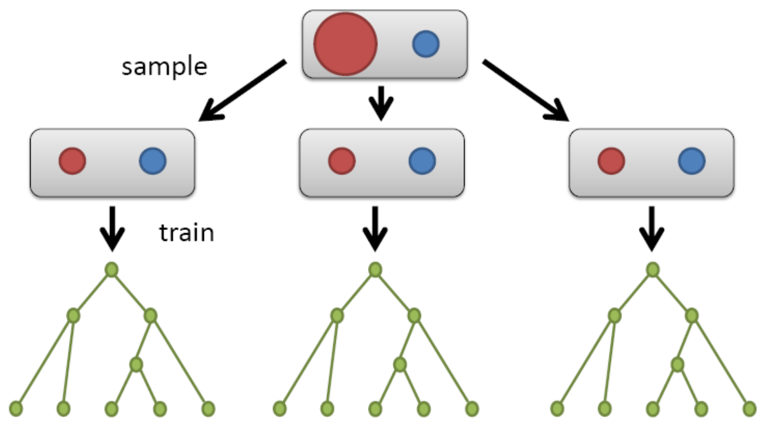
The dataset used is the Car Evaluation Data Set downloaded from the UCI repository with the following link: https://archive.ics.uci.edu/ml/datasets/Car+Evaluation

The dataset consists of scores given to cars during previous evaluations based on many features. The goal is to predict how good a car is from a score of 1 to 4.

3. Model

The model used is a Random Forest Classifier. Random Forests are an ensemble learning method that operate by constructing multiple decision trees during the training and will take combine these trees to create an output. Random forest can be used for both regression and classification. For our business case we need to predict a value of 1 to 4 so we will be using classification.

A quick representation of how Random Forests work:



Setting up a random forest model is very easy. In this project we make one by only using 4 lines of code.

```
# Random Forest algorithm
from sklearn.ensemble import RandomForestClassifier
classifier = RandomForestClassifier (n_estimators = 500)
classifier.fit(X_train,y_train)
classifier.score(X_test,y_test)
```

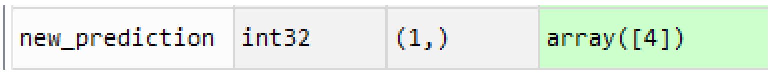
4. How to execute project

To run the project simply execute the whole code. If want to predict the value of your own car you can change the values in this line to fit your vehicle:

```
new_prediction = classifier.predict(np.array([[1,1,6,5,3,3]]))
```

5. Results

This model works very fast. Once it’s done it will create a new variable with the predicted value:



new_prediction	int32	(1,)	array([4])

4_Titanic_kaggle

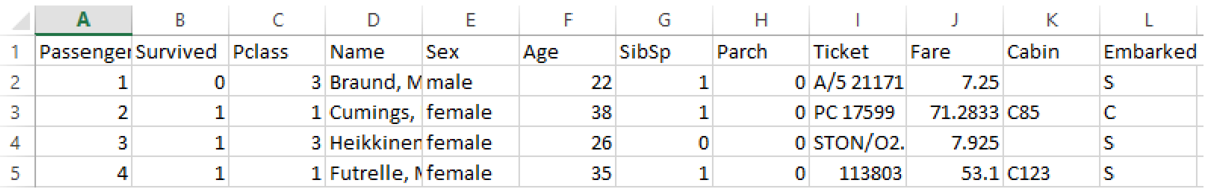
1. Business Problem Description

This project is from an online ML competition hosted by Kaggle. The challenge is to complete the analysis of what sorts of people were likely to survive the Titanic. In particular, they ask you to apply the tools of machine learning to predict which passengers survived the tragedy.

The competition can be found here: https://www.kaggle.com/c/titanic

2. Dataset + Goal

The dataset consists of both a train and test set. As the names might suggest, we use the train set to train our neural network and we will use the test set to make our predictions on. The data contains a list of passengers aboard the RMS Titanic including some specific information per passenger.



	A	B	C	D	E	F	G	H	I	J	K	L
1	Passenger	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked
2	1	0	3	Braund, M	male	22	1	0	A/5 21171	7.25		S
3	2	1	1	Cumings,	female	38	1	0	PC 17599	71.2833	C85	C
4	3	1	3	Heikkinen	female	26	0	0	STON/O2.	7.925		S
5	4	1	1	Futrelle, M	female	35	1	0	113803	53.1	C123	S

Our goal is to predict if the passengers from the ‘test.csv’ file survived or not with as high accuracy as possible.

3. Model

Once again we make use of a Random Forest Classifier. Only this time we make more use out of the adjustable parameters to fine tune our neural network. We also run our model through a pipeline which will make it run a bit smoother.

```
select = SelectKBest(k = 'all')
clf = RandomForestClassifier(n_estimators=10000,
                             max_leaf_nodes=12,
                             max_depth=12,
                             random_state=10,
                             max_features='sqrt')
pipeline = make_pipeline(select, clf)


pipeline.fit(Xtrain, Ytrain)
predictions = pipeline.predict(Xtrain)
predict_proba = pipeline.predict_proba(Xtrain)[:,1]
```

The hard part in this project was not the Model but the data engineering required to make our data fit the model in a way that benefits our predictions. In other words we had to look for what data actually influenced the survival rate the most and split these up into different fields for further use while also dropping unnecessary data.

4. How to execute project

To run this code i suggest executing it line by line. If you don't do this it might sometimes produce errors. Because the Kaggle competiton requires you to load a csv file onto their servers to check your accuracy rating we save away our predictions to a csv file with the following code:

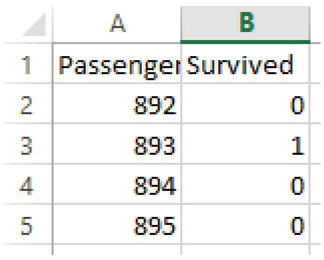
```
#Submission
#We create a new dataframe for the submission
submission = pd.DataFrame()

submission["PassengerId"] = new_test["PassengerId"]
submission["Survived"] = final_pred

#We save the submission as a '.csv' file
submission.to_csv("Results/titanic_cleanedup_engineered_kmeans4.csv",
index=False)
```

You can upload this file on Kaggle to check how well you did compared to other people.

The folder also contains an other python file made by a fellow kaggle competitor from which i learned alot. You can use this file to get an even higher score.

5. Results

The generated csv file has to look like this to comply with the kaggle standards.



	A	B
1	Passenger	Survived
2	892	0
3	893	1
4	894	0
5	895	0

5_Movie_recommendation

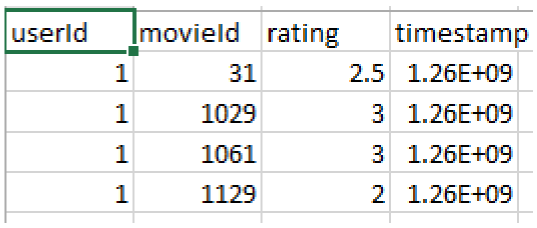
1. Business Problem Description

An online video streaming service wants to give good recommendations to users on movies they haven't seen yet based on their previous reviews and ratings given by other users. We're going to build a simple recommendation system that will be able to do all this.

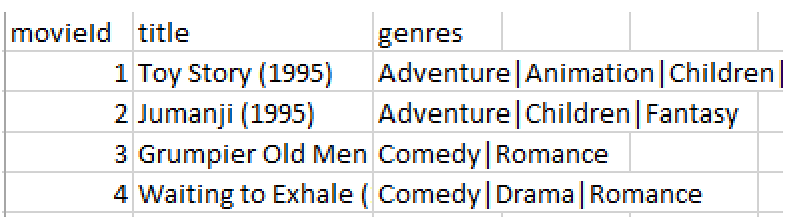
2. Dataset + Goal

We will be using the MovieLens dataset which you can freely download on this link: https://grouplens.org/datasets/movielens/

I have provided the necessary data files that we will be using in this project in the respective folder so you won't have to download anything yourself. The dataset we're using is the ml-latest-small which includes a list of users and the ratings they have given in the past to specific movies. It also includes a list of all the movies available and their genre. We will be combining both datasets to correlate user ratings and make accurate predictions on which movies a user could also like.



userId	movieId	rating	timestamp
1	31	2.5	1.26E+09
1	1029	3	1.26E+09
1	1061	3	1.26E+09
1	1129	2	1.26E+09



movieId	title	genres
1	Toy Story (1995)	Adventure|Animation|Children|
2	Jumanji (1995)	Adventure|Children|Fantasy
3	Grumpier Old Men	Comedy|Romance
4	Waiting to Exhale (	Comedy|Drama|Romance

3. Model

I have included 2 different Python files for this project. Both are able to give good recommendations to users. I will explain them both in this next section:

'recommendation.py' is the simplest model out of the two. The model will recommend the highest predicted rating movie that the user hasn't seen yet. In other words, it values the mean rating of a movie and then compares it to the kinds of movies the user likes based on title and genre. This model will not try to find other users who liked the same movies but rather just recommend generally high rated movies based on genre.

```
# Recommendation function
def recommend_movies(predictions_df, userId, movies_df, original_ratings_df,
num_recommendations=5):

    # Get and sort the user's predictions
    user_row_number = userId - 1 # UserID starts at 1, not 0
    sorted_user_predictions =
predictions_df.iloc[user_row_number].sort_values(ascending=False)

    # Get the user's data and merge in the movie information.
    user_data = original_ratings_df[original_ratings_df.userId == (userId)]
    user_full = (user_data.merge(movies_df, how = 'left', left_on = 'movieId', right_on =
'movieId').
                     sort_values(['rating'], ascending=False)
                 )

    print ('User {0} has already rated {1} movies.'.format(userId, user_full.shape[0]))
    print ('Recommending the highest {0} predicted ratings movies not already
rated.'.format(num_recommendations))

    # Recommend the highest predicted rating movies that the user hasn't seen yet.
    recommendations = (movies_df[~movies_df['movieId'].isin(user_full['movieId'])].
         merge(pd.DataFrame(sorted_user_predictions).reset_index(), how = 'left',
               left_on = 'movieId',
               right_on = 'movieId').
         rename(columns = {user_row_number: 'Predictions'}).
         sort_values('Predictions', ascending = False).
                       iloc[:num_recommendations, :-1]
                      )

    return user_full, recommendations
```

'collaborative_filtering.py' is a slightly more complex model. It starts off working in the same way the first model works but in addition to just looking at what genres a user likes this model will also use that data to start looking for users with similar tastes. Collaborative filtering is a way of predicting user-item scores based on existing scores. It is able to do this without having to manually define new features and it is able to find hidden correlation features that you otherwise wouldn't have been able to find.

```
# Take list of user ratings and correlate them with all other ratings to return list of
recommended movies
def get_movie_similarity(movie_title):
    '''Returns correlation vector for a movie'''
    movie_idx = list(movie_index).index(movie_title)
    return corr_matrix[movie_idx]

def get_movie_recommendations(user_movies):
    '''given a set of movies, it returns all the movies sorted by their correlation with the
user'''
    movie_similarities = np.zeros(corr_matrix.shape[0])
    for movie_id in user_movies:
        movie_similarities = movie_similarities + get_movie_similarity(movie_id)
    similarities_df = pd.DataFrame({
        'movie_title': movie_index,
        'sum_similarity': movie_similarities
        })
    similarities_df = similarities_df[~(similarities_df.movie_title.isin(user_movies))]
    similarities_df = similarities_df.sort_values(by=['sum_similarity'], ascending=False)
    return similarities_df
```

4. How to execute project

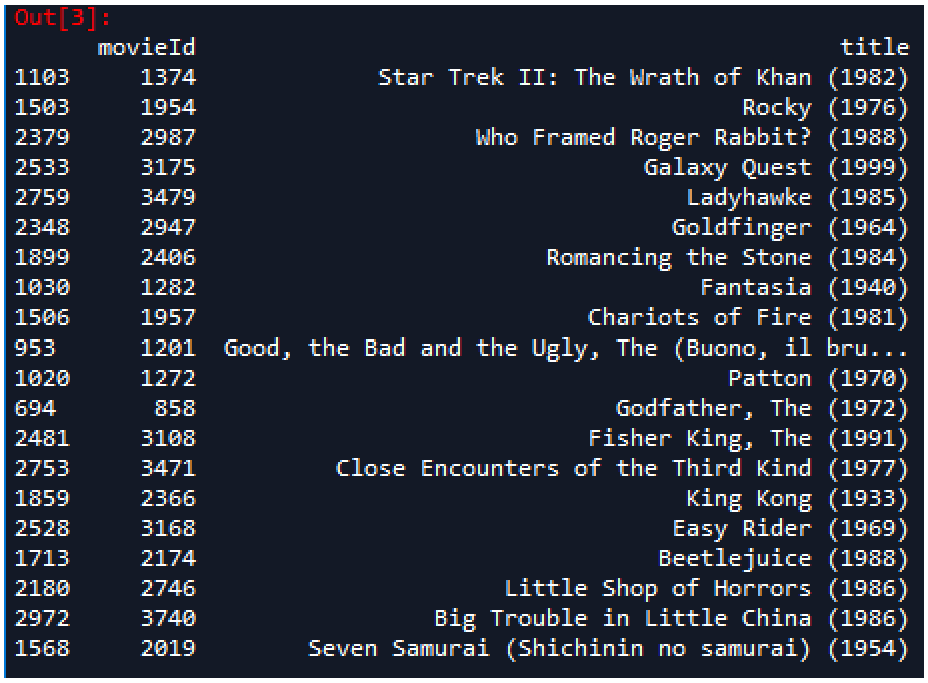
Just run the full code and Python will do the rest! If you want to make recommendations for a new user just change the number in the following code to the one corresponding to your user in the data. You can also make recommendation for yourself by adding in a new user to the 'ratings.csv' file and giving some ratings for movies you have already seen!

```
sample_user = 672
```

```
selected_user = 672
```

5. Results

After have tested both models on different users i found that both were able to give accurate recommendations. Depending on the user and the model sometimes the recommendations would fit perfectly but other times the system would generalize a users tastes and just recommend well known a nd liked movies. After the code is done the recommendations will show up in your console:



```
Out[3]:
      movieId                                              title
1103     1374            Star Trek II: The Wrath of Khan (1982)
1503     1954                                      Rocky (1976)
2379     2987                   Who Framed Roger Rabbit? (1988)
2533     3175                               Galaxy Quest (1999)
2759     3479                                 Ladyhawke (1985)
2348     2947                                Goldfinger (1964)
1899     2406                        Romancing the Stone (1984)
1030     1282                                  Fantasia (1940)
1506     1957                          Chariots of Fire (1981)
953      1201  Good, the Bad and the Ugly, The (Buono, il bru...
1020     1272                                    Patton (1970)
694       858                             Godfather, The (1972)
2481     3108                           Fisher King, The (1991)
2753     3471        Close Encounters of the Third Kind (1977)
1859     2366                                 King Kong (1933)
2528     3168                                Easy Rider (1969)
1713     2174                               Beetlejuice (1988)
2180     2746                      Little Shop of Horrors (1986)
2972     3740                    Big Trouble in Little China (1986)
1568     2019           Seven Samurai (Shichinin no samurai) (1954)
```

6_NMIST_tutorial_exercise

1. Business Problem Description

This project is part of a codelab by Google. We will learn how to build and train a neural network that recognises handwritten digits. The codelab can be found via this link: https://codelabs.developers.google.com/codelabs/cloud-tensorflow-mnist/#0

2. Dataset + Goal

This project uses the well known MNIST dataset. This dataset consists of 60,000 labeled handwritten digits for us to train our neural networks on. Our goal is to create a functional neural network that is capable of accurately predict what number is written down.

Training digits

- The digits highlighted in red are the numbers on which the neural network failed to accurately predict the correct digit.

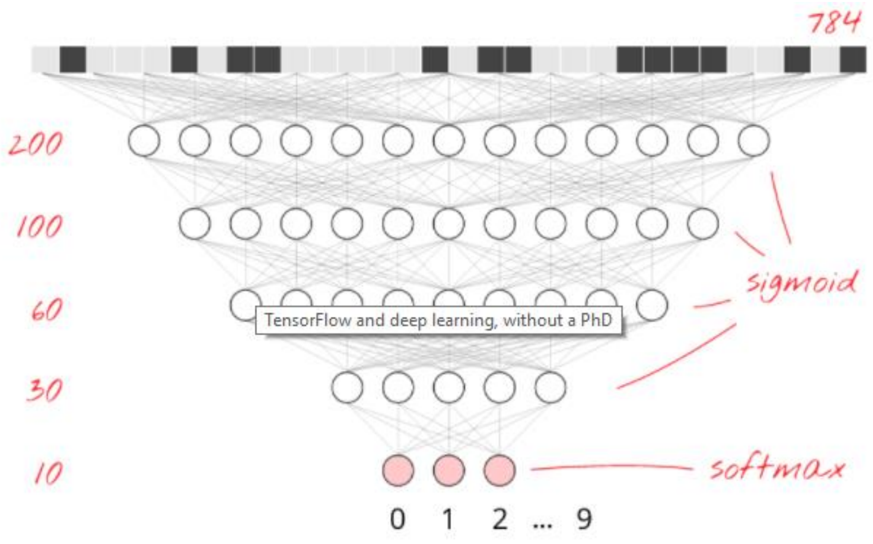
3. Model

We used both an Artificial Neural Network and a Convolutional Neural Network for this project. The codelab of which i provided the link above gives a great in depth explanation of how these models work in theory and also a tutorial on how to implement and improve them. I greatly recommend checking it out.

Here is a quick explanation on how they work if you don't have the time to go through the codelab yourself.

The first model we used is a 5 layered ANN using the ReLu activation function and we also applied dropout to keep out model from overfitting on our training dataset. Overfitting means the model will preform very well on our training dataset but will most likely struggle with new data.

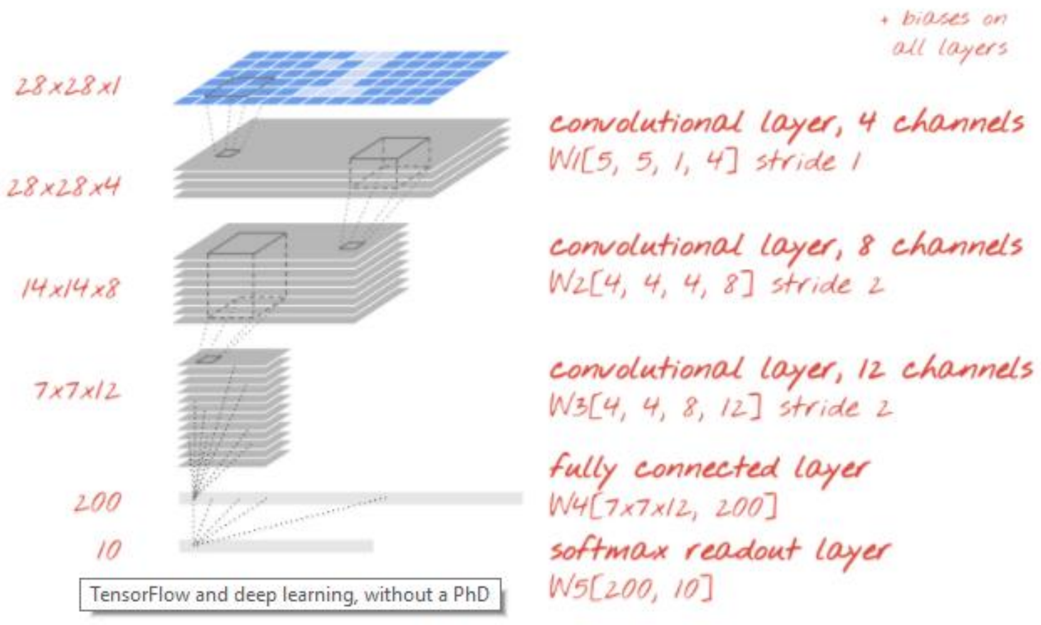
A representation of how this network looks:



From top to bottom it takes the images of the digits (represented by the squares) through all the different layers and node (the circles) to finally create an output of hopefully the corresponding number.

As we've learned from the 'cat_or_dog' case, if we are working with image recognition a CNN is usually the best choice. Thats why we also built another CNN to try and see if it really does preform better than our ANN for this case. The CNN uses 3 convolutional layers and 1 fully connected layer, it also uses the ReLu fuction and dropout to prevent overfitting just like our ANN. This CNN works by transforming our images into pixels and pushing this through the conv layers to try and predict the number.

A representation of the CNN:



If you want a better, quick and live representation of how convnets work I highly suggest checking out this website: http://scs.ryerson.ca/~aharley/vis/conv/

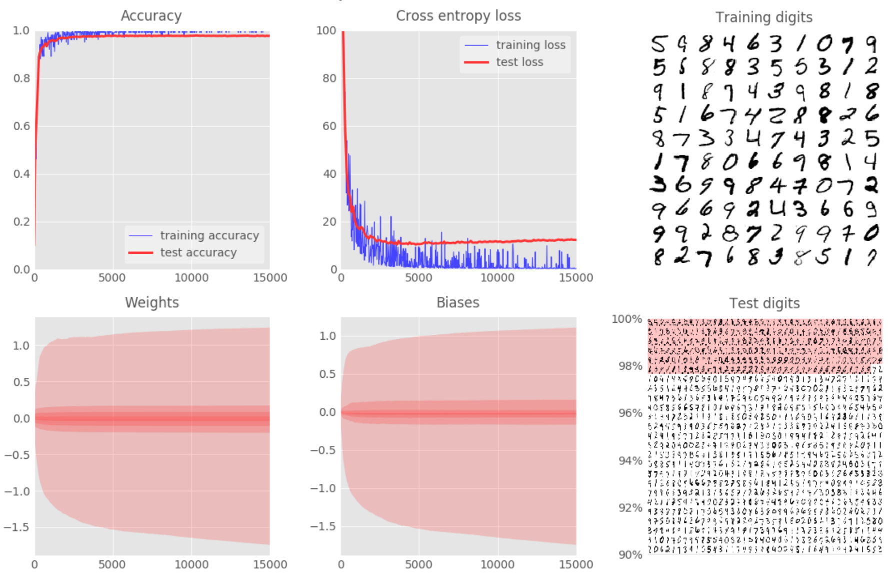
4. How to execute project

This project is very easy to run. Simply make sure that you are in the correct working directory and run the full code. If you don't have the data yet the code will download it for you. If you get an error telling you're missing certain modules or libraries please refer to the debugging section at the start of this document.

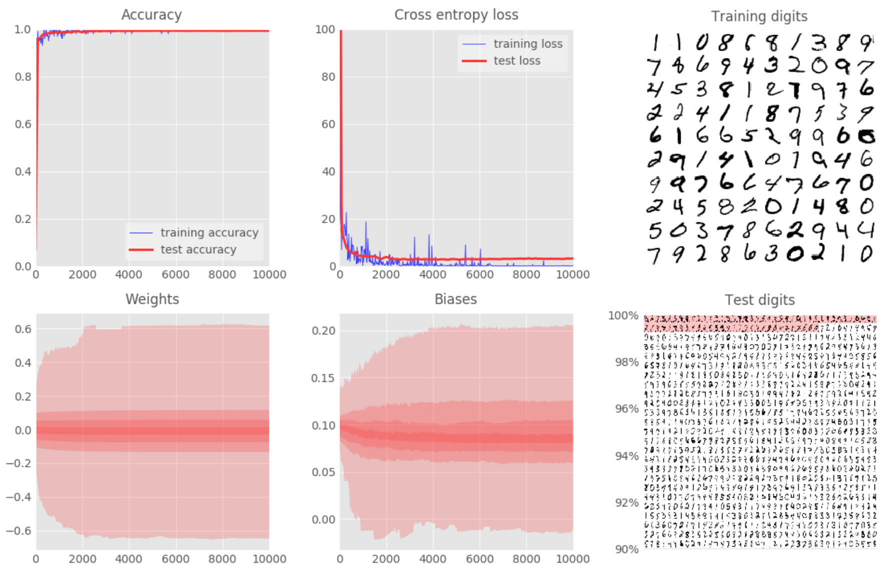
5. Results

If you run the code it will show you a live representation of the learning process of the neural network. Also please be aware that the CNN will take quite a bit longer to finish learning than the ANN but we will also see that the CNN beats the ANN in accuracy by a large margin.

The ANN with a test accuracy of 97.6%



The CNN with a test accuracy of 99.3%



7_DQN_self_driving_car

1. Business Problem Description

We want to build simulation of a self driving car that is able to travel from point A to point B while avoiding all obstacles or by following a predetermined road. The car needs to able to learn by itself without the use of training data.

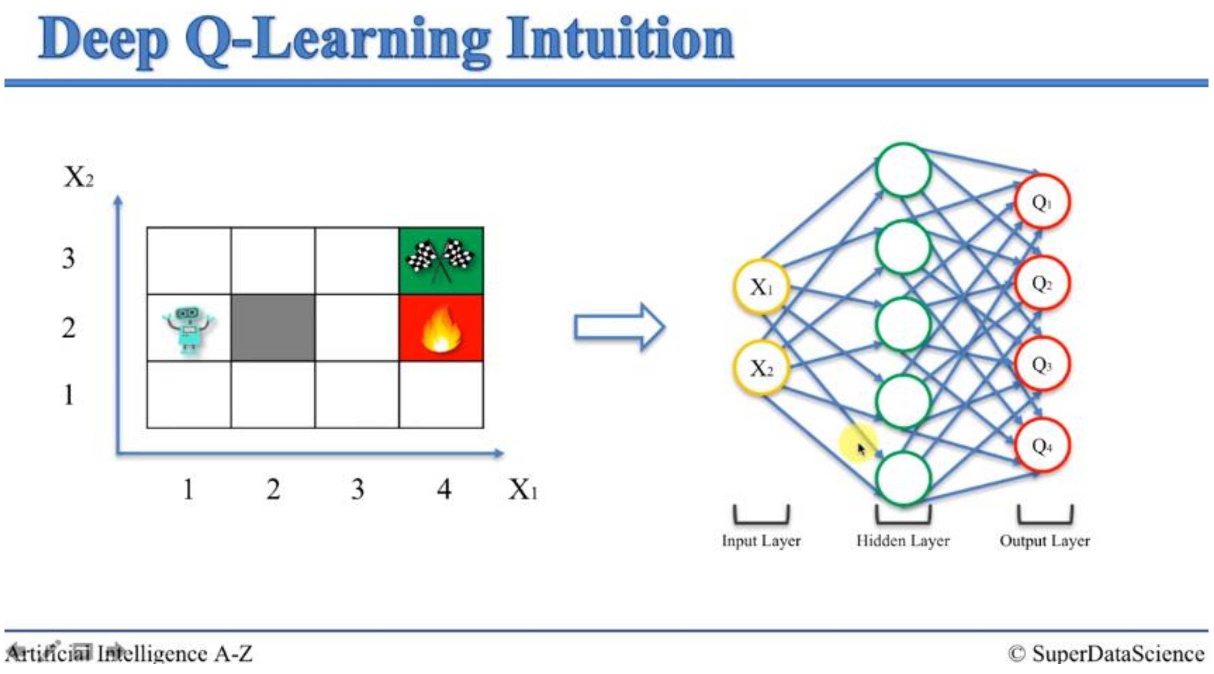
2. Dataset + Goal

This project does not use a dataset. Instead it will learn from experience to travel between two points and how to avoid obstacles in its way. We will be able to save the brain of the car for later use so the training will not go to waste.

The environment to create this project was provided by Udemy during their online course "artificial-intelligence-az" found here: https://www.udemy.com/artificial-intelligence-az/learn/v4/overview

3. Model

We will be using Deep Q-Learning to create our self driving car in this project. Q-Learning is a reinforcement technique. Reinforcement learning (RL) is the act of learning by trial and error. The car will be able to drive by itself without us having to provide any hand-engineered features. The way this works is comparable to the way humans learn. The car will start performing actions. If he does something good like reaching the goal he will recieve a reward. If the car does something bad like hitting an obstacle or a wall it will recieve a punishment. With this data it will require from its sensors it will start to learn what is right or wrong and it will use these experiences to avoid doing anything it isn't supposed to.

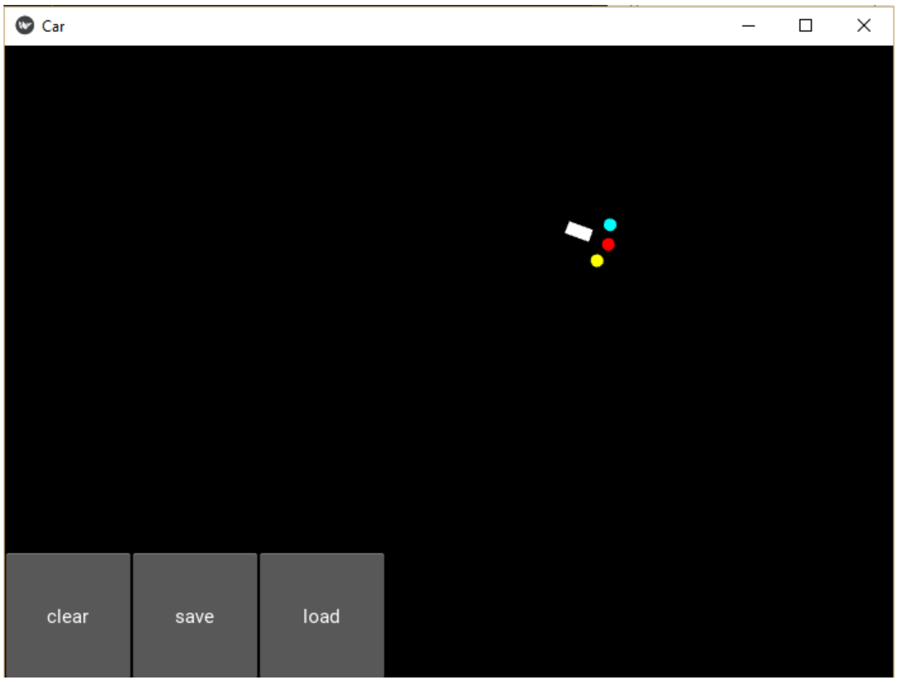


If you take a look at this slide provided by SuperDataScience it might be easier to understand how this works. The little robot in the left image will try to reach to goal in the top right. If it reaches the goal the robot will recieve a score of +1 but if it hits the fire it will recieve a punishment of -2. The robot will always try to get the highest score possible. After a few attempts the robot will put his experience through the neural network to predict the best moves he should make to get the highest score. This is an easy explanation of how Deep Q-Learning works.

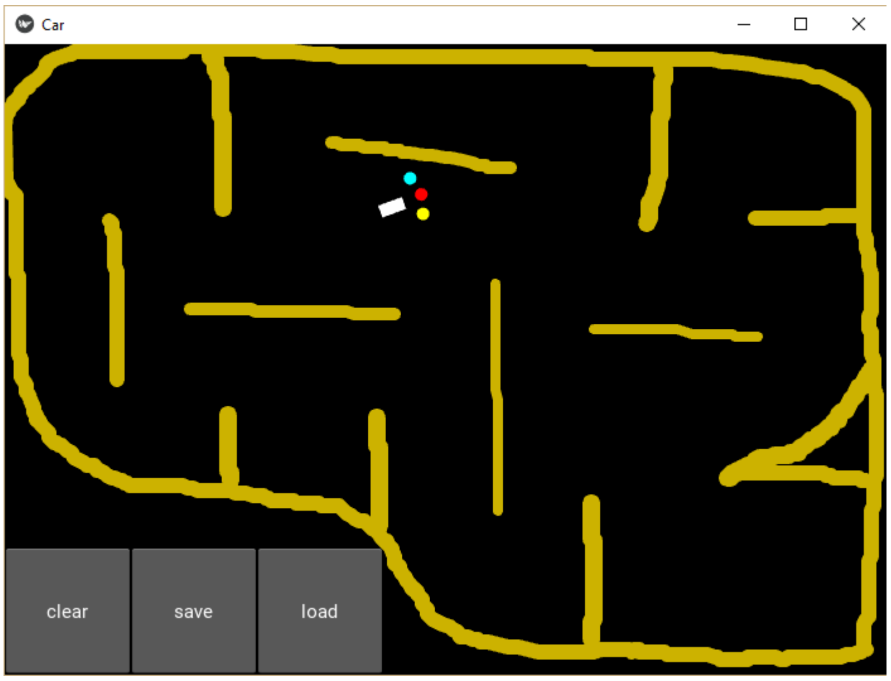
4. How to execute project

This project requires you to have PyTorch installed. If you are using Windows please refer to my instructions on how to get PyTorch working in the "Software installation" section at the start of this document.

Once you have all the required libraries and modules and are in the correct working directory you can run the self driving car widget by executing the code in the "map.py" file.



The car's goal is to travel from the upper left corner to the bottom right corner. Over time it will start learning how to do this by itself. Once you notice the car is comfortably able to reach his goal you can start drawing a road or obstacles for the car to avoid by holding and dragging your left mouse button.

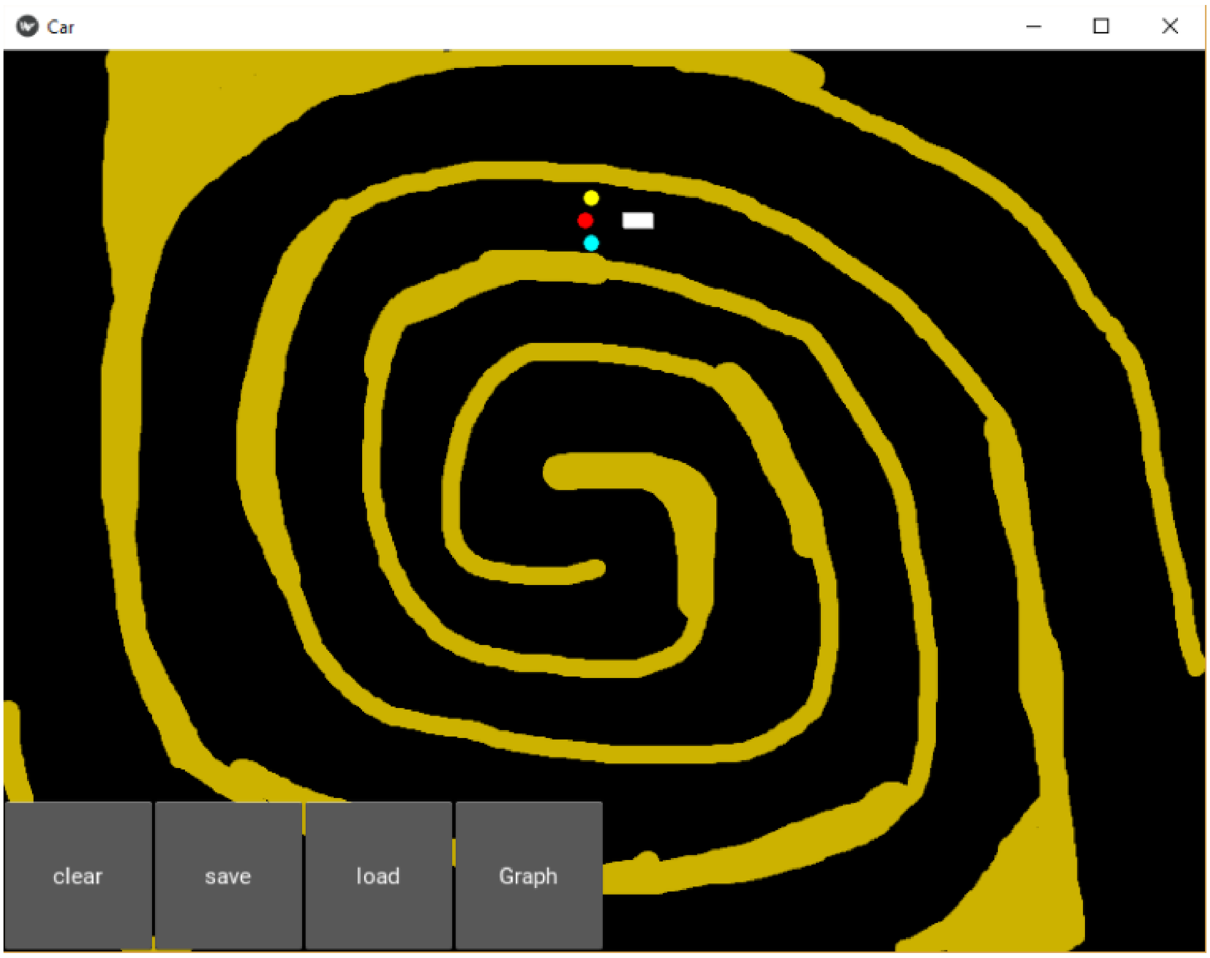


5. Results

Don't worry if your car still goes through the yellow walls sometimes. This AI is a very basic example and is definitely not perfect. If the car continues to make a lot of mistakes after a few minutes training I would suggest making the yellow lines a bit thicker or just drawing a new road altogether. If the road is too complex it will fail to learn the correct path.

If you are satisfied with how your car is driving you can hit the save button to store away its currently trained brain for later use. Just hit the load button and you will be able to continue the training where you left it. I have already provided a pre-trained brain that might give you a little boost if you are running the widget for the first time.

After some time training and maybe tinkering a bit more with the AI we should have a self driving car that is able to navigate through a complex road like this one made by a fellow student Igor:


Car
clear
save
load
Graph