Bike-22-02

February 22, 2019

1 ASIGNATURA: INVESTIGACIÓN EN INTELIGENCIA ARTIFICIAL

1.0.1 Laboratorio 2: Árboles y Random Forest para regresión

Autores:

Joaquín Delgado Fernández
Inés Heras Cáceres
Eloy Alfageme Galeano
Manuel Pasieka


```
In [1]: #Importamos todas las librerías.
        import pandas as pd
        from sklearn.preprocessing import MinMaxScaler, Normalizer,StandardScaler
        from sklearn.cluster import KMeans
        from sklearn.ensemble import RandomForestRegressor
        from sklearn.datasets import make_regression
        import matplotlib.pyplot as plt
        import math
        import time
        from collections import OrderedDict
        from sklearn.datasets import make_classification
        from sklearn.ensemble import RandomForestClassifier
        import numpy as np
        import sys
        import warnings
        from sklearn import preprocessing
        from sklearn.model_selection import train_test_split
        import seaborn as sns
        from sklearn.model_selection import GridSearchCV

        if not sys.warnoptions:
            warnings.simplefilter("ignore")
```

1.0.2 Initial Data-Set exploration

```
In [2]: filename = "Bike-Sharing-Dataset/hour.csv"
        hour = pd.read_csv(filename)
```

In [3]: hour.head()

Out[3]:

	instant	dteday	season	yr	mnth	hr	holiday	weekday	workingday	weathersit
0	1	2011-01-01	1	0	1	0	0	6	0	1
1	2	2011-01-01	1	0	1	1	0	6	0	1
2	3	2011-01-01	1	0	1	2	0	6	0	1
3	4	2011-01-01	1	0	1	3	0	6	0	1
4	5	2011-01-01	1	0	1	4	0	6	0	1

In [4]: hour.isnull().values.any()

Out[4]: False

In [5]: hour.describe().T

Out[5]:

	count	mean	std	min	25%	50%	75%
instant	17379.0	8690.000000	5017.029500	1.00	4345.5000	8690.0000	13034.5000
season	17379.0	2.501640	1.106918	1.00	2.0000	3.0000	3.0000
yr	17379.0	0.502561	0.500008	0.00	0.0000	1.0000	1.0000
mnth	17379.0	6.537775	3.438776	1.00	4.0000	7.0000	10.0000
hr	17379.0	11.546752	6.914405	0.00	6.0000	12.0000	18.0000
holiday	17379.0	0.028770	0.167165	0.00	0.0000	0.0000	0.0000
weekday	17379.0	3.003683	2.005771	0.00	1.0000	3.0000	5.0000
workingday	17379.0	0.682721	0.465431	0.00	0.0000	1.0000	1.0000
weathersit	17379.0	1.425283	0.639357	1.00	1.0000	1.0000	2.0000
temp	17379.0	0.496987	0.192556	0.02	0.3400	0.5000	0.6600
atemp	17379.0	0.475775	0.171850	0.00	0.3333	0.4848	0.6212
hum	17379.0	0.627229	0.192930	0.00	0.4800	0.6300	0.7800
windspeed	17379.0	0.190098	0.122340	0.00	0.1045	0.1940	0.2537
casual	17379.0	35.676218	49.305030	0.00	4.0000	17.0000	48.0000
registered	17379.0	153.786869	151.357286	0.00	34.0000	115.0000	220.0000
cnt	17379.0	189.463088	181.387599	1.00	40.0000	142.0000	281.0000

In [6]: hour.dtypes

Out[6]:

```
instant         int64
dteday         object
season          int64
yr              int64
mnth            int64
hr              int64
holiday         int64
weekday         int64
workingday      int64
weathersit      int64
temp          float64
atemp         float64
hum           float64
windspeed     float64
```

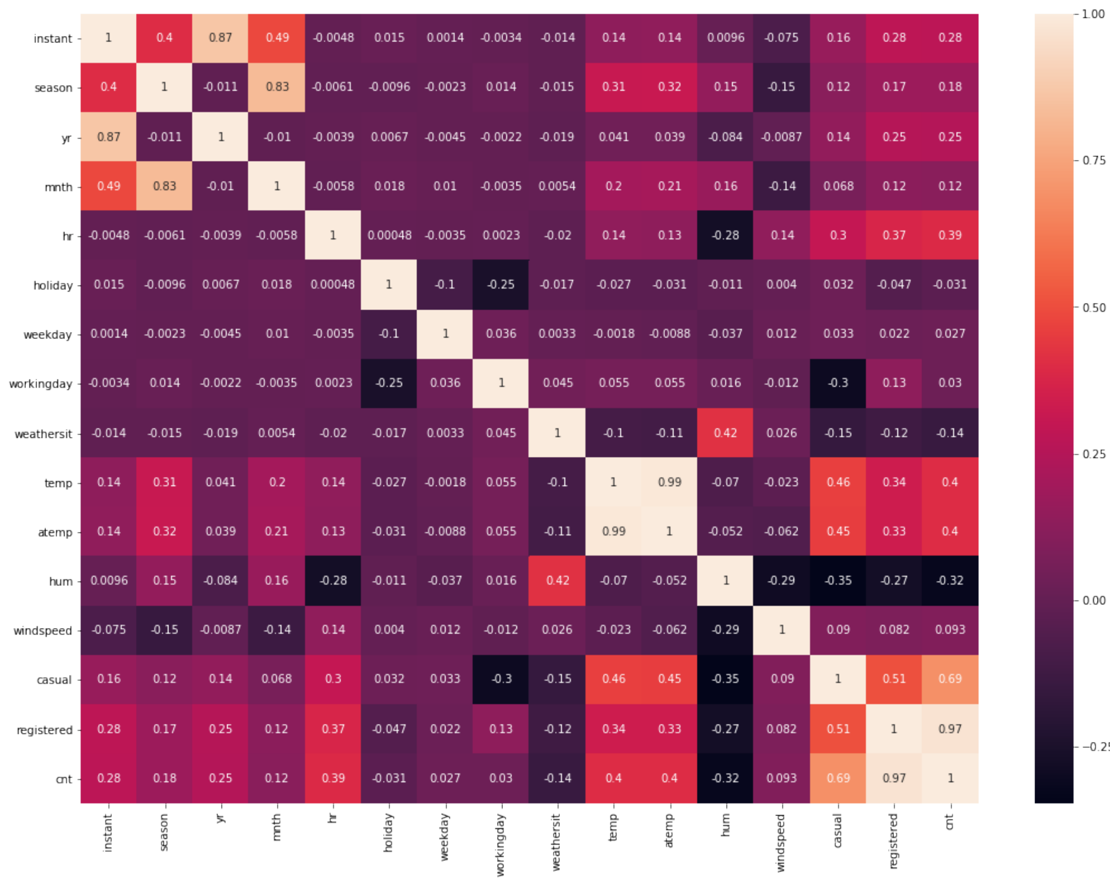
```
casual              int64
registered          int64
cnt                 int64
dtype: object
```

In [7]:
```
hour.head()
```

Out[7]:

	instant	dteday	season	yr	mnth	hr	holiday	weekday	workingday	weathersit
0	1	2011-01-01	1	0	1	0	0	6	0	1
1	2	2011-01-01	1	0	1	1	0	6	0	1
2	3	2011-01-01	1	0	1	2	0	6	0	1
3	4	2011-01-01	1	0	1	3	0	6	0	1
4	5	2011-01-01	1	0	1	4	0	6	0	1

In [8]:
```
plt.figure(figsize=(18,13))
sns.heatmap(hour.corr(), annot=True)
plt.show()
```

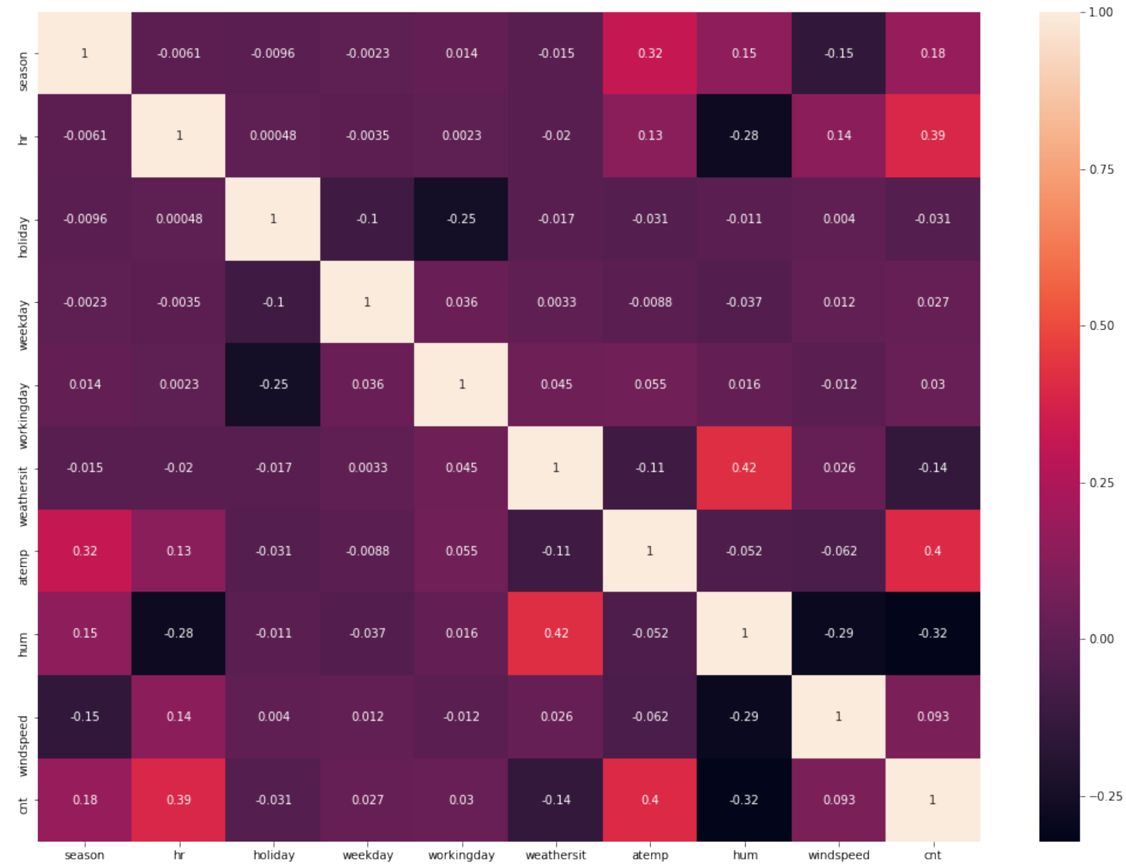


2 Eliminating Columns

Based on the correlation plot above and the type of somme of the columns, we a removing several columns.

```
In [9]: hour.drop(columns=['instant', 'dteday', 'yr', 'mnth', 'temp', 'casual', 'registered'],i
```

```
In [10]: plt.figure(figsize=(18,13))
         sns.heatmap(hour.corr(), annot=True)
         plt.show()
```



2.0.1 Treatment of Categorical Variables

In order for the decision trees to focus on specific values of categorical variables, we create one hot encoded dummy variables of several categorical variables.

```
In [11]: hour_dummied = pd.DataFrame.copy(hour)
         dummy_columns = ["season", "hr", "weekday", "weathersit"]
         hour_dummied = pd.concat([hour_dummied.drop(dummy_columns, axis=1), pd.get_dummies(da
```

```
In [12]: hour_dummied.head()
```

Out[12]:

	holiday	workingday	atemp	hum	windspeed	cnt	season_1	season_2	season_3	se
0	0	0	0.2879	0.81	0.0	16	1	0	0	
1	0	0	0.2727	0.80	0.0	40	1	0	0	
2	0	0	0.2727	0.80	0.0	32	1	0	0	
3	0	0	0.2879	0.75	0.0	13	1	0	0	
4	0	0	0.2879	0.75	0.0	1	1	0	0	

2.0.2 Randomize row order

In [13]:
```
#Randomize data
hour_rnd= hour_dummied.sample(frac=1)
```

In [14]:
```
hour_rnd.head()
```

Out[14]:

	holiday	workingday	atemp	hum	windspeed	cnt	season_1	season_2	season_3
8630	0	0	0.3939	0.76	0.0000	90	1	0	(
13850	0	0	0.7121	0.74	0.2985	68	0	0	1
4478	0	0	0.7576	0.46	0.2836	377	0	0	1
4080	0	1	0.6667	0.74	0.2239	162	0	0	1
594	0	1	0.1970	0.80	0.1642	16	1	0	(

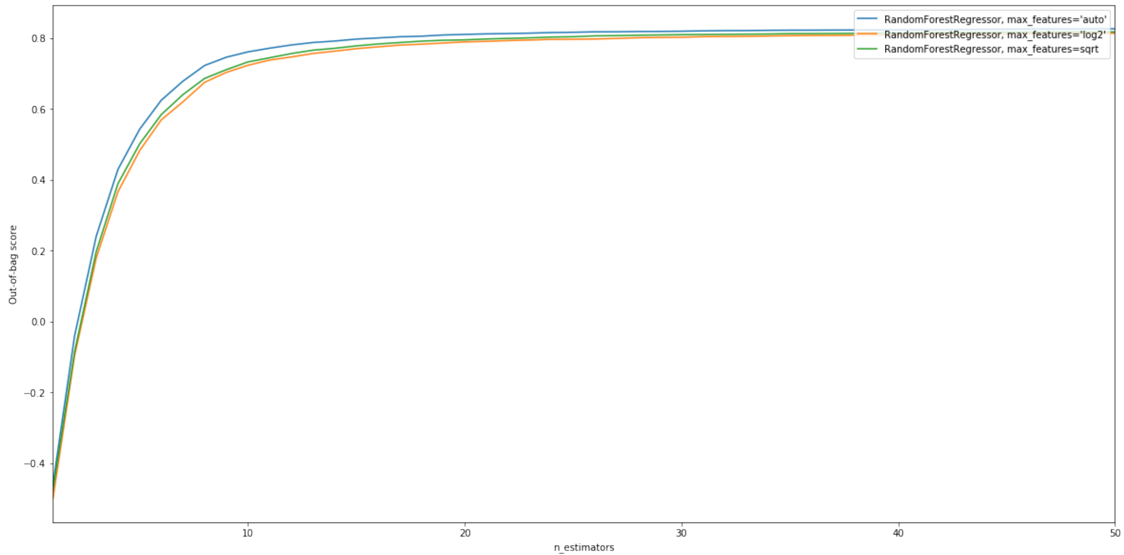
In [15]:
```
X = hour_rnd.drop(['cnt'], 1)
y = hour_rnd.cnt
```

3 Find the best model using the Out-of-bag Error - RandomForest

In [16]:
```
RANDOM_STATE = 123


# NOTE: Setting the `warm_start` construction parameter to `True` disables
# support for parallelized ensembles but is necessary for tracking the OOB
# error trajectory during training.
ensemble_clfs = [
        ("RandomForestRegressor, max_features='auto'",
        RandomForestRegressor(n_estimators=100,
                              max_features="auto",
                              oob_score= True,
                              random_state=RANDOM_STATE, n_jobs=-1)),
        ("RandomForestRegressor, max_features='log2'",
        RandomForestRegressor(n_estimators=100,
                              max_features='log2',
                              oob_score= True,
                              random_state=RANDOM_STATE, n_jobs=-1)),
        ("RandomForestRegressor, max_features=sqrt",
        RandomForestRegressor(n_estimators=100,
                             max_features='sqrt',
                             oob_score= True,
                           random_state=RANDOM_STATE, n_jobs=-1))
]
```

```python
    # Map a classifier name to a list of (<n_estimators>, <error rate>) pairs.
    score_rate = OrderedDict((label, []) for label, _ in ensemble_clfs)

    # Range of `n_estimators` values to explore.
    min_estimators = 1
    max_estimators = 50


    for label, clf in ensemble_clfs:
        print(label)
        for i in range(min_estimators, max_estimators + 1):
            clf.set_params(n_estimators=i)
            clf.fit(X, y)

            # Record the OOB error for each `n_estimators=i` setting.
            #score = clf.score(X,y)
            score = clf.oob_score_
            score_rate[label].append((i, score))

    plt.figure(figsize=(20,10))
    # Generate the "OOB error rate" vs. "n_estimators" plot.
    for label, clf_err in score_rate.items():
        xs, ys = zip(*clf_err)
        plt.plot(xs, ys, label=label)

    plt.xlim(min_estimators, max_estimators)
    plt.xlabel("n_estimators")
    plt.ylabel("Out-of-bag score")
    plt.legend(loc="upper right")
    plt.show()
```

```
RandomForestRegressor, max_features='auto'
RandomForestRegressor, max_features='log2'
RandomForestRegressor, max_features=sqrt
```



3.1 OOB Results

The RandomForestRegressor with max_features='auto' and 20 Estimators seems to be a good choice as a model, achieving a OOB score of almost 0.8.

4 Model Feature Analysis

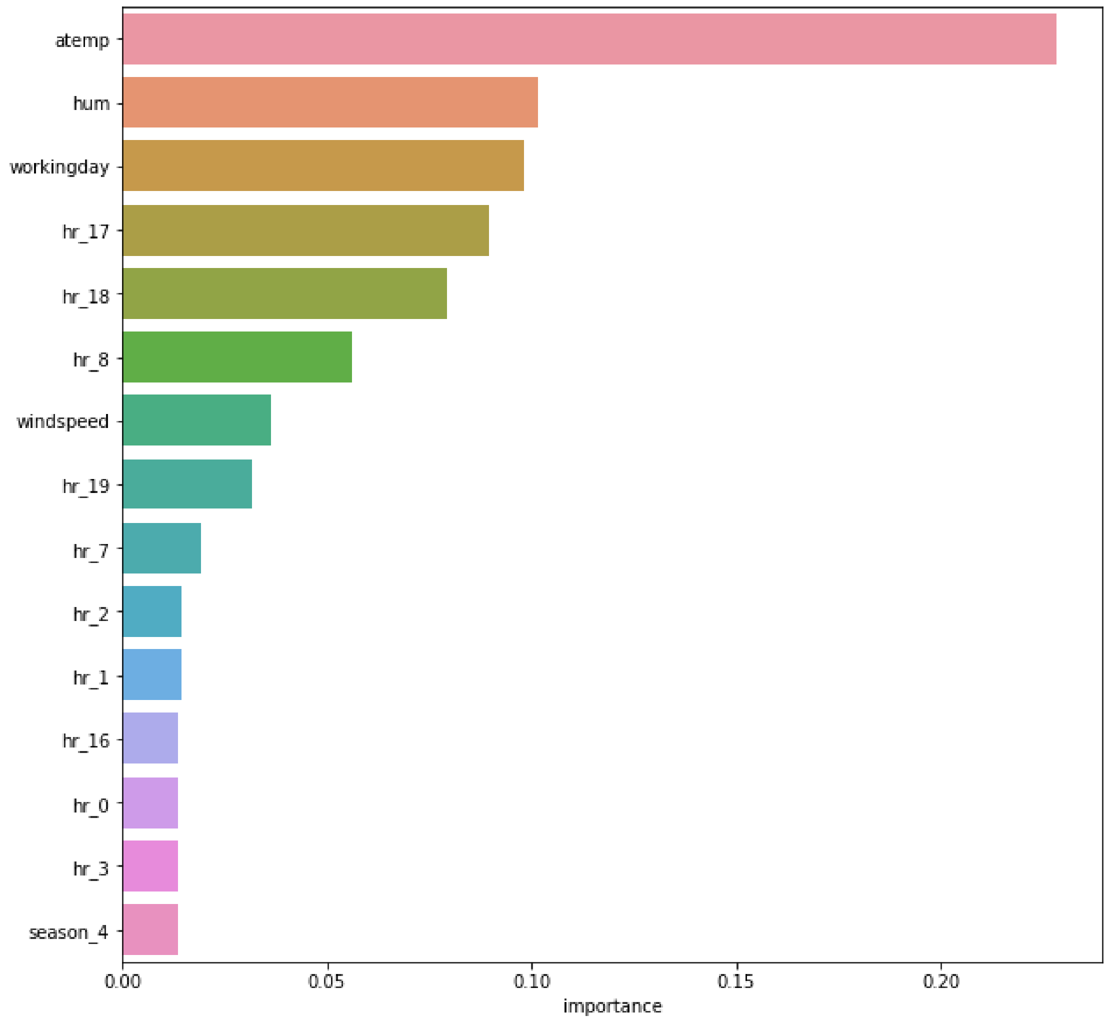
Next study the importance of the different features and try to explain the model

```
In [17]: clf = ensemble_clfs[0][1]

         # Look at the mean feature importance
         feature_importances = pd.DataFrame(clf.feature_importances_, index = X.columns, colum
         feature_importances.sort_values('importance', ascending=False, inplace=True)
         feature_importances.head(10)
```

Out[17]:

	importance
atemp	0.228127
hum	0.101830
workingday	0.098168
hr_17	0.089796
hr_18	0.079291
hr_8	0.056171
windspeed	0.036367
hr_19	0.031756
hr_7	0.019289
hr_2	0.014593

In [18]:
```
# Only look at the most important once
fig, ax = plt.subplots(figsize=(10, 10))
most_important_features = feature_importances[0:15]
sns.barplot(y=most_important_features.index, x=most_important_features['importance'],
plt.show()
```



5 Feature Analysis

The most important features are temperature, humidity, workingday and the hour.

Looking at the dataset per hand and the specific features it seems that

- Temperature: Bikes are rented most in mid temperatures (aterm > 0.2 and < 0.8, corresponding to above 10 and below 40 degrees)
- Humidity: People like to bike most if mudity is between 30% and 90%
- Workingday: People prefare bikes on woring days
- Popular hours for renting bikes are 8am, 5-6pm

5.1 Adaboost - Boosting method

AdaBoost, short for "Adaptive Boosting", is the first practical boosting algorithm proposed by Freund and Schapire in 1996. It focuses on classification problems and aims to convert a set of weak classifiers into a strong one.

The output of the other learning algorithms ('weak learners') is combined into a weighted sum that represents the final output of the boosted classifier.

The model achieves a R^2 score of 0.38, and is therefore faar less than the R^2 score of 0.8 achieved by the RF.

For this dataset the RandomForest clearly outperforms Adaboost.

```
In [26]: from sklearn.tree import DecisionTreeRegressor
         from sklearn.ensemble import AdaBoostRegressor
```

```
In [20]: Adaregressor = AdaBoostRegressor()
         parameters = [{'n_estimators' : [150,200,250,300], 'loss' : ['linear','square','expone
         grid_search = GridSearchCV(estimator = Adaregressor, param_grid = parameters)
         grid_search = grid_search.fit(X, y)
         best_parameters = grid_search.best_params_
```

```
In [21]: best_parameters
```

```
Out[21]: {'base_estimator': DecisionTreeRegressor(criterion='mse', max_depth=5, max_features=No
                    max_leaf_nodes=None, min_impurity_decrease=0.0,
                    min_impurity_split=None, min_samples_leaf=1,
                    min_samples_split=2, min_weight_fraction_leaf=0.0,
                    presort=False, random_state=None, splitter='best'),
          'loss': 'linear',
          'n_estimators': 300}
```

```
In [22]: print(f'Achieves a R^2 score of {grid_search.best_score_}!')
```

```
Achieves a R^2 score of 0.4012393871974058!
```

6 Training a classifier

Generat a group feature that is seperating between low and high usage hours, and train a classifier for prediction.

```
In [23]: # Define a new group variable
         def add_group(x):
             if x <= 20:
                 return 1
             else:
                 return 2
         hour_rnd['group'] = hour_rnd['cnt'].apply(add_group)
         hour_rnd.head()
```

Out[23]:

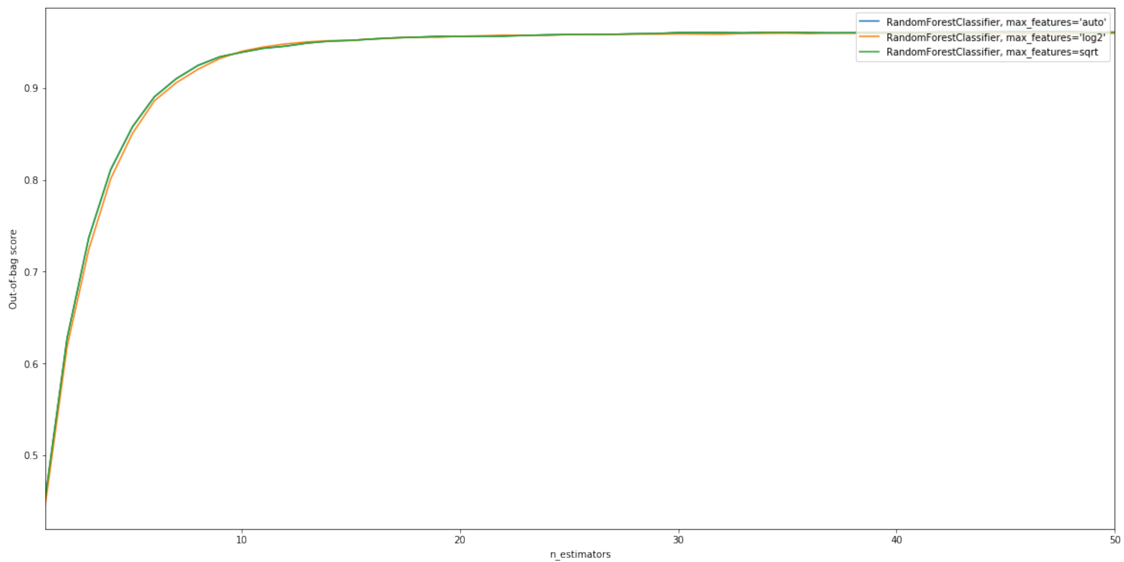
	holiday	workingday	atemp	hum	windspeed	cnt	season_1	season_2	season_3
8630	0	0	0.3939	0.76	0.0000	90	1	0	0
13850	0	0	0.7121	0.74	0.2985	68	0	0	1
4478	0	0	0.7576	0.46	0.2836	377	0	0	1
4080	0	1	0.6667	0.74	0.2239	162	0	0	1
594	0	1	0.1970	0.80	0.1642	16	1	0	0

In [24]:

```python
# Run again the OOB analysis using the group variable as target
RANDOM_STATE = 12
X = hour_rnd.drop(['group', 'cnt'], 1)
y = hour_rnd.group


ensemble_clfs = [
        ("RandomForestClassifier, max_features='auto'",
        RandomForestClassifier(n_estimators=100,
                               max_features="auto",
                               oob_score= True,
                               random_state=RANDOM_STATE, n_jobs=-1)),
        ("RandomForestClassifier, max_features='log2'",
        RandomForestClassifier(n_estimators=100,
                               max_features='log2',
                               oob_score= True,
                               random_state=RANDOM_STATE, n_jobs=-1)),
        ("RandomForestClassifier, max_features=sqrt",
        RandomForestClassifier(n_estimators=100,
                              max_features='sqrt',
                              oob_score= True,
                            random_state=RANDOM_STATE, n_jobs=-1))
]

# Map a classifier name to a list of (<n_estimators>, <error rate>) pairs.
score_rate = OrderedDict((label, []) for label, _ in ensemble_clfs)

# Range of `n_estimators` values to explore.
min_estimators = 1
max_estimators = 50


for label, clf in ensemble_clfs:
    print(label)
    for i in range(min_estimators, max_estimators + 1):
        clf.set_params(n_estimators=i)
        clf.fit(X, y)

        # Record the OOB error for each `n_estimators=i` setting.
        #score = clf.score(X,y)
        score = clf.oob_score_
```

```
            score_rate[label].append((i, score))

    plt.figure(figsize=(20,10))
    # Generate the "OOB error rate" vs. "n_estimators" plot.
    for label, clf_err in score_rate.items():
        xs, ys = zip(*clf_err)
        plt.plot(xs, ys, label=label)

    plt.xlim(min_estimators, max_estimators)
    plt.xlabel("n_estimators")
    plt.ylabel("Out-of-bag score")
    plt.legend(loc="upper right")
    plt.show()
```

```
RandomForestClassifier, max_features='auto'
RandomForestClassifier, max_features='log2'
RandomForestClassifier, max_features=sqrt
```



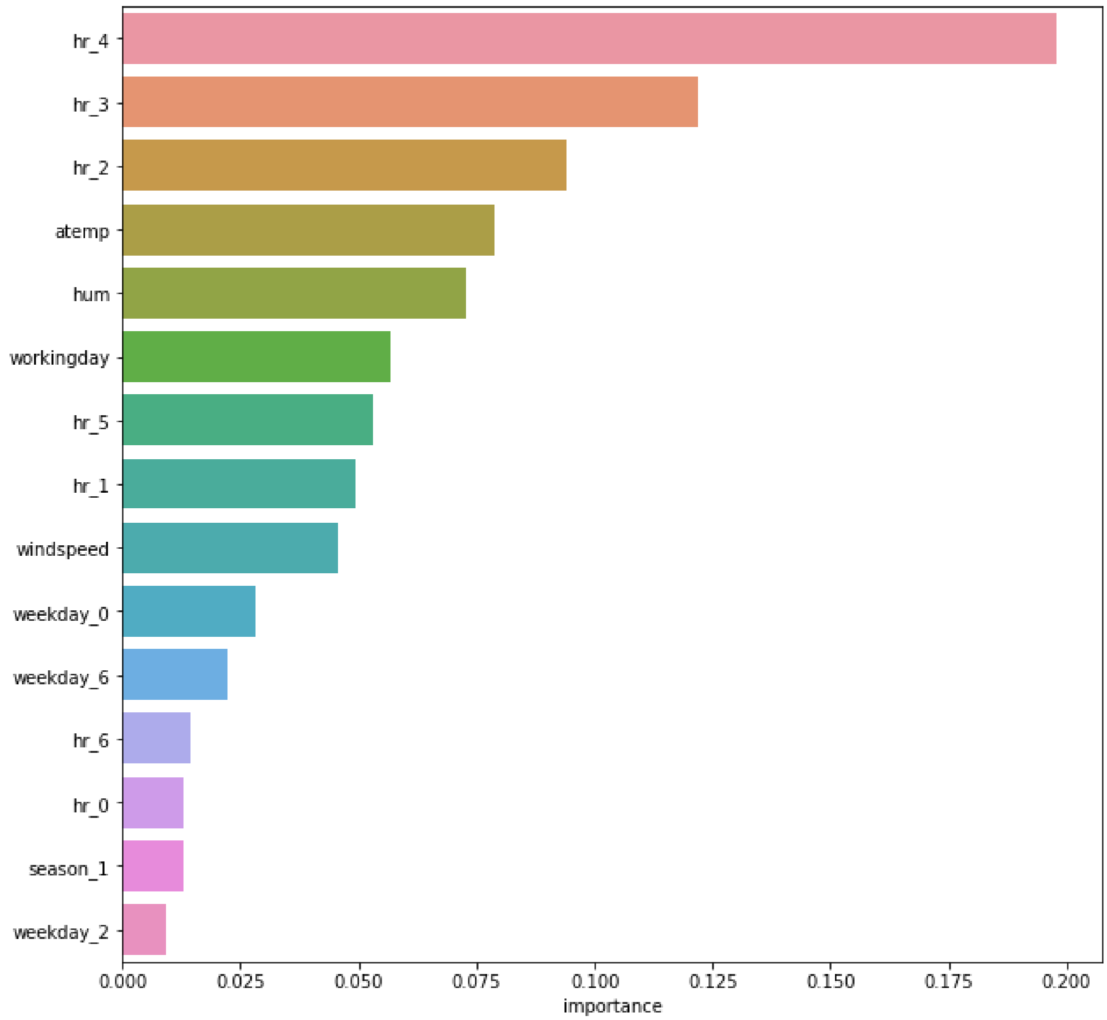

```
In [38]: print('Mean accuracy %f' % score_rate["RandomForestClassifier, max_features='auto'"][
```

```
Mean accuracy 0.956154
```

6.1 OOB Results

The RandomForestRegressor with max_features='auto' and 20 or 30 Estimators seems to be a good choice as a model.

In [25]:
```python
clf = ensemble_clfs[0][1]

# Look at the mean feature importance
feature_importances = pd.DataFrame(clf.feature_importances_, index = X.columns, colum
feature_importances.sort_values('importance', ascending=False, inplace=True)
# Only look at the most important once
fig, ax = plt.subplots(figsize=(10, 10))
most_important_features = feature_importances[0:15]
sns.barplot(y=most_important_features.index, x=most_important_features['importance'],
plt.show()
```



7 Explaining the features

The low usage group (i.e. group ==1) are those hours of the day with a very low bike usage, and in the feature importance this is reflected, making the hours from 2-4am the once with the lowest usage and most predictive capacity.