Lambda School Data Science — Tree Ensembles

Random Forests

Pre-read

- Scikit-Learn User Guide, Ensemble Methods (https://scikit-learn.org/stable/modules/ensemble.html)
- Coloring with Random Forests (http://structuringtheunstructured.blogspot.com/2017/11/coloring-with-random-forests.html)
- Beware Default Random Forest Importances (https://explained.ai/rf-importance/index.html)

More

- Machine Learning Explainability: Permutation Importance (https://www.kaggle.com/dansbecker/permutation-importance)
- eli5: Permutation Importance (https://eli5.readthedocs.io/en/latest/blackbox/permutation_importance.html)
- eli5: Explaining XGBoost predictions on the Titanic dataset (https://eli5.readthedocs.io/en/latest/_notebooks/xgboost-titanic.html)
- The Mechanics of Machine Learning: Categorically Speaking (https://mlbook.explained.ai/catvars.html)

Selecting good features – Part III: random forests (https://blog.datadive.net/selecting-good-features-part-iii-random-forests/)

> There are a few things to keep in mind when using the impurity based ranking. Firstly, feature selection based on impurity reduction is biased towards preferring variables with more categories.
>
> Secondly, when the dataset has two (or more) correlated features, then from the point of view of the model, any of these correlated features can be used as the predictor, with no concrete preference of one over the others. But once one of them is used, the importance of others is significantly reduced since effectively the impurity they can remove is already removed by the first feature. As a consequence, they will have a lower reported importance. This is not an issue when we want to use feature selection to reduce overfitting, since it makes sense to remove features that are mostly duplicated by other features. But when interpreting the data, it can lead to the incorrect conclusion that one of the variables is a strong predictor while the others in the same group are unimportant, while actually they are very close in terms of their relationship with the response variable.

[An Introduction to Statistical Learning (http://www-bcf.usc.edu/~gareth/ISL/)](http://www-bcf.usc.edu/~gareth/ISL/), Chapter 8.2.1, Out-of-Bag Error Estimation

> It turns out that **there is a very straightforward way to estimate the test error of a bagged model, without the need to perform cross-validation or the validation set approach.**
>
> Recall that the key to bagging is that trees are repeatedly fit to bootstrapped subsets of the observations. One can show that on average, each bagged tree makes use of around two-thirds of the observations. The remaining one-third of the **observations not used to fit a given bagged tree are referred to as the out-of bag (OOB) observations.**
>
> We can predict the response for the ith observation using each of the trees in which that observation was OOB. This will yield around B/3 predictions for the ith observation. In order to obtain a single prediction for the ith observation, we can average these predicted responses (if regression is the goal) or can take a majority vote (if classification is the goal).
>
> This leads to a single OOB prediction for the ith observation. An OOB prediction can be obtained in this way for each of the n observations, from which the overall OOB MSE (for a regression problem) or classification error (for a classification problem) can be computed. The resulting **OOB error is a valid estimate of the test error for the bagged model, since the response for each observation is predicted using only the trees that were not fit using that observation.** ...
>
> It can be shown that with B sufficiently large, OOB error is virtually equivalent to leave-one-out cross-validation error. The OOB approach for estimating the test error is particularly **convenient when performing bagging on large data sets for which cross-validation would be computationally onerous.**

Libraries

- [eli5 (https://github.com/TeamHG-Memex/eli5)](https://github.com/TeamHG-Memex/eli5): `conda install -c conda-forge eli5 / pip install eli5`
- [category_encoders (https://github.com/scikit-learn-contrib/categorical-encoding)](https://github.com/scikit-learn-contrib/categorical-encoding): `conda install -c conda-forge category_encoders / pip install category_encoders`
- [mlxtend (https://github.com/rasbt/mlxtend)](https://github.com/rasbt/mlxtend): `pip install mlxtend`
- [ipywidgets (https://ipywidgets.readthedocs.io/en/stable/examples/Using%20Interact.html)](https://ipywidgets.readthedocs.io/en/stable/examples/Using%20Interact.html): included with Anaconda, doesn't work on Google Colab

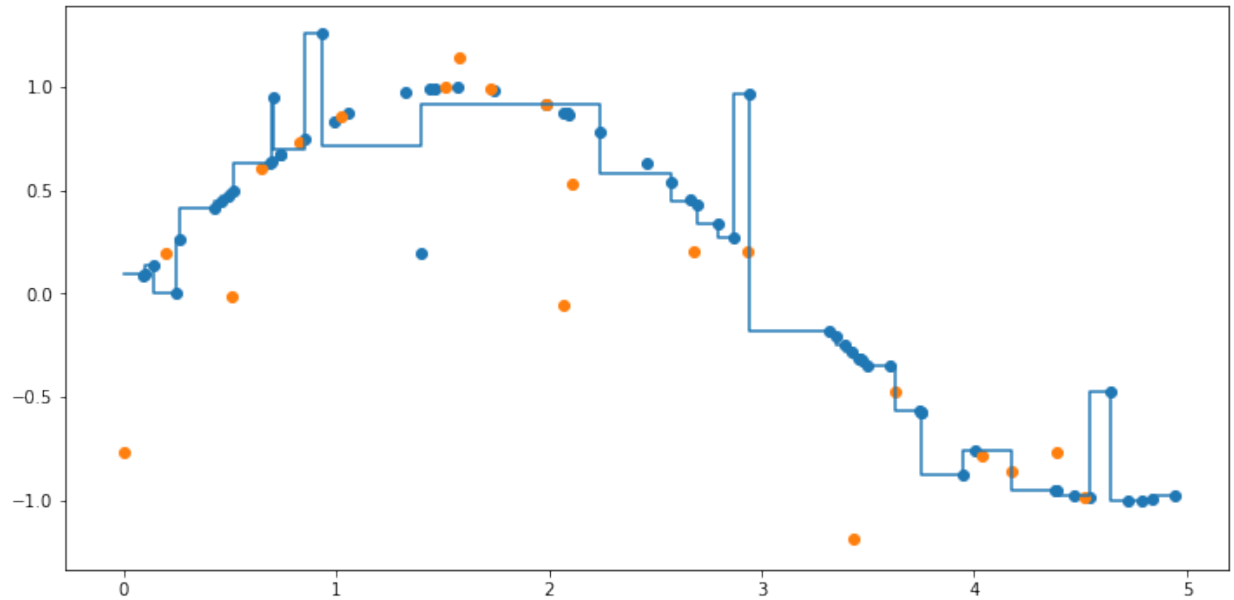
ipywidgets revisited: Decision Tree vs Random Forest

In [3]:
```
%matplotlib inline
from ipywidgets import interact
import matplotlib.pyplot as plt
from sklearn.ensemble import RandomForestRegressor
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeRegressor

# Example from http://scikit-learn.org/stable/auto_examples/tree/plot_tr
def make_data():
    import numpy as np
    rng = np.random.RandomState(1)
    X = np.sort(5 * rng.rand(80, 1), axis=0)
    y = np.sin(X).ravel()
    y[::5] += 2 * (0.5 - rng.rand(16))
    return X, y

X, y = make_data()

X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.25, random_state=42)

def regress_wave(max_depth):
    dt = DecisionTreeRegressor(max_depth=max_depth)
    dt.fit(X_train, y_train)
    print('Decision Tree train R^2:', dt.score(X_train, y_train))
    print('Decision Tree test R^2:', dt.score(X_test, y_test))
    plt.gcf().set_size_inches(12, 6)
    plt.scatter(X_train, y_train)
    plt.scatter(X_test, y_test)
    plt.step(X, dt.predict(X))
    plt.show()

    rf = RandomForestRegressor(max_depth=max_depth, n_estimators=100, n_
    rf.fit(X_train, y_train)
    print('Random Forest train R^2:', rf.score(X_train, y_train))
    print('Random Forest test R^2:', rf.score(X_test, y_test))
    plt.gcf().set_size_inches(12, 6)
    plt.scatter(X_train, y_train)
    plt.scatter(X_test, y_test)
    plt.step(X, rf.predict(X))
    plt.show()

interact(regress_wave, max_depth=(1,8,1));
```

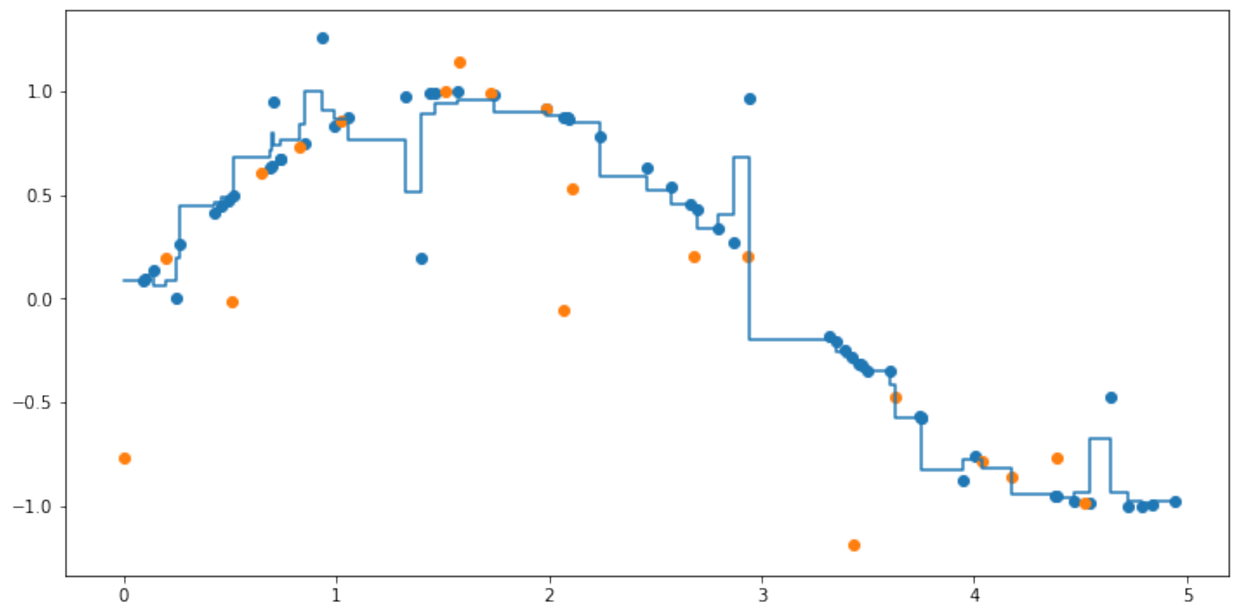
max_depth 6

max_depth

```
Decision Tree train R^2: 0.9846072146401021
Decision Tree test R^2: 0.6675139268793822
```




```
Random Forest train R^2: 0.9833229991881686
Random Forest test R^2: 0.7150540416259283
```

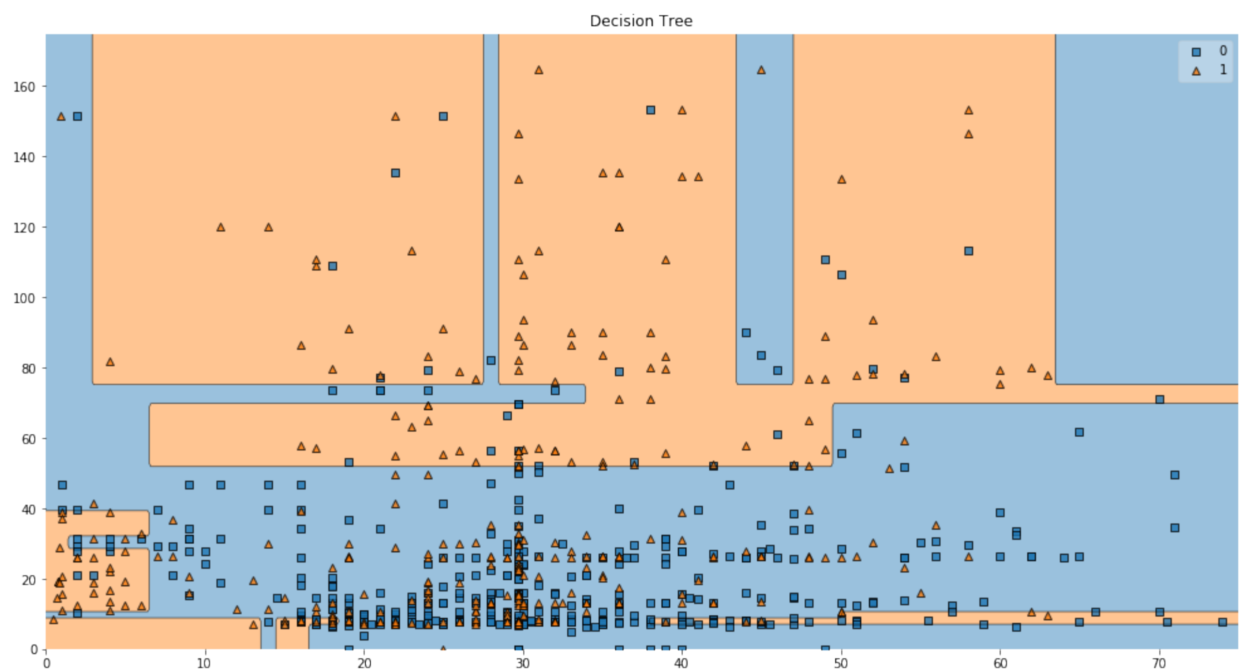


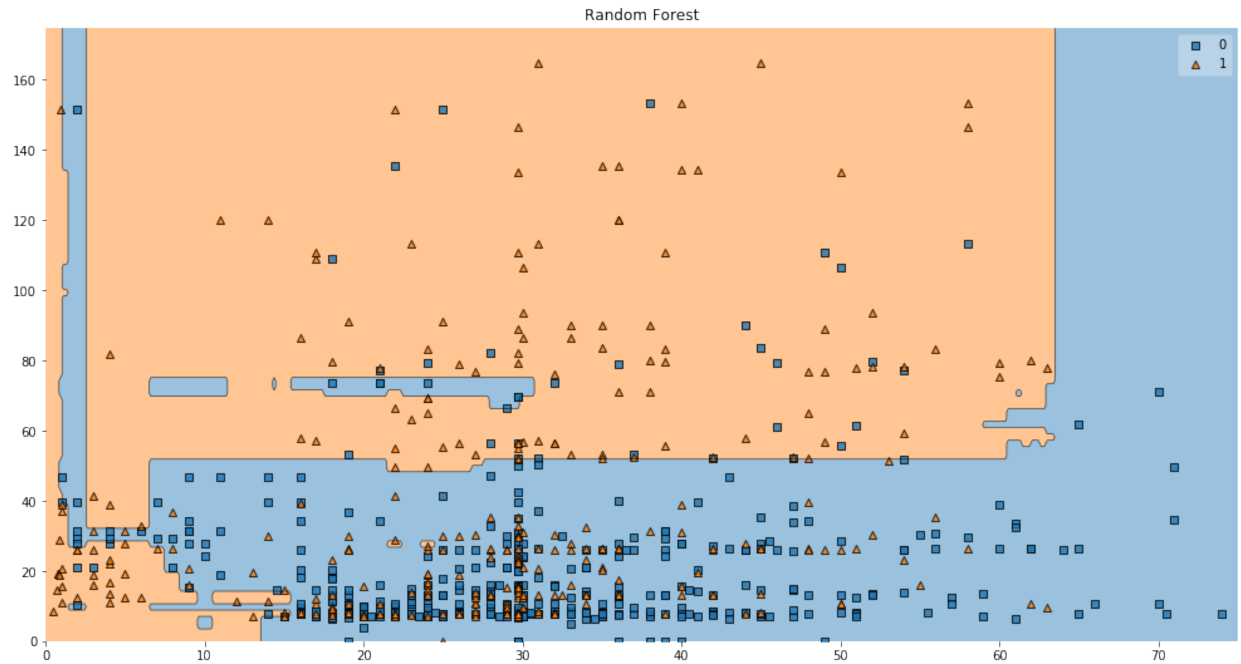
Regressing a wave

Titanic survival, by Age & Fare

In [2]:
```
from mlxtend.plotting import plot_decision_regions
import seaborn as sns
from sklearn.ensemble import RandomForestClassifier
from sklearn.impute import SimpleImputer
from sklearn.tree import DecisionTreeClassifier

titanic = sns.load_dataset('titanic')
X = SimpleImputer().fit_transform(titanic[['age', 'fare']])
y = titanic['survived'].values

def classify_titanic(max_depth):
    dt = DecisionTreeClassifier(max_depth=max_depth)
    dt.fit(X, y)
    plot_decision_regions(X, y, dt)
    plt.gcf().set_size_inches(17, 9)
    plt.title('Decision Tree')
    plt.axis((0,75,0,175))
    plt.show()

    rf = RandomForestClassifier(max_depth=max_depth, n_estimators=100, n_
    rf.fit(X, y)
    plot_decision_regions(X, y, rf)
    plt.gcf().set_size_inches(17, 9)
    plt.title('Random Forest')
    plt.axis((0,75,0,175))
    plt.show()

interact(classify_titanic, max_depth=(1,8,1));
```

max_depth 6





Lending Club

Read csv files downloaded from [Kaggle (https://www.kaggle.com/c/ds2-tree-ensembles/data)](https://www.kaggle.com/c/ds2-tree-ensembles/data)

In [34]:
```
%%time
import pandas as pd
pd.options.display.max_columns = 200
pd.options.display.max_rows = 200

X_train = pd.read_csv('/Users/wel51x/Downloads/ds2-tree-ensembles/train_
X_test = pd.read_csv('/Users/wel51x/Downloads/ds2-tree-ensembles/test_fe
y_train = pd.read_csv('/Users/wel51x/Downloads/ds2-tree-ensembles/train_
sample_submission = pd.read_csv('/Users/wel51x/Downloads/ds2-tree-ensemb

X_train.shape, X_test.shape, y_train.shape, y_test.shape
```

```
CPU times: user 26.5 s, sys: 4.92 s, total: 31.4 s
Wall time: 32.9 s
```

Wrangle X_train and X_test in the same way

In [35]:
```
def wrangle(X):
    X = X.copy()

    # Drop some columns
    X = X.drop(columns='id')  # id is random
    X = X.drop(columns=['member_id', 'url', 'desc'])  # All null
```

```python
    X = X.drop(columns='title')  # Duplicative of purpose
    X = X.drop(columns='grade')  # Duplicative of sub_grade

    # Transform sub_grade from "A1" - "G5" to 1.1 - 7.5
    def wrangle_sub_grade(x):
        first_digit = ord(x[0]) - 64
        second_digit = int(x[1])
        return first_digit + second_digit/10

    X['sub_grade'] = X['sub_grade'].apply(wrangle_sub_grade)

    # Convert percentages from strings to floats
    X['int_rate'] = X['int_rate'].str.strip('%').astype(float)
    X['revol_util'] = X['revol_util'].str.strip('%').astype(float)

    # Transform earliest_cr_line to an integer: how many days it's been
    X['earliest_cr_line'] = pd.to_datetime(X['earliest_cr_line'], infer_
    X['earliest_cr_line'] = pd.Timestamp.today() - X['earliest_cr_line']
    X['earliest_cr_line'] = X['earliest_cr_line'].dt.days

    # Create features for three employee titles: teacher, manager, owner
    X['emp_title'] = X['emp_title'].str.lower()
    X['emp_title_teacher'] = X['emp_title'].str.contains('teacher', na=F
    X['emp_title_manager'] = X['emp_title'].str.contains('manager', na=F
    X['emp_title_owner']   = X['emp_title'].str.contains('owner', na=Fal

    # Drop categoricals with high cardinality
    X = X.drop(columns=['emp_title', 'zip_code'])

    # Transform features with many nulls to binary flags
    many_nulls = ['sec_app_mths_since_last_major_derog',
                  'sec_app_revol_util',
                  'sec_app_earliest_cr_line',
                  'sec_app_mort_acc',
                  'dti_joint',
                  'sec_app_collections_12_mths_ex_med',
                  'sec_app_chargeoff_within_12_mths',
                  'sec_app_num_rev_accts',
                  'sec_app_open_act_il',
                  'sec_app_open_acc',
                  'revol_bal_joint',
                  'annual_inc_joint',
                  'sec_app_inq_last_6mths',
                  'mths_since_last_record',
                  'mths_since_recent_bc_dlq',
                  'mths_since_last_major_derog',
                  'mths_since_recent_revol_delinq',
                  'mths_since_last_delinq',
                  'il_util',
                  'emp_length',
```

```
                       'mths_since_recent_inq',
                       'mo_sin_old_il_acct',
                       'mths_since_rcnt_il',
                       'num_tl_120dpd_2m',
                       'bc_util',
                       'percent_bc_gt_75',
                       'bc_open_to_buy',
                       'mths_since_recent_bc']

    for col in many_nulls:
        X[col] = X[col].isnull()

    # For features with few nulls, do mean imputation
    for col in X:
        if X[col].isnull().sum() > 0:
            X[col] = X[col].fillna(X[col].mean())

    # Return the wrangled dataframe
    return X


X_train = wrangle(X_train)
X_test  = wrangle(X_test)
X_train.shape, X_test.shape
```

Out[35]: ((1309457, 98), (26724, 98))

Now X_train (and X_test) have no nulls

In [36]:
```
null_counts = X_train.isnull().sum()
all(null_counts == 0)
```

Out[36]: True

And no high cardinality categoricals

In [37]:
```
cardinality = X_train.select_dtypes(exclude='number').nunique()
all(cardinality <= 50)
```

Out[37]: False

In [38]:
```
cardinality
```

Out[38]:
```
term                                  2
emp_length                            2
home_ownership                        6
purpose                              14
addr_state                           51
mths_since_last_delinq                2
mths_since_last_record                2
initial_list_status                   2
mths_since_last_major_derog           2
application_type                      2
annual_inc_joint                      2
dti_joint                             2
mths_since_rcnt_il                    2
il_util                               2
bc_open_to_buy                        2
bc_util                               2
mo_sin_old_il_acct                    2
mths_since_recent_bc                  2
mths_since_recent_bc_dlq              2
mths_since_recent_inq                 2
mths_since_recent_revol_delinq        2
num_tl_120dpd_2m                      2
percent_bc_gt_75                      2
revol_bal_joint                       2
sec_app_earliest_cr_line              2
sec_app_inq_last_6mths                2
sec_app_mort_acc                      2
sec_app_open_acc                      2
sec_app_revol_util                    2
sec_app_open_act_il                   2
sec_app_num_rev_accts                 2
sec_app_chargeoff_within_12_mths      2
sec_app_collections_12_mths_ex_med    2
sec_app_mths_since_last_major_derog   2
disbursement_method                   2
emp_title_teacher                     2
emp_title_manager                     2
emp_title_owner                       2
dtype: int64
```

Decision Tree

In [39]:
```
%%time
import category_encoders as ce
from sklearn.model_selection import cross_val_score
from sklearn.pipeline import make_pipeline
from sklearn.tree import DecisionTreeClassifier

pipe = make_pipeline(
    ce.OrdinalEncoder(),
    DecisionTreeClassifier(max_depth=5, class_weight='balanced')
)

cross_val_score(pipe, X_train, y_train, cv=5, scoring='roc_auc')
```

```
CPU times: user 4min 44s, sys: 44.5 s, total: 5min 28s
Wall time: 5min 58s
```

In [40]:
```
%%time
from sklearn.ensemble import RandomForestClassifier

pipe = make_pipeline(
    ce.OrdinalEncoder(),
    RandomForestClassifier(
        n_estimators=100,
        class_weight='balanced',
        min_samples_leaf=0.005,
        oob_score=True,
        n_jobs=-1)
)

cross_val_score(pipe, X_train, y_train, cv=5, scoring='roc_auc', verbose
```

```
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurren
t workers.

[CV]  ................................................................
[CV] ........................ , score=0.714071853559479, total= 4.8min
[CV]  ................................................................

[Parallel(n_jobs=1)]: Done   1 out of   1 | elapsed:  4.8min remaining
:    0.0s

[CV] ........................ , score=0.712632525364129, total= 4.8min
[CV]  ................................................................

[Parallel(n_jobs=1)]: Done   2 out of   2 | elapsed:  9.6min remaining
:    0.0s

[CV] ....................... , score=0.7128238788575855, total= 4.4min
```

```
[CV]  ...............................................................

[Parallel(n_jobs=1)]: Done   3 out of   3 | elapsed: 14.0min remaining
:    0.0s

[CV] ...................... , score=0.7149196967120637, total= 4.4min
[CV]  ...............................................................

[Parallel(n_jobs=1)]: Done   4 out of   4 | elapsed: 18.4min remaining
:    0.0s

[CV] ...................... , score=0.7147479463043442, total= 3.7min
CPU times: user 3min 24s, sys: 1min 29s, total: 4min 53s
Wall time: 22min 7s

[Parallel(n_jobs=1)]: Done   5 out of   5 | elapsed: 22.1min remaining
:    0.0s
[Parallel(n_jobs=1)]: Done   5 out of   5 | elapsed: 22.1min finished
```

Out-of-Bag estimated score

Out-of-bag is a faster way to get an estimated score with Random Forest, using the parameter `oob_score=True`

Random Forest

Improves ROC AUC compared to Decision Tree

In [41]:
```
from sklearn.metrics import roc_auc_score
```

In [42]:
```
%%time
pipe.fit(X_train, y_train)
y_pred_proba = pipe.named_steps['randomforestclassifier'].oob_decision_f
print('ROC AUC, Out-of-Bag estimate:', roc_auc_score(y_train, y_pred_pro
```

```
ROC AUC, Out-of-Bag estimate: 0.71302387007396
CPU times: user 13min 15s, sys: 22.9 s, total: 13min 38s
Wall time: 4min 31s
```

In [43]:
```
pipe.named_steps
```

Out[43]:
```
{'ordinalencoder': OrdinalEncoder(cols=['term', 'home_ownership', 'pur
pose', 'addr_state', 'initial_list_status', 'application_type', 'disbu
rsement_method'],
         drop_invariant=False, handle_unknown='impute', impute_missing
=True,
         mapping=[{'col': 'term', 'mapping': [(' 36 months', 1), (' 60
months', 2)]}, {'col': 'home_ownership', 'mapping': [('MORTGAGE', 1),
('RENT', 2), ('OWN', 3), ('ANY', 4), ('OTHER', 5), ('NONE', 6)]}, {'co
l': 'purpose', 'mapping': [('home_improvement', 1), ('debt_consolidati
on', 2), ('major_purchase', ... 1), ('Joint App', 2)]}, {'col': 'disbu
rsement_method', 'mapping': [('Cash', 1), ('DirectPay', 2)]}],
         return_df=True, verbose=0),
 'randomforestclassifier': RandomForestClassifier(bootstrap=True, clas
s_weight='balanced',
             criterion='gini', max_depth=None, max_features='auto',
             max_leaf_nodes=None, min_impurity_decrease=0.0,
             min_impurity_split=None, min_samples_leaf=0.005,
             min_samples_split=2, min_weight_fraction_leaf=0.0,
             n_estimators=100, n_jobs=-1, oob_score=True, random_state
=None,
             verbose=0, warm_start=False)}
```

###You can explore hyperparameter values

In [44]:
```
%%time

max_depths = list(range(2, 12, 2)) + [None]

for max_depth in max_depths:

    pipe = make_pipeline(
        ce.OrdinalEncoder(),
        RandomForestClassifier(
            n_estimators=100,
            class_weight='balanced',
            max_depth=max_depth,
            oob_score=True,
            n_jobs=-1
        )
    )

    pipe.fit(X_train, y_train)
    y_pred_proba = pipe.named_steps['randomforestclassifier'].oob_decisi
    print('Max Depth:', max_depth)
    print('ROC AUC, OOB:', roc_auc_score(y_train, y_pred_proba))
```

```
Max Depth: 2
ROC AUC, OOB: 0.6985295117020649
Max Depth: 4
ROC AUC, OOB: 0.7074454057927698
Max Depth: 6
ROC AUC, OOB: 0.712182426840721
Max Depth: 8
ROC AUC, OOB: 0.716150086711317
Max Depth: 10
ROC AUC, OOB: 0.7182985911393234
Max Depth: None
ROC AUC, OOB: 0.6980663211203326
CPU times: user 1h 31min 6s, sys: 2min 56s, total: 1h 34min 3s
Wall time: 35min 43s
```

Feature Importances

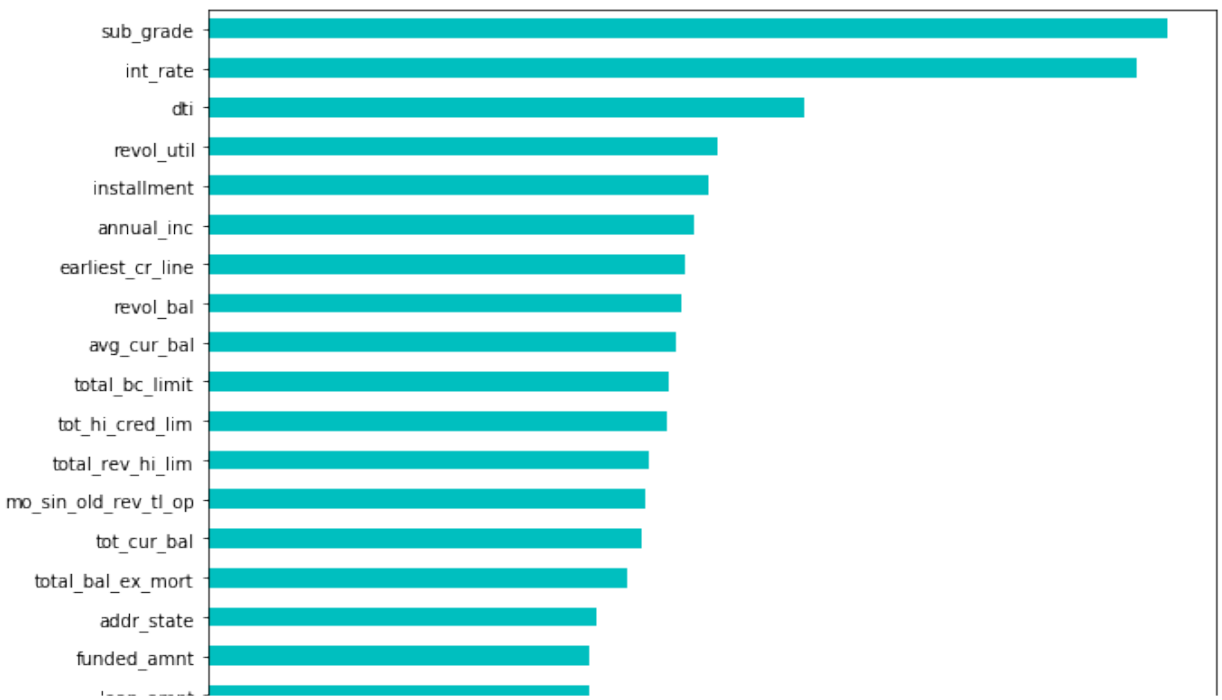
We can look at feature importances. [But remember: (https://blog.datadive.net/selecting-good-features-part-iii-random-forests/)](https://blog.datadive.net/selecting-good-features-part-iii-random-forests/)

> Firstly, feature selection based on impurity reduction is biased towards preferring variables with more categories.
>
> Secondly, when the dataset has two (or more) correlated features, then from the point of view of the model, any of these correlated features can be used as the predictor, with no concrete preference of one over the others.

Drop Column Importance / "Ablation Study"

`sub_grade` and `int_rate` are highly correlated. If we drop one of those features, the model uses the other more, so the score remains similar.

In [45]:
```
def show_feature_importances(
    pipe, X, y, estimator_name='randomforestclassifier',
    n=20, figsize=(8, 8), color='blue'):

    # pipe must not change dimensions of X dataframe
    pipe.fit(X, y)

    importances = pd.Series(
        pipe.named_steps[estimator_name].feature_importances_,
        X.columns)

    top_n = importances.sort_values(ascending=False)[:n]

    plt.figure(figsize=figsize)
    top_n.sort_values().plot.barh(color=color)


show_feature_importances(pipe, X_train, y_train, color='c')
```

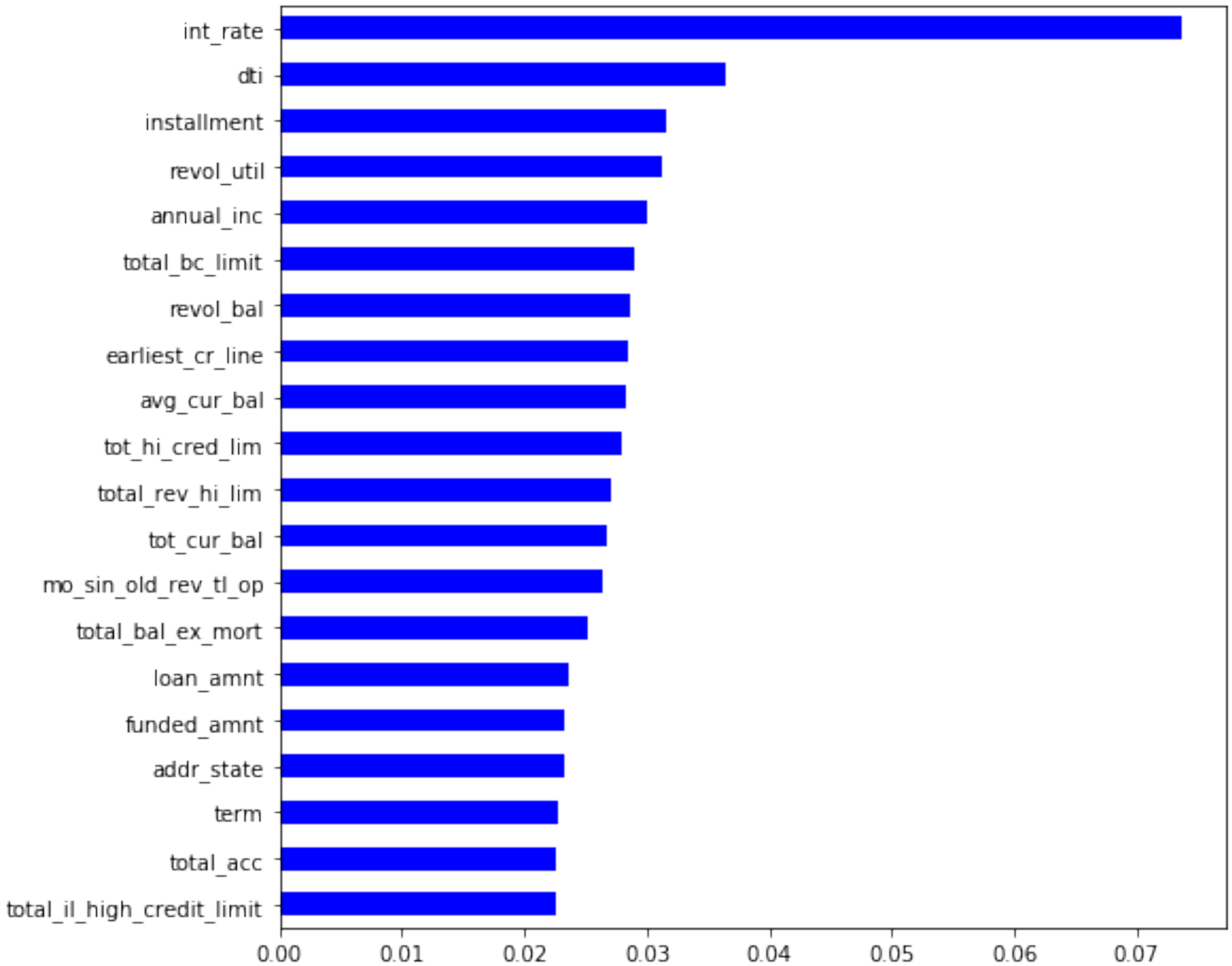


In [46]:
```
cross_val_score(pipe, X_train.drop(columns='sub_grade'), y_train, cv=5,
```

Out[46]:
```
array([0.71519181, 0.71073021, 0.71369791, 0.71529829, 0.7136068 ])
```

In [47]:
```
%%time
show_feature_importances(pipe, X_train.drop(columns='sub_grade'), y_trai
```

```
CPU times: user 31min 16s, sys: 1min 7s, total: 32min 24s
Wall time: 12min 52s
```



But if we drop *both* features, then the score decreases:

In [0]:
```
cross_val_score(pipe, X_train.drop(columns=['sub_grade', 'int_rate']), y
```

Out[16]:
```
array([0.70238724, 0.69620403, 0.69929053, 0.70917935, 0.69917305])
```

For more information, see [Beware Default Random Forest Importances (https://explained.ai/rf-importance/index.html)](https://explained.ai/rf-importance/index.html).

Permutation Importance

Permutation Importance is a compromise between Feature Importance based on impurity reduction (which is the fastest) and Drop Column Importance (which is the "best.")

[The ELI5 library documentation explains, (https://eli5.readthedocs.io/en/latest/blackbox/permutation_importance.html)](https://eli5.readthedocs.io/en/latest/blackbox/permutation_importance.html)

> Importance can be measured by looking at how much the score (accuracy, F1, R^2, etc. - any score we're interested in) decreases when a feature is not available.
>
> To do that one can remove feature from the dataset, re-train the estimator and check the score. But it requires re-training an estimator for each feature, which can be computationally intensive. ...
>
> To avoid re-training the estimator we can remove a feature only from the test part of the dataset, and compute score without using this feature. It doesn't work as-is, because estimators expect feature to be present. So instead of removing a feature we can replace it with random noise - feature column is still there, but it no longer contains useful information. This method works if noise is drawn from the same distribution as original feature values (as otherwise estimator may fail). The simplest way to get such noise is to shuffle values for a feature, i.e. use other examples' feature values - this is how permutation importance is computed.
>
> The method is most suitable for computing feature importances when a number of columns (features) is not huge; it can be resource-intensive otherwise.

For more documentation on using this library, see:

- eli5.sklearn.PermutationImportance (https://eli5.readthedocs.io/en/latest/autodocs/sklearn.html#eli5.sklearn.permutation_importa
- eli5.show_weights (https://eli5.readthedocs.io/en/latest/autodocs/eli5.html#eli5.show_weights)

In [48]:
```
%%time
import eli5
from eli5.sklearn import PermutationImportance

encoder = ce.OrdinalEncoder()
X_train_transformed = encoder.fit_transform(X_train)

model = RandomForestClassifier(
    n_estimators=100,
    class_weight='balanced',
    min_samples_leaf=0.005,
    n_jobs=-1)

model.fit(X_train_transformed, y_train)
permuter = PermutationImportance(model, scoring='roc_auc', n_iter=1, cv=
permuter.fit(X_train_transformed, y_train)
```

```
/Library/anaconda3/lib/python3.7/site-packages/lightgbm/__init__.py:46
: UserWarning: Starting from version 2.2.1, the library file in distri
bution wheels for macOS is built by the Apple Clang (Xcode_8.3.1) comp
iler.
This means that in case of installing LightGBM from PyPI via the ``pip
install lightgbm`` command, you don't need to install the gcc compiler
anymore.
Instead of that, you need to install the OpenMP library, which is requ
ired for running LightGBM on the system with the Apple Clang compiler.
You can install the OpenMP library by the following command: ``brew in
stall libomp``.
  "You can install the OpenMP library by the following command: ``brew
install libomp``.", UserWarning)

CPU times: user 49min 41s, sys: 2min 14s, total: 51min 56s
Wall time: 20min 49s
```

In [49]:
```
eli5.show_weights(permuter, top=None, feature_names=X_train_transformed.
```

Out[49]:

Weight	Feature
0.0325 ± 0.0000	sub_grade
0.0128 ± 0.0000	int_rate
0.0114 ± 0.0000	term
0.0032 ± 0.0000	dti
0.0022 ± 0.0000	acc_open_past_24mths
0.0013 ± 0.0000	avg_cur_bal
0.0013 ± 0.0000	annual_inc
0.0011 ± 0.0000	loan_amnt
0.0010 ± 0.0000	tot_hi_cred_lim
0.0009 ± 0.0000	funded_amnt
0.0009 ± 0.0000	mort_acc
0.0008 ± 0.0000	home_ownership
0.0008 ± 0.0000	total_bc_limit
0.0006 ± 0.0000	installment
0.0006 ± 0.0000	num_tl_op_past_12m
0.0006 ± 0.0000	num_rev_tl_bal_gt_0
0.0005 ± 0.0000	tot_cur_bal
0.0005 ± 0.0000	all_util
0.0005 ± 0.0000	num_actv_rev_tl
0.0004 ± 0.0000	emp_length
0.0004 ± 0.0000	total_rev_hi_lim
0.0004 ± 0.0000	revol_util

We can use Permutation Importance weights for feature selection. For example, we can remove features with zero weight. The model trains faster and the score does not decrease.

In [51]:
```
subset = X_train.columns[permuter.feature_importances_ > 0]
cross_val_score(pipe, X_train[subset], y_train, cv=5, scoring='roc_auc',
```

```
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurren
t workers.

[CV]  ...................................................................

/Library/anaconda3/lib/python3.7/site-packages/sklearn/model_selection
/_validation.py:542: FutureWarning: From version 0.22, errors during f
it will result in a cross validation score of NaN by default. Use erro
r_score='raise' if you want an exception raised or error_score=np.nan
to adopt the behavior from version 0.22.
  FutureWarning)

----------------------------------------------------------------------
-----
KeyError                                  Traceback (most recent call
last)
/Library/anaconda3/lib/python3.7/site-packages/pandas/core/indexes/bas
e.py in get_loc(self, key, method, tolerance)
   3077             try:
-> 3078                 return self._engine.get_loc(key)
```

In [ ]: