Traffic Image Classification

Kevin Chuang, 012480052

Overview

In this assignment, I developed a predictive model that can determine, given an image, which of 11 objects it is. The 11 objects include **car, suv, small_truck, medium_truck, large_truck, pedestrian, bus, van, people, bicycle, and motorcycle**. This report will document and provide the reasoning behind the techniques I used for feature selection, the implementation of the model, and the tuning of the model's hyperparameters.

Exploratory Data Analysis (EDA)

Data exploration is used to understand the data. In this step, I also split the training data into a **training set (80%)** & a **validation set (20%)**. The validation set (i.e. unseen test data) is set aside for model evaluation, and will be utilized after the optimal model hyperparameters are determined. The final model will be trained on all of the data (training + validation sets).

I dropped the duplicate instances with the same class labels, since they were contributing extraneous information and thus more noise to the dataset. There were around 23 duplicates that were removed. The classes are severely imbalanced. There are many ways to handle this, and I chose to use powerful algorithms to handle class imbalance. These algorithms handle class imbalance by essentially "balancing" them using weighting. The weighting is based on the class labels and is inversely proportional to the class frequencies in the input data (specifically n_samples / (n_classes * np.bincount(y))).

Data Preprocessing (Feature Engineering)

First, I scaled the input features, since the given features were on very different scales and concatenated from different image feature extraction techniques. This scaling is done because specific machine learning algorithms can only succeed when features are in a similar range or follow a normal distribution (i.e. **SVM**). If not, a feature with a variance that is orders of magnitude larger than others will largely dominate the objective function. I use **StandardScaler** to scale the input features by removing the mean and scaling to unit variance.

Then, I experimented with different feature selection techniques including principal component analysis (with different number of components based on kept variance), linear discriminant analysis (same process as PCA), locally linear embedding, and finally variance thresholding. Cross-validation was used to verify which feature selection technique was the best. The optimal feature selection technique was using **VarianceThreshold**, which I use to remove all features with 0 variance (i.e. remove features that have the same value in all samples). These features are considered unimportant, since features with low variance do not provide any useful information. This feature selection reduced **887 original features** to **48 features.**

I combined these two preprocessing steps into a **Pipeline** to automate, standardize and define the data preprocessing workflow.

Model Implementation, Evaluation, and Tuning

For the model, I utilized an ensemble method called the **VotingClassifier**. The goal of ensemble methods is to combine the predictions of several different base models in order to improve the robustness and generalization power over a single model. In addition, ensemble methods have been proven empirically to be powerful in competitions (such as kaggle). Within the **VotingClassifier**, I used a **Support Vector Machine** with **a radial basis function (Gaussian & non-linear) kernel**, a **Random Forest**, an **Extra-Trees classifier**, and finally, **a gradient boosting model** called **LightGBM**.

All of these models were individually tuned to their most optimal hyperparameters using **5-Fold cross-validation** and **Bayesian optimization**. Bayesian optimization tries to approximate or fit a Gaussian process (a regression model known as a surrogate model) on function/model evaluations in order to propose sampling points in the search space of a model's hyperparameters using acquisition functions. Essentially, it is a smarter grid search, and empirically, it has been proven to be more efficient than a grid search and more effective than a random search. After tuning each individual classifier, I ensemble these models using **VotingClassifier** with **soft** voting. In contrast to majority hard voting, **soft** voting uses the argmax of the sums of predicted probabilities to determine the class label, and is typically recommended for ensembles with well-calibrated models. The weights for each model in **VotingClassifier** were also tuned using **GridSearch** and 5-Fold CV, and it was determined the uniform weighting of each model was most optimal. Finally, I retrain the final model on the whole training set and use that for predicting the test set.

Rank & F1-score

My current rank on CLP public leaderboard is **1st** with a F1-score of **0.8598**.

Leaderboard

Rank	F1 (on 50%)	User ID	Submission Count
1	0.8598	12480052	12
2	0.8581	11459526	2
3	0.8554	12525123	18

Running the code

The script can be run by calling it with python in a terminal (i.e. **python3 traffic-image-classification.py**). The code assumes that **train.dat, test.dat, train.labels,** and **format.dat** are in the same directory as the script. The dependencies are located in **requirements.txt**. The script will produce a text file called **submission.txt** with the prediction results of the test set.