Data Challenge - Bike Sharing Demand Prediction

Kaggle Name: Evangelos Tsogkas

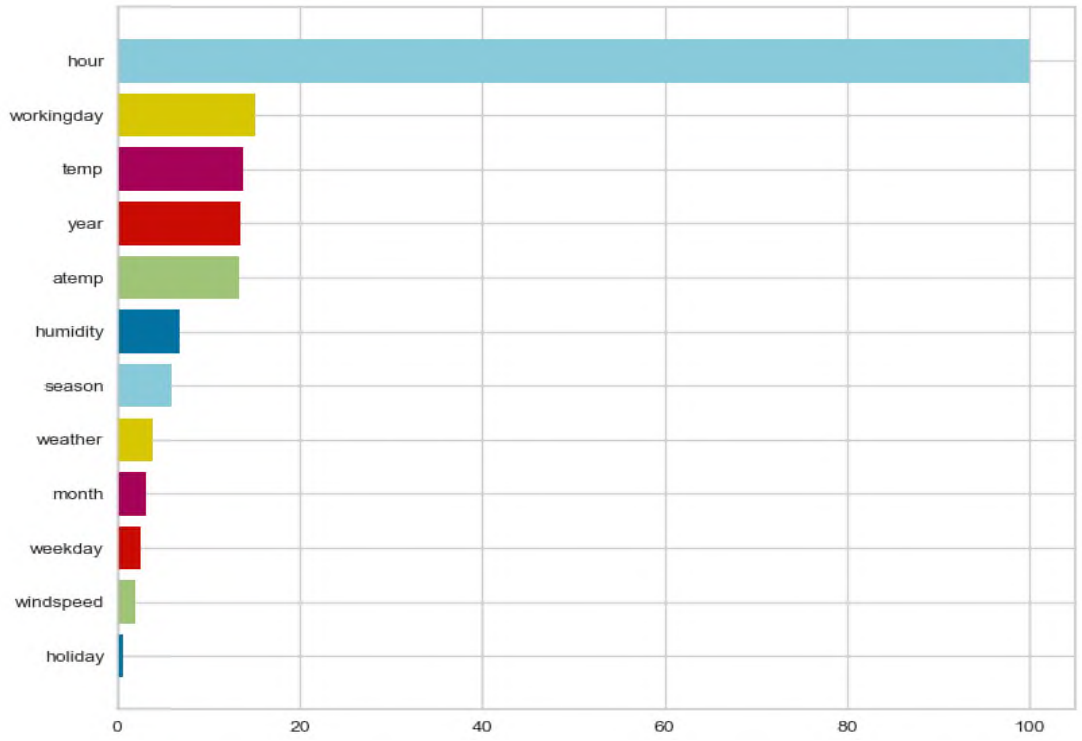
A.M: p3150185

1. Feature Selection

The feature selection is based on the work provided in the notebook of this challenge.

- 'atemp' is dropped as it is strongly correlated with 'temp' .
- 'windspeed' is dropped as it has little correlation with 'count' which we want to predict.
- 'casual', 'registered' and 'count' will all play their roles as target values, but only as target values and not as features, in order to avoid data leakage.

I also obtained the feature importances using an Extra Trees Regressor. It seems that the *'hour'* is really important and that the '*holiday*' although it's not so important it contributes even a little to a better prediction. In some models like Decision Tree by removing 'weekday' and keeping 'atemp' I got slightly better results, but on most regressors I trained, removing 'atemp' and especially 'windspeed' worked well enough.



2. Dealing with Categorical Features

Three approaches were tested:

- Keeping the label encoding
- One-hot encoding
- Cyclic encoding of temporal variables

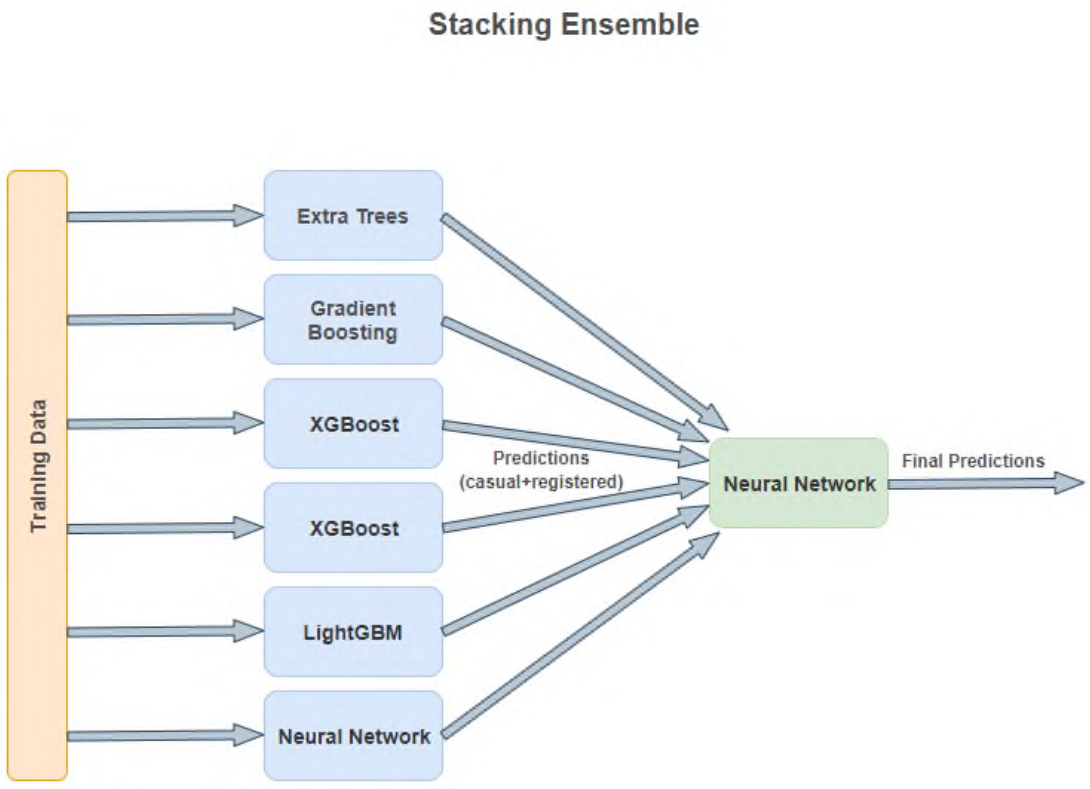
On tree based models the label encoding seemed to work the best, but the neural network is more sensitive so the one hot encoding was used for it.

The cyclic encoding had pretty good results too for the neural network, but one hot encoding was a bit better. Specifically, I tried the cyclic representation for the 'hour', 'weekday', 'month' and 'season' by using X, y locations on the unit circle.

I also tried excluding 'weather' from the one hot encoding process, since it has ordinal characteristics (greater value=worse weather), but one hot encoding it too worked better.

3. Model Building

The final model is a **stacking ensemble** consisting of 6 base regressors and one meta regressor.



Base regressors

1. **Extra Trees** (scikit-learn)
   Parameters: n_estimators=500, max_depth=26
2. **Gradient Boosting** (scikit-learn)
   Parameters: loss='lad', estimators=80, subsample=0.8, max_depth=13
3. **XGBoost** trained with the traditional gbtree booster.
   Parameters: n_estimators=60, subsample=0.7, max_depth=15, reg_lambda=6.5
4. **XGBoost** trained with the dart booster.
   Parameters: n_estimators=50, subsample=0.7, max_depth=17, reg_lambda=6
5. **LightGBM** trained with the dart booster.
   Parameters: n_estimators=350, subsample=0.1, num_leaves=550, reg_lambda=6
6. **Neural Network** (Tensorflow.Keras)
   Three hidden layers (400, 200, 100), dropout=0.5, activation='relu' and l2 regularization=0.0001 used for each hidden layer. The output layer uses linear activation for the regression. I used Adam as optimizer with learning rate=0.001, loss='msle' and batch size=32. I also used early stopping by setting aside a 10% validation set.

Meta regressor

A Neural Network similar to the base Neural Network. The only differences are that this one has only one hidden layer with 100 hidden units, learning rate=0.0001 and uses a batch size=8 for the training.

How it works

First, the base regressors are trained in a KFold cross validation manner to produce the meta-train set with their predictions. That means that the training set is split in 5 parts, and each time the base regressors are trained with 4 of them and predict the 5th one. The predictions of each base regressor is a feature column of the new meta-train set.

At this point it's important to specify that the base regressors are trained twice and predict the **'casual'** and **'registered'** values separately, which are then added together to produce the final predictions of the 'count' values. That's because I noticed that each of the base regressors had better results this way, and the stacking ensemble performed better too.

After that, the base regressors are trained with the whole training set and predict the test set. These predictions are used as the meta-test set.

Finally, the meta-regressor is trained with the meta-train set, and makes its predictions for the meta-test, which also are the final predictions.

Picking the best parameters and models

The parameters of the base regressors except the neural network are chosen by grid search cross validation. The parameters of the neural networks (both base and meta) are chosen by paying attention to train and validation learning curves. The target values in the training for choosing the best parameters were the 'count' values, even though in the stacking ensemble I predict the 'casual' and 'registered' values separately. That's because I noticed that the results were slightly better than picking different parameters for predicting the 'casual' and 'registered' values, so there was no need to add more complexity to the model.

As for picking the best models, I first trained a lot of them. Linear Regression, Stohastic Gradient Descent and SVM were dropped early as they were not good enough. The models that I considered adding to the ensemble are shown in the following table along with their public scores.

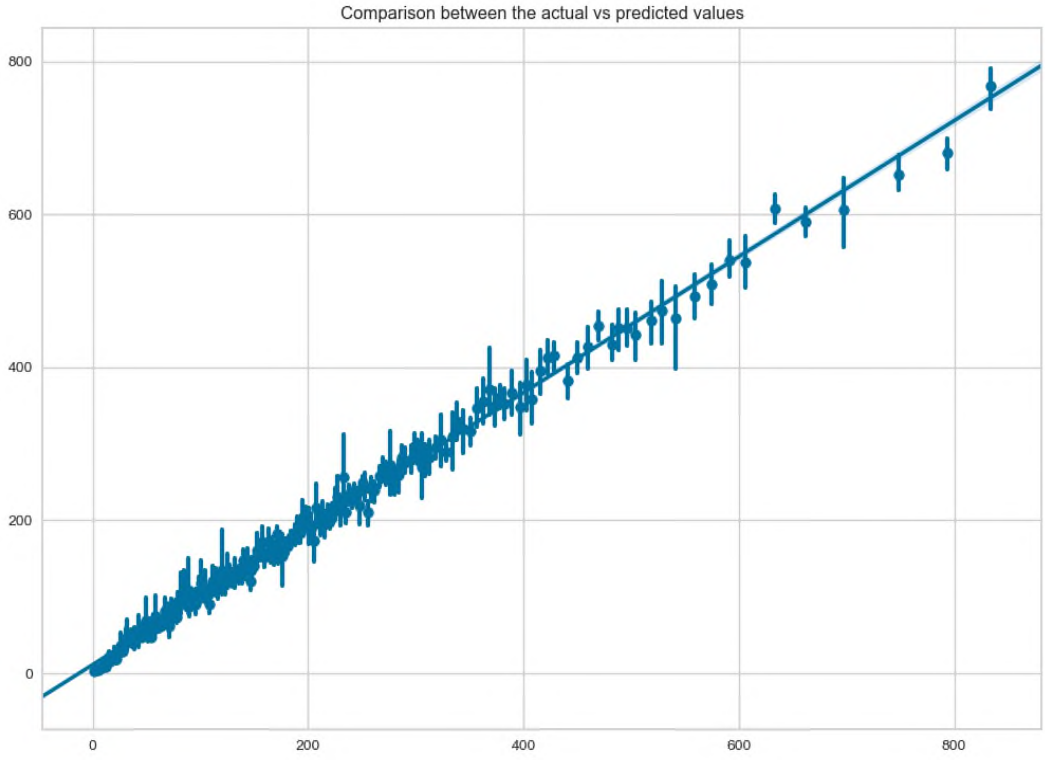
Model	Public Score
KNN	0.38255
Decision Tree	0.37330
Adaboost	0.32930
Random Forest	0.32397
Extra Trees	0.32275
Neural Network	0.31529
Gradient Boosting	0.31240
XGBoost (gbtree)	0.30162
LightGBM (dart)	0.29161
XGBoost (dart)	0.28957

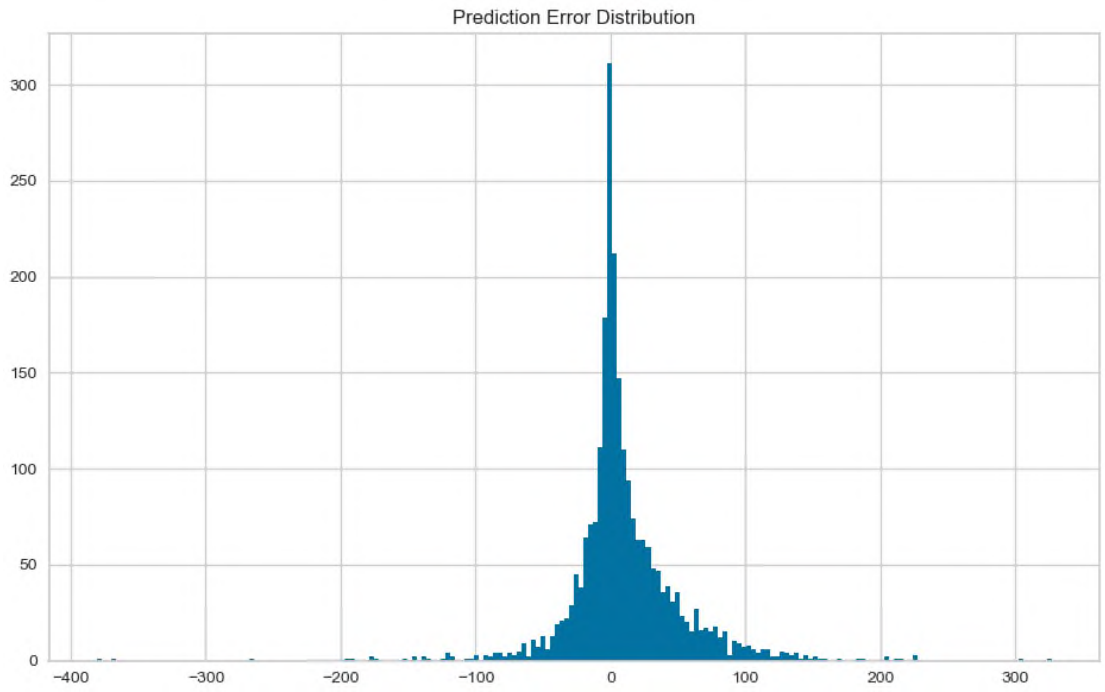
In order to decide which models work the best together I used cross validation with weights for a voting ensemble, by excluding each time one model until the best models were left. The neural network was added later to the stacking ensemble so it was not used for the cross validation, but the results when I added it were clearly better. It's interesting to point out, that even though Random Forest as an individual has much better results than KNN, it was excluded earlier during the cross validation. Different algorithms work better together and that's why when I added the neural network to the tree-based models it gave a nice boost to the predictions quality. As for the meta-regressor the neural network was by far the best estimator, although KNN performed pretty well too.

4. Performance

The **public score** of the final predictions is **0.27207**. There can be some changes in the 3rd decimal though due to the randomness of the neural networks.

The model needs about **20 minutes** to train (using CPU i7-8700). It's quite a lot mainly because of the base neural network model. Without it, it takes about 5 minutes to train, but I kept it anyways as it contributes to getting better predictions.





Above are the Regression Plot and the Error Distribution that I plotted after training the model by splitting the training data to train-test and evaluating the results. The test RMSLE score in this case was 0.2947. The deviations of the predicted values from the actual values are relatively low. From the error distribution we can see that most errors are around zero, but there are a few great ones probably caused by outliers.