# Using Machine Learning to Switch Lanes in Autonomous Cars

Claire Savard
Vandana Sridhar
Vikas Hanasoge Nataraja

**Problem Space:**

The development of vehicles is arguably one of the greatest developments in human history. Vehicles allows for convenient travel which greatly saves time from previous modes of transportation. However, an increase in vehicle ownership bring about many accidents that can cause great harm to the population, both vehicle owners and non-owners. Operating cars, buses, motorcycles, and more, is inherently dangerous and requires skill and concentration from drivers to ensure the safety of their environment. Quick decisions are constantly being made by drivers and judgment errors are often the reason behind a majority of motorized accidents.

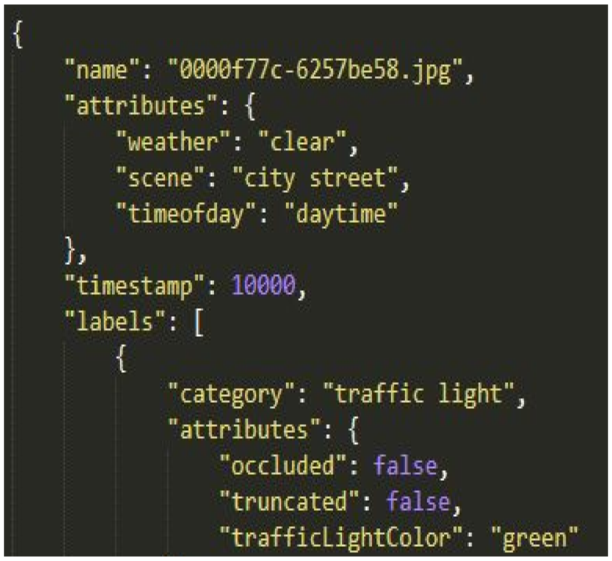
With the major developments in AI and machine learning today, the ability to minimize vehicle accidents and give more convenience to vehicle owners is evident with autonomous cars. The inspiration behind this technology is to improve safety and efficiency by automating the decision processes required when driving to minimize error. Machine learning is being used in many different tasks applied to autonomous cars, such as road sign detection, yellow light decisions at intersections, and determining right-of-way in multiple scenarios. Reasons for this include the complexity of these tasks and different environments, the quick decisions that need to be made, and the ability to try and train a car to "think like a human".

A major task in autonomous vehicles is obstacle avoidance, which can oftentimes be surpassed by lane switching. Generally, this is done using radars to gain perception of the surrounding environment, then deciding whether to change lanes or not. Using image labeling from the front of the car in tandem with the the classic radar imaging can give the car more information on switching lanes is a good idea or not. For example, seeing a sign ahead which indicates the merging of the lane should indicate that the vehicle should not change into that merging lane. This project will focus on applying machine learning techniques to image classification in front of the car in order to gain more information on whether changing lanes in possible and a good idea given the environment.

**Dataset:**

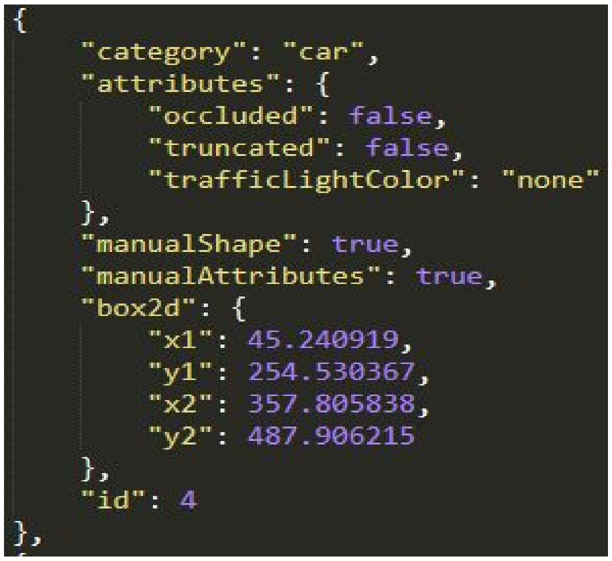
Our dataset came from UC Berkeley DeepDrive, a research group working on computer vision and machine learning algorithms for autonomous cars. In May 2018, they released the largest and most diverse driving dataset complete with annotations called "BDD100K". The dataset is very rich and vast, covering a variety of scenarios and tracking multiple parameters. There are over 100,000 images available in this dataset which includes training examples, test and validation examples. However, the dataset does not have any output variables. This is because it is an open-ended raw dataset with data taken from a single camera. We will, therefore, be labeling the output variables ourselves. We have 1300 entries in our parsed dataset. The data, in general, has 10 different kind of objects - bus, traffic lights, traffic signs, person, bike, truck, motor, car, train and rider. These objects are given with the name of the particular object and only its coordinates in the image. Next we get lane coordinates and driveable area coordinates as well as weather, time of day and scene attributes. The lanes are categorized into type, style and direction while the driveable area is categorized into direct or alternative. For the purposes of this project, the output variable is the binary decision to change lanes with 1 indicating it is okay to switch lanes, 0 indicating not okay to change lanes.

The total number of original features were 23 (not including sub-features), number of entries was 1300. While this was later changed, this information pertains to the original dataset from DeepDrive. The images below show the original dataset in JSON format.



```
{
    "name": "0000f77c-6257be58.jpg",
    "attributes": {
        "weather": "clear",
        "scene": "city street",
        "timeofday": "daytime"
    },
    "timestamp": 10000,
    "labels": [
        {
            "category": "traffic light",
            "attributes": {
                "occluded": false,
                "truncated": false,
                "trafficLightColor": "green"
```

Figure 1: Image name and traffic light data



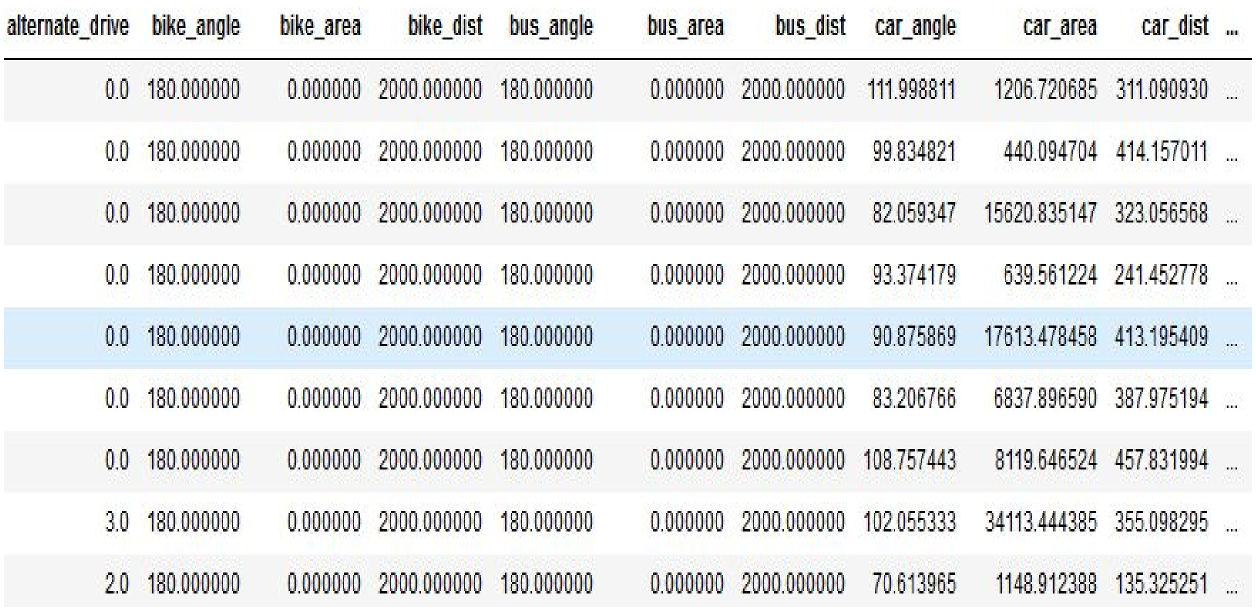
```
{
    "category": "car",
    "attributes": {
        "occluded": false,
        "truncated": false,
        "trafficLightColor": "none"
    },
    "manualShape": true,
    "manualAttributes": true,
    "box2d": {
        "x1": 45.240919,
        "y1": 254.530367,
        "x2": 357.805838,
        "y2": 487.906215
    },
    "id": 4
},
```

Figure 2: Car (object) data

**Approach:**

Instead of directly using the dataset in a machine learning model, we used feature engineering methods including feature reduction to better understand the data from an algorithmic perspective. Certain combination of features were reduced to different numeric values which would provide the model with a better distinction between images and different features.

The initial challenges started with the dataset which in turn arose from the fact that the data file was a nested JSON file (COCO format) meaning each feature had sub features. It was also not dimensionally symmetric meaning each image had different number of detections and therefore unequal number of features. This made it all the more difficult to read the data into the program. Conversion to other formats proved futile as well. This meant feature engineering would play a key role in our interpretation of the data. In the end, we used pandas and its dataframe to construct the dataset in a readable manner. The next step was to condense some features to a more useful format because, for example, just using the object coordinates in the dataset would not prove to be useful for a machine learning model because it does not convey anything palpable. So, we reduced object coordinates to three parameters - area, angle and minimum distance from the reference point of the camera Any outlier cases (i.e. cases like object not being present in that entry) were dealt by assigning extreme values to these 3 parameters. This would help the algorithm to recognize an outlier from a non-outlier. In each category, we narrowed the number of instances down to a single object based on distance from the camera frame of reference since that would be the one immediately assessed during a lane change (single nearest car, single closest bus, etc.). We chose these 3 parameters to be features because they accurately describe the position of the object in the image and any changes could be easily compared. Next, we converted the string values in the data (the columns having non-numeric data) to a quantifiable value. We achieved this using categorical encoding (we also tried one-hot encoding which did not significantly improve results). The resulting formatted dataset had 43 features not including the outcome variable. It is a balance between feature reduction and preventing loss of any information. Screenshots of the original dataset are shown in figures 1 and 2, that of the formatted dataset is shown in figure 3. The full original raw dataset from UC Berkeley can be accessed here. The complete formatted dataset is here. A visualization of the outcome variable is shown below in figure 4:



| alternate_drive | bike_angle | bike_area | bike_dist | bus_angle | bus_area | bus_dist | car_angle | car_area | car_dist | ... |
|---|---|---|---|---|---|---|---|---|---|---|
| 0.0 | 180.000000 | 0.000000 | 2000.000000 | 180.000000 | 0.000000 | 2000.000000 | 111.998811 | 1206.720685 | 311.090930 | ... |
| 0.0 | 180.000000 | 0.000000 | 2000.000000 | 180.000000 | 0.000000 | 2000.000000 | 99.834821 | 440.094704 | 414.157011 | ... |
| 0.0 | 180.000000 | 0.000000 | 2000.000000 | 180.000000 | 0.000000 | 2000.000000 | 82.059347 | 15620.835147 | 323.056568 | ... |
| 0.0 | 180.000000 | 0.000000 | 2000.000000 | 180.000000 | 0.000000 | 2000.000000 | 93.374179 | 639.561224 | 241.452778 | ... |
| 0.0 | 180.000000 | 0.000000 | 2000.000000 | 180.000000 | 0.000000 | 2000.000000 | 90.875869 | 17613.478458 | 413.195409 | ... |
| 0.0 | 180.000000 | 0.000000 | 2000.000000 | 180.000000 | 0.000000 | 2000.000000 | 83.206766 | 6837.896590 | 387.975194 | ... |
| 0.0 | 180.000000 | 0.000000 | 2000.000000 | 180.000000 | 0.000000 | 2000.000000 | 108.757443 | 8119.646524 | 457.831994 | ... |
| 3.0 | 180.000000 | 0.000000 | 2000.000000 | 180.000000 | 0.000000 | 2000.000000 | 102.055333 | 34113.444385 | 355.098295 | ... |
| 2.0 | 180.000000 | 0.000000 | 2000.000000 | 180.000000 | 0.000000 | 2000.000000 | 70.613965 | 1148.912388 | 135.325251 | ... |

Figure 3: Formatted dataset

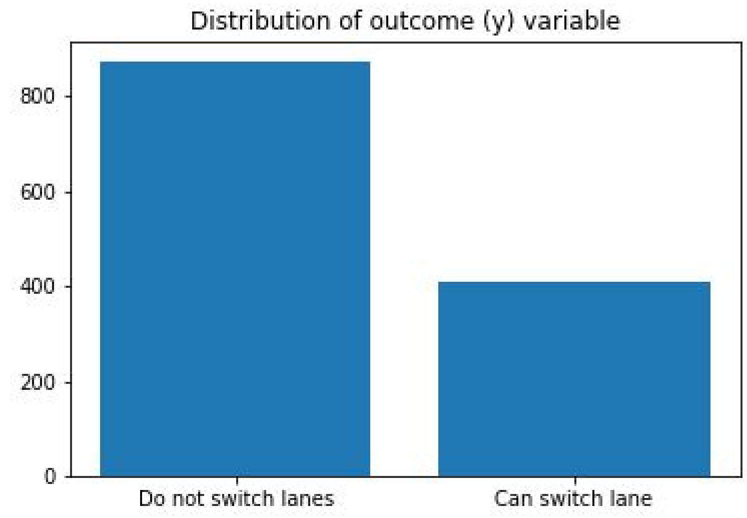


Figure 4: Distribution of outcome variable

For the algorithms, due to the nature of our dataset, particularly that the features were not exactly independent, we decided to start with Decision Trees and Random Forest algorithms. We also eventually tried a number of other model including Logistic Regression, K-Nearest Neighbors and SVMs. We followed this by implementing ensemble learning methods like Adaboost and Gradient Boosting. The details of the accuracies are explained in the results section. The boosting methods performed the best primarily because many of the features have extreme values (high difference of value) and therefore the wrong classifications were penalized more yielding better accuracy.

In a broader machine learning context, the learning rate and the number of estimators for the boosting algorithms were noticeably affecting the convergence of the prediction. This also meant that hyperparameter tuning was very important. Using cross-validation, we were able to tune the hyperparameters to yield the highest accuracies (or lower error rates). We used K-Fold cross validation with 5 folds for most models as this configuration produced the best results with a reasonably good split between the training and testing set.

**Results:**

We trained our model with several machine learning algorithms and we obtained the best accuracy with the Gradient Boosting Classifier algorithm. It provides us with an accuracy of 82.49% on the test set, owing to the classifier's large number of estimators. The number of estimators denote the number of boosting stages to perform and Gradient Boosting is fairly robust to

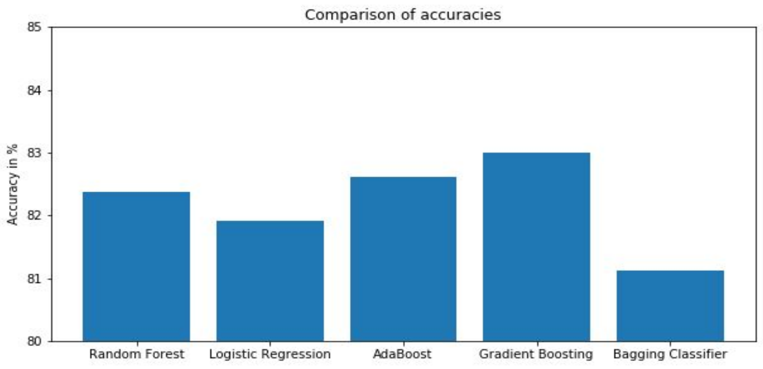
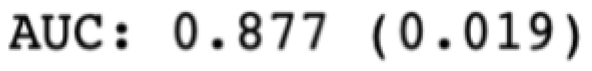


Figure 5: Comparison of accuracies for different models

overfitting. Increasing the number of estimators results in better performance. The second in line in terms of accuracy and performance is the Random Forest which provided us an accuracy of 82.38%. Similar to ensembles, Random Forests depend on number of estimators and the maximum depth parameters for their efficiency. The maximum depth shouldn't be assigned too large a value since this might overfit the dataset. The number of estimators for the Random Forest algorithm can range between 10 to 100 for optimal performance. Other algorithms we tried out were Logistic Regression (0.819) , Decision Trees(0.811) and the Adaboost ensemble algorithm (0.817) which provided similar accuracies, differing in one thousandth of a decimal point.

Some classification metrics used to evaluate the predictions for this problem space is the area under the ROC curve, the accuracy on the test set which is already mentioned above, a confusion matrix to present the classification accuracies and a final classification report which provides a detailed explanation of a couple measures such as precision, recall , F1- score and support.

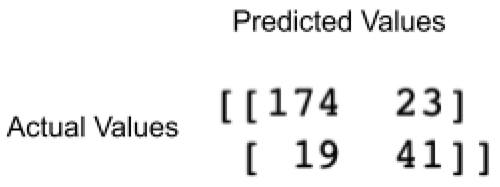
**Area under the ROC curve:** This is a performance measure for binary classification problems. The AUC exhibits the model's ability to discriminate between positive and negative classes. An area of 1 can be interpreted as a perfect model that made all the predictions correctly and a value of 0.5 denotes a good random model. ROC curves can be divided into two parameters, namely: Sensitivity and specificity. Sensitivity is the true positive rate or commonly called as Recall. It is the number of instances from the positive class that are predicted correctly. In our problem space, this would denote the scenarios where the car could switch lanes and the model correctly predicts this. Specificity is called the true negative rate, where the model correctly predicts the instances where the car cannot switch lanes. The area obtained under the ROC curve while using the Gradient Boosting algorithm is 0.877 which is relatively close to 1 and greater than 0.5 suggesting some efficiency in prediction (figure 6).



```
AUC: 0.877 (0.019)
```

Figure 6: Area under the curve for gradient boosting tree

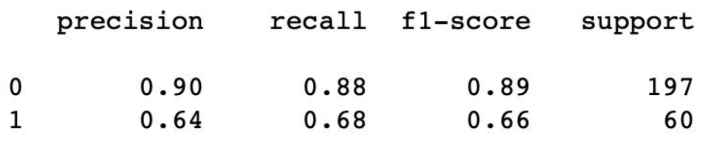
**Confusion Matrix:** This presents the accuracy of a model with two or more classes. Predictions are denoted on the x-axis and the actual outcome is displayed on the y-axis. The majority of the predictions done by the model, shown in figure 7, is along the diagonal line of the matrix which shows that the model has correctly predicted them.



```
[[174   23]
 [ 19   41]]
```

Figure 7: Confusion matrix for gradient boosting tree

**Classification report:** A classification report gives a good idea of accuracy using numerous measures. The classification report module from the scikit-learn library discusses measures such as precision, recall, F1-score and support. Precision is defined as the accuracy of positive predictions. It calculates the true positive predictions as a ratio of both the true positive and false positive predictions. Recall aka sensitivity is the overall true positive rate which obtains the number of instances from the positive class which is predicted correctly. F1-score considers both precision and recall and takes their harmonic mean. The last parameter of the report is called support which provides the number of occurrences of each label in the y_true dataset.



```
     precision    recall  f1-score   support

0         0.90      0.88      0.89       197
1         0.64      0.68      0.66        60
```

Figure 8: Classification report for gradient boosting tree

From the statistics shown in figure 8, the Gradient Boosting classification model provides a precision of 90%  on class 0 and 64% on class 1. The precision is the ability of the classifier not to label as positive a sample that is negative. Hence the model correctly predicted 90% of the instances where the car cannot switch lanes when those instances were labeled as 0. Similarly the model predicted only 64% of the instances where the car could definitely switch lanes given those instances had a label of 1. This could be due to the fact that the some instances may have been unclear for the model to decide upon.  The model provides a recall of 88% for class 0, which is a percentage of correctly predicted scene instances where the car cannot switch lanes. The support shows that there are 197 instances under label 0 ( car cannot switch lanes) and 60 instances with label 1 ( car can switch lanes) in the true y test set.

From the statistics above, it is clear the model possesses a good understanding of the problem space and intelligently decides whether or not to switch lanes. However this interpretation is from one perspective of the car. Given all four perspectives, the model could be extended and could thereby make a collective decision on lane switching which is more appropriate and astute. Currently we do not have a baseline model to compare our results against. Additionally, most of the work implemented in the area of autonomous vehicles is done using dynamic image processing. Our current project encompassses static images from one perspective of the car and the accuracy obtained by the model isn't significantly high enough to make meaningful changes in the domain yet.

**Discussion:**

Although the accuracies shown may be fairly high, they are not high enough to make any significant changes in the autonomous cars field. Due to the high consequences when a mistake is made by autonomous cars, like jeopardizing the passenger's safety, algorithms should only be implemented when their likelihood of failure is very small. Initial implementations of machine learning on labeled images is promising, but will require much more work if this technology wants to be used in actual cars. However, there are many ways that this work can be furthered and reasons as to why using cameras for lane-switching may be a preferable option.

This work is a stepping stone to many more things that should be studied for this task. The possibly most challenging task in upping the efficiency for these algorithms would be spending more time working on feature engineering and finding the best features to work with and how to represent them. As explained in the approach section, issues arose with the data not having the same dimensions for each image. Therefore, we created new variables that could be generalized within each image which ensured a consistent number of features in each image. Unfortunately, these new variables were not necessarily independent from one another. For example, the minimum car distance and its angle and area made up 3 features but they were all describing the same car and were thus not independent. It would be very interesting to explore combining these features into one or turning them into subfeatures. In general, there are many different ways to take the data we worked with and create a new set of features to run through our lane-changing algorithms, and these would all have to be explored to deem the best set of features for this task.

After finding the proper features to work with, further research could include combining multiple images from around the car to give a complete 360 degree view of the car's space. This could then provide the car with the same amount of information as collected by current lidar technology used in autonomous cars, but would have many advantages. For one, cameras are much cheaper and more durable than current radar technology. They also require less power to run and take up less memory storage. Cameras are not live however, such as radars, but taking multiple photos within a short time span can create a time-varying scenario of the scene surrounding the car. With these advantages, it is worth continuing this research to see whether cameras can compete with current lidar technology. It is also worth mentioning that cameras are much cheaper than radars, and can thus allow autonomous cars to be more accessible to the general public or give room for investing more in other aspects of the car that may need it.

In summary, although the results shown are not up to the level needed to be seriously considered in the field, these initial results prove that it is worth continuing this research. This type of technology may end up being more efficient than current lane-switching techniques with cuts in cost, memory storage, and energy required. Future work in feature engineering and testing with more cameras will be important in the success of this method. This may end up being a very large task, but the turn-out could be great and exciting to the growing field of autonomous cars that may become a norm in our future!

**References:**

[1] Xing, Shiyu and Jakiela, Mark, "Lane Change Strategy for Autonomous Vehicle" (2018). Mechanical Engineering and Materials Science Independent Study. 61. https://openscholarship.wustl.edu/mems500/61
[2] Pek, Christian and Zahn, Peter and Althoff, Matthias, "Verifying the Safety of Lane Change Maneuvers of Self-driving vehicles Based on Formalized Traffic Rules". (2018). http://mediatum.ub.tum.de/doc/1379669/794156.pdf
[3] Atagoziyev, Maksat and Schmidt, Klaus W. and Schmidt, Ece G., "Lane Change Scheduling for Autonomous Vehicles". (2016).
https://ac.els-cdn.com/S2405896316302063/1-s2.0-S2405896316302063-main.pdf?_tid=f23523c4-509e-41fe-80e4-e6dc9cf57c97&acdnat=1552098889_1b72109c2dab5bd8d74abb5c85098183
[4] Dang, Ruina & Wang, Jianqiang & Li, Shengbo & Li, Keqiang. (2015). "Coordinated Adaptive Cruise Control System With Lane-Change Assistance".
https://www.researchgate.net/publication/273181935_Coordinated_Adaptive_Cruise_Control_System_With_Lane-Change_Assistance