Mini Project: Tree-Based Algorithms

The "German Credit" Dataset

Dataset Details

This dataset has two classes (these would be considered labels in Machine Learning terms) to describe the worthiness of a personal loan: "Good" or "Bad". There are predictors related to attributes, such as: checking account status, duration, credit history, purpose of the loan, amount of the loan, savings accounts or bonds, employment duration, installment rate in percentage of disposable income, personal information, other debtors/guarantors, residence duration, property, age, other installment plans, housing, number of existing credits, job information, number of people being liable to provide maintenance for, telephone, and foreign worker status.

Many of these predictors are discrete and have been expanded into several 0/1 indicator variables (a.k.a. they have been one-hot-encoded).

This dataset has been kindly provided by Professor Dr. Hans Hofmann of the University of Hamburg, and can also be found on the UCI Machine Learning Repository.

Decision Trees

As we have learned in the previous lectures, Decision Trees as a family of algorithms (irrespective to the particular implementation) are powerful algorithms that can produce models with a predictive accuracy higher than that produced by linear models, such as Linear or Logistic Regression. Primarily, this is due to the fact the DT's can model nonlinear relationships, and also have a number of tuning paramters, that allow for the practicioner to achieve the best possible model. An added bonus is the ability to visualize the trained Decision Tree model, which allows for some insight into how the model has produced the predictions that it has. One caveat here, to keep in mind, is that sometimes, due to the size of the dataset (both in the sense of the number of records, as well as the number of features), the visualization might prove to be very large and complex, increasing the difficulty of interpretation.

To give you a very good example of how Decision Trees can be visualized and interpreted, we would strongly recommend that, before continuing on with solving the problems in this Mini Project, you take the time to read this fanstastic, detailed and informative blog post: http://explained.ai/decision-tree-viz/index.html (http://explained.ai/decision-tree-viz/index.html)

Building Your First Decision Tree Model

So, now it's time to jump straight into the heart of the matter. Your first task, is to build a Decision Tree model, using the aforementioned "German Credit" dataset, which contains 1,000 records, and 62 columns (one of them presents the labels, and the other 61 present the potential features for the model.)

For this task, you will be using the scikit-learn library, which comes already pre-installed with the Anaconda Python distribution. In case you're not using that, you can easily install it using pip.

Before embarking on creating your first model, we would strongly encourage you to read the short tutorial for Decision Trees in scikit-learn (http://scikit-learn.org/stable/modules/tree.html (http://scikit-learn.org/stable/modules/tree.html)), and then dive a bit deeper into the documentation of the algorithm itself (http://scikit-learn.org/stable/modules/generated/sklearn.tree.DecisionTreeClassifier.html (http://scikit-learn.org/stable/modules/generated/sklearn.tree.DecisionTreeClassifier.html)).

Also, since you want to be able to present the results of your model, we suggest you take a look at the tutorial for accuracy metrics for classification models (http://scikit-learn.org/stable/modules/model_evaluation.html#classification-report (http://scikit-learn.org/stable/modules/model_evaluation.html#classification-report)) as well as the more detailed documentation (http://scikit-learn.org/stable/modules/generated/sklearn.metrics.classification_report.html (http://scikit-learn.org/stable/modules/generated/sklearn.metrics.classification_report.html)).

Finally, an *amazing* resource that explains the various classification model accuracy metrics, as well as the relationships between them, can be found on Wikipedia: https://en.wikipedia.org/wiki/Confusion_matrix (https://en.wikipedia.org/wiki/Confusion_matrix)

(Note: as you've already learned in the Logistic Regression mini project, a standard practice in Machine Learning for achieving the best possible result when training a model is to use hyperparameter tuning, through Grid Search and k-fold Cross Validation. We strongly encourage you to use it here as well, not just because it's standard practice, but also becuase it's not going to be computationally to intensive, due to the size of the dataset that you're working with. Our suggestion here is that you split the data into 70% training, and 30% testing. Then, do the hyperparameter tuning and Cross Validation on the training set, and afterwards to a final test on the testing set.)

Now we pass the torch onto you! You can start building your first Decision Tree model! :)

In [9]:
```
import pandas as pd
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import classification_report
from sklearn.model_selection import train_test_split
from sklearn.model_selection import GridSearchCV
```

In [10]:
```
#retrieve the data and check for any nulls

df = pd.read_csv("GermanCredit.csv", sep = ',', header = 0)

print("shape", df.shape)

#Checking for missing data
NAs = pd.concat([df.isnull().sum()], axis=1) #, keys=[‘Train’])
NAs[NAs.sum(axis=1) > 0]

#pd.concat([df], axis=1) # , keys=['Train'])
#pd.concat([df.isnull()], axis=1)
#pd.concat([df.isnull().sum()], axis=1)
```

```
shape (1000, 62)
```

Out[10]:

0

In [11]:
```
df.columns
```

Out[11]:
```
Index(['Duration', 'Amount', 'InstallmentRatePercentage', 'ResidenceDuratio
n',
       'Age', 'NumberExistingCredits', 'NumberPeopleMaintenance', 'Telephon
e',
       'ForeignWorker', 'Class', 'CheckingAccountStatus.lt.0',
       'CheckingAccountStatus.0.to.200', 'CheckingAccountStatus.gt.200',
       'CheckingAccountStatus.none', 'CreditHistory.NoCredit.AllPaid',
       'CreditHistory.ThisBank.AllPaid', 'CreditHistory.PaidDuly',
       'CreditHistory.Delay', 'CreditHistory.Critical', 'Purpose.NewCar',
       'Purpose.UsedCar', 'Purpose.Furniture.Equipment',
       'Purpose.Radio.Television', 'Purpose.DomesticAppliance',
       'Purpose.Repairs', 'Purpose.Education', 'Purpose.Vacation',
       'Purpose.Retraining', 'Purpose.Business', 'Purpose.Other',
       'SavingsAccountBonds.lt.100', 'SavingsAccountBonds.100.to.500',
       'SavingsAccountBonds.500.to.1000', 'SavingsAccountBonds.gt.1000',
       'SavingsAccountBonds.Unknown', 'EmploymentDuration.lt.1',
       'EmploymentDuration.1.to.4', 'EmploymentDuration.4.to.7',
       'EmploymentDuration.gt.7', 'EmploymentDuration.Unemployed',
       'Personal.Male.Divorced.Seperated', 'Personal.Female.NotSingle',
       'Personal.Male.Single', 'Personal.Male.Married.Widowed',
       'Personal.Female.Single', 'OtherDebtorsGuarantors.None',
       'OtherDebtorsGuarantors.CoApplicant',
       'OtherDebtorsGuarantors.Guarantor', 'Property.RealEstate',
       'Property.Insurance', 'Property.CarOther', 'Property.Unknown',
       'OtherInstallmentPlans.Bank', 'OtherInstallmentPlans.Stores',
       'OtherInstallmentPlans.None', 'Housing.Rent', 'Housing.Own',
       'Housing.ForFree', 'Job.UnemployedUnskilled', 'Job.UnskilledResident',
       'Job.SkilledEmployee', 'Job.Management.SelfEmp.HighlyQualified'],
      dtype='object')
```

In [19]:
```
# Your code here! :)


import numpy as np
from sklearn.metrics import confusion_matrix
from sklearn.tree import DecisionTreeClassifier
#from sklearn.metrics import accuracy_score

#put data attributes into X
#put classifier in Y
X = df.drop(['Class'], axis=1)
Y = df['Class']

print(df.shape)
print(X.shape)
print(Y.shape)

#split the data into 70% training, and 30% testing
X_train, X_test, y_train, y_test = train_test_split(X, Y,
    random_state=100,
    test_size = 0.30,
    train_size = 0.70)

print(X_train.shape)
print(y_train.shape)

print(X_test.shape)
print(y_test.shape)
```

```
(1000, 62)
(1000, 61)
(1000,)
(700, 61)
(700,)
(300, 61)
(300,)
```

In [21]:
```
# use GridSearchCV to do k-fold and grid optimization

from sklearn import svm, datasets
from sklearn.model_selection import GridSearchCV
import numpy as np

max_depths = np.linspace(1, 32, 32, endpoint=True)
min_samples_splits = np.linspace(0.1, 1.0, 10, endpoint=True)
min_samples_leafs = np.linspace(0.1, 0.5, 5, endpoint=True)

#param ranges used during the optimization effort
param_grid = {
    'max_depth': max_depths,
    'min_samples_split': min_samples_splits,
    'min_samples_leaf': min_samples_leafs,
    "criterion" : ['gini', 'entropy'],
    #'n_estimators': [200, 700],
    'max_features': ['auto', 'sqrt', 'log2']
}

dtc = DecisionTreeClassifier(random_state=100)

#optimize dtc then fit
GSCV_dtc = GridSearchCV(estimator = dtc, param_grid = param_grid, cv = 5, n_jo
bs=-1)

GSCV_dtc.fit(X, Y)

#show the optimized params
print (GSCV_dtc.best_params_)
```

```
{'criterion': 'entropy', 'max_depth': 4.0, 'max_features': 'log2', 'min_sampl
es_leaf': 0.1, 'min_samples_split': 0.1}
```

In [31]:
```
best_model = GSCV_dtc.best_estimator_
```

In [22]:
```
from sklearn.metrics import accuracy_score

#predict and evaluate the accuracy

y_predict  = GSCV_dtc.predict(X_test)

print ("Accuracy Score : ",
accuracy_score(y_test, y_predict)) # sklearn.metrics.accuracy_score(y_true, y_
pred, normalize=True, sample_weight=None)

print("Report : ",
classification_report(y_test, y_predict))
```

```
Accuracy Score :  0.74
Report :                precision    recall  f1-score   support

         Bad       0.54      0.23      0.33        81
        Good       0.77      0.93      0.84       219

    accuracy                           0.74       300
   macro avg       0.65      0.58      0.58       300
weighted avg       0.71      0.74      0.70       300
```

After you've built the best model you can, now it's time to visualize it!

Rememeber that amazing blog post from a few paragraphs ago, that demonstrated how to visualize and interpret the results of your Decision Tree model. We've seen that this can perform very well, but let's see how it does on the "German Credit" dataset that we're working on, due to it being a bit larger than the one used by the blog authors.

First, we're going to need to install their package. If you're using Anaconda, this can be done easily by running:

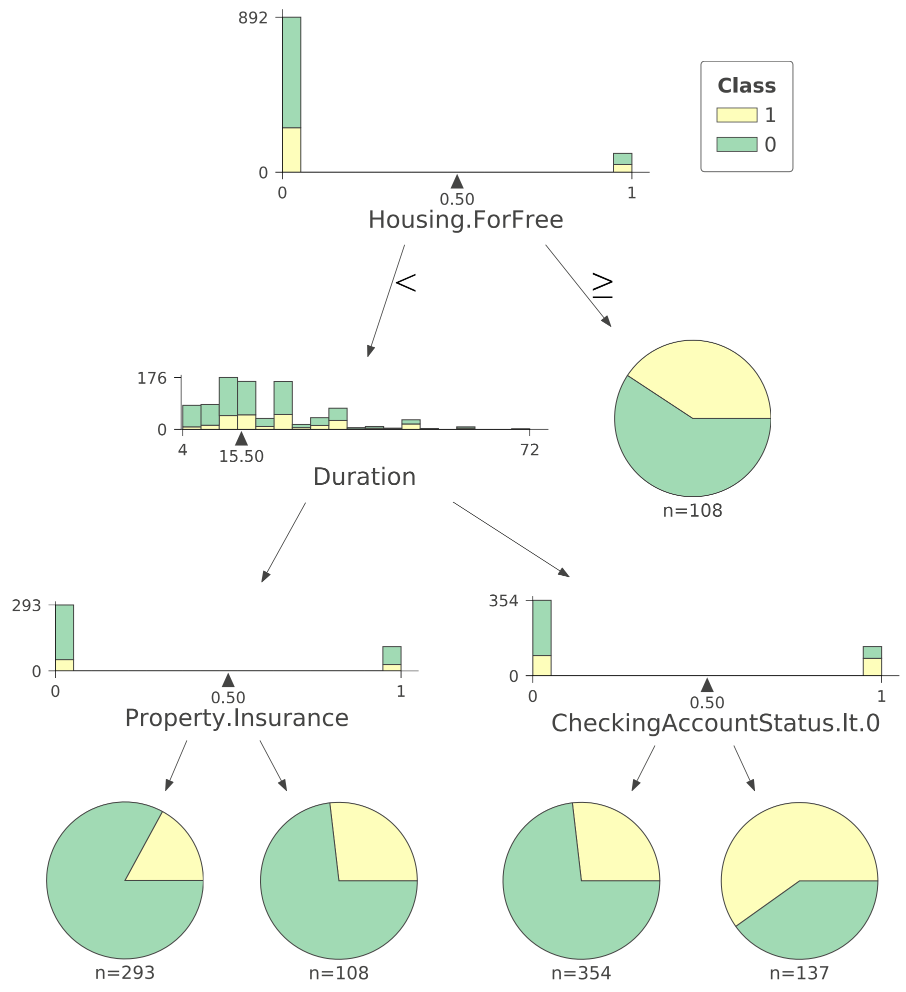
In [ ]:
```
! pip install dtreeviz
```

If for any reason this way of installing doesn't work for you straight out of the box, please refer to the more detailed documentation here: https://github.com/parrt/dtreeviz (https://github.com/parrt/dtreeviz)

Now you're ready to visualize your Decision Tree model! Please feel free to use the blog post for guidance and inspiration!

In [65]:
```
%matplotlib inline

from sklearn.datasets import *
from sklearn import tree
from dtreeviz.trees import *

mapped = Y.map({'Good': 1, 'Bad': 0})

dtreeviz(best_model, X, mapped,
         target_name = 'Class',
         feature_names = X.columns,
         class_names = mapped.unique().tolist())
```

Out[65]:



Random Forests

As discussed in the lecture videos, Decision Tree algorithms also have certain undesireable properties. Mainly the have low bias, which is good, but tend to have high variance - which is *not* so good (more about this problem here: https://en.wikipedia.org/wiki/Bias%E2%80%93variance_tradeoff (https://en.wikipedia.org/wiki/Bias%E2%80%93variance_tradeoff)).

Noticing these problems, the late Professor Leo Breiman, in 2001, developed the Random Forests algorithm, which mitigates these problems, while at the same time providing even higher predictive accuracy than the majority of Decision Tree algorithm implementations. While the curriculum contains two excellent lectures on Random Forests, if you're interested, you can dive into the original paper here: https://link.springer.com/content/pdf/10.1023%2FA%3A1010933404324.pdf (https://link.springer.com/content/pdf/10.1023%2FA%3A1010933404324.pdf).

In the next part of this assignment, your are going to use the same "German Credit" dataset to train, tune, and measure the performance of a Random Forests model. You will also see certain functionalities that this model, even though it's a bit of a "black box", provides for some degree of interpretability.

First, let's build a Random Forests model, using the same best practices that you've used for your Decision Trees model. You can reuse the things you've already imported there, so no need to do any re-imports, new train/test splits, or loading up the data again.

In [8]:
```
from sklearn.ensemble import RandomForestClassifier
```

```
C:\Users\wrgorman\AppData\Local\Continuum\anaconda3\lib\site-packages\sklearn
\ensemble\weight_boosting.py:29: DeprecationWarning: numpy.core.umath_tests i
s an internal NumPy module and should not be imported. It will be removed in
a future NumPy release.
  from numpy.core.umath_tests import inner1d
```

In [9]:
```
# Your code here! :)

import numpy as np

#number of trees in the forest
#n_estimators  = np.linspace(10, 100, 10, endpoint=True)
n_estimators = [10, 50, 100]
#The function to measure the quality of a split. Supported criteria are “gini”
for the Gini impurity and
#“entropy” for the information gain.
#Note: this parameter is tree-specific.
criterion = ['gini', 'entropy']

print(n_estimators)
#print(np.reshape(n_estimators, (-1,1)))
#print(np.reshape(n_estimators, (1,-1)))
#n_estimators = np.reshape(n_estimators, (1,-1))
print(np.ravel(n_estimators))
print(np.reshape(n_estimators, (-1,)))
print(np.reshape(n_estimators, (1, -1)))

n_estimators = np.reshape(n_estimators, (-1,))
n_estimators = np.ravel(n_estimators)

print(criterion)


param_grid = {
    'n_estimators': n_estimators,
    'criterion': criterion

}

rfc = RandomForestClassifier()


GSCV_rfc = GridSearchCV(estimator = rfc, param_grid = param_grid, cv = 5)

X = df.drop(['Class'], axis=1)
Y = df[['Class']]

GSCV_rfc.fit(X, Y)
print (GSCV_rfc.best_params_) # {'max_depth': 2.0, 'min_samples_leaf': 0.2, 'm
in_samples_split': 0.1}


#split the data into 70% training, and 30% testing
X_train, X_test, y_train, y_test = train_test_split(X, Y,
    random_state=100,
    test_size = 0.30,
    train_size = 0.70)



from sklearn.metrics import accuracy_score
```

```
y_predict  = GSCV_rfc.predict(X_test)
#model.fit(X, Y)
#y_predict  = model.predict(X_test)

print ("Accuracy : ",
accuracy_score(y_test, y_predict)) # sklearn.metrics.accuracy_score(y_true, y_
pred, normalize=True, sample_weight=None)

# sklearn.metrics.classification_report(y_true, y_pred, labels=None, target_na
mes=None, sample_weight=None, digits=2,
#    output_dict=False)[source]

#target_names = ['class 0', 'class 1', 'class 2']
print("Report : ",
classification_report(y_test, y_predict) ) #, target_names = target_names))
```

```
[10, 50, 100]
[ 10  50 100]
[ 10  50 100]
[[ 10  50 100]]
['gini', 'entropy']
```

```
C:\Users\wrgorman\AppData\Local\Continuum\anaconda3\lib\site-packages\sklearn
\model_selection\_validation.py:458: DataConversionWarning: A column-vector y
was passed when a 1d array was expected. Please change the shape of y to (n_s
amples,), for example using ravel().
  estimator.fit(X_train, y_train, **fit_params)
C:\Users\wrgorman\AppData\Local\Continuum\anaconda3\lib\site-packages\sklearn
\model_selection\_validation.py:458: DataConversionWarning: A column-vector y
was passed when a 1d array was expected. Please change the shape of y to (n_s
amples,), for example using ravel().
  estimator.fit(X_train, y_train, **fit_params)
C:\Users\wrgorman\AppData\Local\Continuum\anaconda3\lib\site-packages\sklearn
\model_selection\_validation.py:458: DataConversionWarning: A column-vector y
was passed when a 1d array was expected. Please change the shape of y to (n_s
amples,), for example using ravel().
  estimator.fit(X_train, y_train, **fit_params)
C:\Users\wrgorman\AppData\Local\Continuum\anaconda3\lib\site-packages\sklearn
\model_selection\_validation.py:458: DataConversionWarning: A column-vector y
was passed when a 1d array was expected. Please change the shape of y to (n_s
amples,), for example using ravel().
  estimator.fit(X_train, y_train, **fit_params)
C:\Users\wrgorman\AppData\Local\Continuum\anaconda3\lib\site-packages\sklearn
\model_selection\_validation.py:458: DataConversionWarning: A column-vector y
was passed when a 1d array was expected. Please change the shape of y to (n_s
amples,), for example using ravel().
  estimator.fit(X_train, y_train, **fit_params)
C:\Users\wrgorman\AppData\Local\Continuum\anaconda3\lib\site-packages\sklearn
\model_selection\_validation.py:458: DataConversionWarning: A column-vector y
was passed when a 1d array was expected. Please change the shape of y to (n_s
amples,), for example using ravel().
  estimator.fit(X_train, y_train, **fit_params)
C:\Users\wrgorman\AppData\Local\Continuum\anaconda3\lib\site-packages\sklearn
\model_selection\_validation.py:458: DataConversionWarning: A column-vector y
was passed when a 1d array was expected. Please change the shape of y to (n_s
amples,), for example using ravel().
  estimator.fit(X_train, y_train, **fit_params)
C:\Users\wrgorman\AppData\Local\Continuum\anaconda3\lib\site-packages\sklearn
\model_selection\_validation.py:458: DataConversionWarning: A column-vector y
was passed when a 1d array was expected. Please change the shape of y to (n_s
amples,), for example using ravel().
  estimator.fit(X_train, y_train, **fit_params)
C:\Users\wrgorman\AppData\Local\Continuum\anaconda3\lib\site-packages\sklearn
\model_selection\_validation.py:458: DataConversionWarning: A column-vector y
was passed when a 1d array was expected. Please change the shape of y to (n_s
amples,), for example using ravel().
  estimator.fit(X_train, y_train, **fit_params)
C:\Users\wrgorman\AppData\Local\Continuum\anaconda3\lib\site-packages\sklearn
\model_selection\_validation.py:458: DataConversionWarning: A column-vector y
was passed when a 1d array was expected. Please change the shape of y to (n_s
amples,), for example using ravel().
  estimator.fit(X_train, y_train, **fit_params)
C:\Users\wrgorman\AppData\Local\Continuum\anaconda3\lib\site-packages\sklearn
\model_selection\_validation.py:458: DataConversionWarning: A column-vector y
was passed when a 1d array was expected. Please change the shape of y to (n_s
amples,), for example using ravel().
  estimator.fit(X_train, y_train, **fit_params)
C:\Users\wrgorman\AppData\Local\Continuum\anaconda3\lib\site-packages\sklearn
\model_selection\_validation.py:458: DataConversionWarning: A column-vector y
```

```
was passed when a 1d array was expected. Please change the shape of y to (n_s
amples,), for example using ravel().
  estimator.fit(X_train, y_train, **fit_params)
C:\Users\wrgorman\AppData\Local\Continuum\anaconda3\lib\site-packages\sklearn
\model_selection\_validation.py:458: DataConversionWarning: A column-vector y
was passed when a 1d array was expected. Please change the shape of y to (n_s
amples,), for example using ravel().
  estimator.fit(X_train, y_train, **fit_params)
C:\Users\wrgorman\AppData\Local\Continuum\anaconda3\lib\site-packages\sklearn
\model_selection\_validation.py:458: DataConversionWarning: A column-vector y
was passed when a 1d array was expected. Please change the shape of y to (n_s
amples,), for example using ravel().
  estimator.fit(X_train, y_train, **fit_params)
C:\Users\wrgorman\AppData\Local\Continuum\anaconda3\lib\site-packages\sklearn
\model_selection\_validation.py:458: DataConversionWarning: A column-vector y
was passed when a 1d array was expected. Please change the shape of y to (n_s
amples,), for example using ravel().
  estimator.fit(X_train, y_train, **fit_params)
C:\Users\wrgorman\AppData\Local\Continuum\anaconda3\lib\site-packages\sklearn
\model_selection\_validation.py:458: DataConversionWarning: A column-vector y
was passed when a 1d array was expected. Please change the shape of y to (n_s
amples,), for example using ravel().
  estimator.fit(X_train, y_train, **fit_params)
C:\Users\wrgorman\AppData\Local\Continuum\anaconda3\lib\site-packages\sklearn
\model_selection\_validation.py:458: DataConversionWarning: A column-vector y
was passed when a 1d array was expected. Please change the shape of y to (n_s
amples,), for example using ravel().
  estimator.fit(X_train, y_train, **fit_params)
C:\Users\wrgorman\AppData\Local\Continuum\anaconda3\lib\site-packages\sklearn
\model_selection\_validation.py:458: DataConversionWarning: A column-vector y
was passed when a 1d array was expected. Please change the shape of y to (n_s
amples,), for example using ravel().
  estimator.fit(X_train, y_train, **fit_params)
C:\Users\wrgorman\AppData\Local\Continuum\anaconda3\lib\site-packages\sklearn
\model_selection\_validation.py:458: DataConversionWarning: A column-vector y
was passed when a 1d array was expected. Please change the shape of y to (n_s
amples,), for example using ravel().
  estimator.fit(X_train, y_train, **fit_params)
C:\Users\wrgorman\AppData\Local\Continuum\anaconda3\lib\site-packages\sklearn
\model_selection\_validation.py:458: DataConversionWarning: A column-vector y
was passed when a 1d array was expected. Please change the shape of y to (n_s
amples,), for example using ravel().
  estimator.fit(X_train, y_train, **fit_params)
C:\Users\wrgorman\AppData\Local\Continuum\anaconda3\lib\site-packages\sklearn
\model_selection\_validation.py:458: DataConversionWarning: A column-vector y
was passed when a 1d array was expected. Please change the shape of y to (n_s
amples,), for example using ravel().
  estimator.fit(X_train, y_train, **fit_params)
C:\Users\wrgorman\AppData\Local\Continuum\anaconda3\lib\site-packages\sklearn
\model_selection\_validation.py:458: DataConversionWarning: A column-vector y
was passed when a 1d array was expected. Please change the shape of y to (n_s
amples,), for example using ravel().
  estimator.fit(X_train, y_train, **fit_params)
C:\Users\wrgorman\AppData\Local\Continuum\anaconda3\lib\site-packages\sklearn
\model_selection\_validation.py:458: DataConversionWarning: A column-vector y
was passed when a 1d array was expected. Please change the shape of y to (n_s
amples,), for example using ravel().
```

```
  estimator.fit(X_train, y_train, **fit_params)
C:\Users\wrgorman\AppData\Local\Continuum\anaconda3\lib\site-packages\sklearn
\model_selection\_validation.py:458: DataConversionWarning: A column-vector y
was passed when a 1d array was expected. Please change the shape of y to (n_s
amples,), for example using ravel().
  estimator.fit(X_train, y_train, **fit_params)
C:\Users\wrgorman\AppData\Local\Continuum\anaconda3\lib\site-packages\sklearn
\model_selection\_validation.py:458: DataConversionWarning: A column-vector y
was passed when a 1d array was expected. Please change the shape of y to (n_s
amples,), for example using ravel().
  estimator.fit(X_train, y_train, **fit_params)
C:\Users\wrgorman\AppData\Local\Continuum\anaconda3\lib\site-packages\sklearn
\model_selection\_validation.py:458: DataConversionWarning: A column-vector y
was passed when a 1d array was expected. Please change the shape of y to (n_s
amples,), for example using ravel().
  estimator.fit(X_train, y_train, **fit_params)
C:\Users\wrgorman\AppData\Local\Continuum\anaconda3\lib\site-packages\sklearn
\model_selection\_validation.py:458: DataConversionWarning: A column-vector y
was passed when a 1d array was expected. Please change the shape of y to (n_s
amples,), for example using ravel().
  estimator.fit(X_train, y_train, **fit_params)
C:\Users\wrgorman\AppData\Local\Continuum\anaconda3\lib\site-packages\sklearn
\model_selection\_validation.py:458: DataConversionWarning: A column-vector y
was passed when a 1d array was expected. Please change the shape of y to (n_s
amples,), for example using ravel().
  estimator.fit(X_train, y_train, **fit_params)
C:\Users\wrgorman\AppData\Local\Continuum\anaconda3\lib\site-packages\sklearn
\model_selection\_validation.py:458: DataConversionWarning: A column-vector y
was passed when a 1d array was expected. Please change the shape of y to (n_s
amples,), for example using ravel().
  estimator.fit(X_train, y_train, **fit_params)
C:\Users\wrgorman\AppData\Local\Continuum\anaconda3\lib\site-packages\sklearn
\model_selection\_validation.py:458: DataConversionWarning: A column-vector y
was passed when a 1d array was expected. Please change the shape of y to (n_s
amples,), for example using ravel().
  estimator.fit(X_train, y_train, **fit_params)
C:\Users\wrgorman\AppData\Local\Continuum\anaconda3\lib\site-packages\sklearn
\model_selection\_search.py:740: DataConversionWarning: A column-vector y was
passed when a 1d array was expected. Please change the shape of y to (n_sampl
es,), for example using ravel().
  self.best_estimator_.fit(X, y, **fit_params)

{'criterion': 'gini', 'n_estimators': 50}
Accuracy :  1.0
Report :               precision    recall  f1-score   support

        Bad       1.00      1.00      1.00        81
       Good       1.00      1.00      1.00       219

avg / total       1.00      1.00      1.00       300
```

As mentioned, there are certain ways to "peek" into a model created by the Random Forests algorithm. The first, and most popular one, is the Feature Importance calculation functionality. This allows the ML practitioner to see an ordering of the importance of the features that have contributed the most to the predictive accuracy of the model.

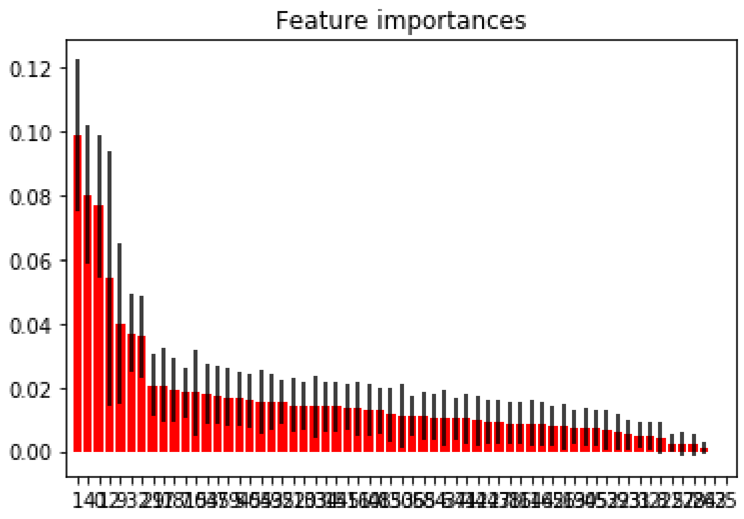
You can see how to use this in the scikit-learn documentation (http://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html#sklearn.ensemble.RandomFo
(http://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html#sklearn.ensemble.RandomFo
Now, if you tried this, you would just get an ordered table of not directly interpretable numeric values. Thus, it's much more useful to show the feature importance in a visual way. You can see an example of how that's done here: http://scikit-learn.org/stable/auto_examples/ensemble/plot_forest_importances.html#sphx-glr-auto-examples-ensemble-plot-forest-importances-py (http://scikit-learn.org/stable/auto_examples/ensemble/plot_forest_importances.html#sphx-glr-auto-examples-ensemble-plot-forest-importances-py)

Now you try! Let's visualize the importance of features from your Random Forests model!

In [27]:
```
# Your code here

#print(GSCV_rfc.best_estimator_.feature_importances_)

importances = GSCV_rfc.best_estimator_.feature_importances_

#print(GSCV_rfc.best_estimator_.estimators_)

forest = GSCV_rfc.best_estimator_.estimators_

print('tree count', len(GSCV_rfc.best_estimator_.estimators_))
#print('forest', forest)
print('feature importances', len(importances))
print('features', len(X.columns))

#get list of stdevs for each of the importances of each of the 61 features (co
lumnns)
std = np.std([tree.feature_importances_ for tree in forest], axis=0)

indices = np.argsort(importances)[::-1]

# Print the feature ranking
print("Feature ranking:")

for f in range(X.shape[1]):
    print("%d. feature %d (%f)" % (f + 1, indices[f], importances[indices[f
]]))

# Plot the feature importances of the forest
plt.figure()
plt.title("Feature importances")
plt.bar(range(X.shape[1]), importances[indices],
       color="r", yerr=std[indices], align="center")
plt.xticks(range(X.shape[1]), indices)
plt.xlim([-1, X.shape[1]])
plt.show()
```

```
tree count 50
feature importances 61
features 61
num of stdevs 61
Feature ranking:
1. feature 1 (0.098863)
2. feature 4 (0.080267)
3. feature 0 (0.076888)
4. feature 12 (0.054027)
5. feature 9 (0.039829)
6. feature 3 (0.036862)
7. feature 2 (0.035962)
8. feature 29 (0.020648)
9. feature 17 (0.020629)
10. feature 18 (0.019097)
11. feature 7 (0.018314)
12. feature 10 (0.018274)
13. feature 53 (0.018008)
14. feature 47 (0.017552)
15. feature 59 (0.016858)
16. feature 5 (0.016553)
17. feature 40 (0.015750)
18. feature 55 (0.015628)
19. feature 49 (0.015463)
20. feature 35 (0.015315)
21. feature 21 (0.014334)
22. feature 20 (0.014224)
23. feature 33 (0.013974)
24. feature 34 (0.013936)
25. feature 15 (0.013884)
26. feature 41 (0.013766)
27. feature 51 (0.013414)
28. feature 60 (0.012726)
29. feature 48 (0.012669)
30. feature 13 (0.011525)
31. feature 50 (0.011289)
32. feature 36 (0.011094)
33. feature 58 (0.010847)
34. feature 54 (0.010682)
35. feature 6 (0.010380)
36. feature 37 (0.010173)
37. feature 44 (0.010168)
38. feature 14 (0.009588)
39. feature 24 (0.009352)
40. feature 27 (0.009313)
41. feature 38 (0.008821)
42. feature 16 (0.008796)
43. feature 11 (0.008667)
44. feature 46 (0.008269)
45. feature 42 (0.008052)
46. feature 56 (0.007835)
47. feature 19 (0.007566)
48. feature 30 (0.007441)
49. feature 45 (0.007081)
50. feature 52 (0.006598)
51. feature 39 (0.005906)
52. feature 23 (0.005086)
```

```
53. feature 31 (0.004946)
54. feature 32 (0.004880)
55. feature 8 (0.004074)
56. feature 22 (0.002492)
57. feature 57 (0.002362)
58. feature 28 (0.002026)
59. feature 26 (0.000977)
60. feature 43 (0.000000)
61. feature 25 (0.000000)
```



A final method for gaining some insight into the inner working of your Random Forests models is a so-called Partial Dependence Plot. The Partial Dependence Plot (PDP or PD plot) shows the marginal effect of a feature on the predicted outcome of a previously fit model. The prediction function is fixed at a few values of the chosen features and averaged over the other features. A partial dependence plot can show if the relationship between the target and a feature is linear, monotonic or more complex.

In scikit-learn, PDPs are implemented and available for certain algorithms, but at this point (version 0.20.0) they are not yet implemented for Random Forests. Thankfully, there is an add-on package called **PDPbox** (https://pdpbox.readthedocs.io/en/latest/ (https://pdpbox.readthedocs.io/en/latest/)) which adds this functionality to Random Forests. The package is easy to install through pip.

```
In [11]: ! pip install pdpbox
```

```
Requirement already satisfied: pdpbox in c:\users\wrgorman\appdata\local\cont
inuum\anaconda3\lib\site-packages (0.2.0)
Requirement already satisfied: pandas in c:\users\wrgorman\appdata\local\cont
inuum\anaconda3\lib\site-packages (from pdpbox) (0.23.4)
Requirement already satisfied: numpy in c:\users\wrgorman\appdata\local\conti
nuum\anaconda3\lib\site-packages (from pdpbox) (1.15.1)
Requirement already satisfied: psutil in c:\users\wrgorman\appdata\local\cont
inuum\anaconda3\lib\site-packages (from pdpbox) (5.4.7)
Requirement already satisfied: scikit-learn in c:\users\wrgorman\appdata\loca
l\continuum\anaconda3\lib\site-packages (from pdpbox) (0.19.2)
Requirement already satisfied: scipy in c:\users\wrgorman\appdata\local\conti
nuum\anaconda3\lib\site-packages (from pdpbox) (1.1.0)
Requirement already satisfied: joblib in c:\users\wrgorman\appdata\local\cont
inuum\anaconda3\lib\site-packages (from pdpbox) (0.13.2)
Requirement already satisfied: matplotlib>=2.1.2 in c:\users\wrgorman\appdata
\local\continuum\anaconda3\lib\site-packages (from pdpbox) (2.2.3)
Requirement already satisfied: python-dateutil>=2.5.0 in c:\users\wrgorman\ap
pdata\local\continuum\anaconda3\lib\site-packages (from pandas->pdpbox) (2.7.
3)
Requirement already satisfied: pytz>=2011k in c:\users\wrgorman\appdata\local
\continuum\anaconda3\lib\site-packages (from pandas->pdpbox) (2018.5)
Requirement already satisfied: cycler>=0.10 in c:\users\wrgorman\appdata\loca
l\continuum\anaconda3\lib\site-packages (from matplotlib>=2.1.2->pdpbox) (0.1
0.0)
Requirement already satisfied: pyparsing!=2.0.4,!=2.1.2,!=2.1.6,>=2.0.1 in
c:\users\wrgorman\appdata\local\continuum\anaconda3\lib\site-packages (from m
atplotlib>=2.1.2->pdpbox) (2.2.0)
Requirement already satisfied: six>=1.10 in c:\users\wrgorman\appdata\local\c
ontinuum\anaconda3\lib\site-packages (from matplotlib>=2.1.2->pdpbox) (1.11.
0)
Requirement already satisfied: kiwisolver>=1.0.1 in c:\users\wrgorman\appdata
\local\continuum\anaconda3\lib\site-packages (from matplotlib>=2.1.2->pdpbox)
(1.0.1)
Requirement already satisfied: setuptools in c:\users\wrgorman\appdata\local
\continuum\anaconda3\lib\site-packages (from kiwisolver>=1.0.1->matplotlib>=
2.1.2->pdpbox) (40.2.0)
```

While we encourage you to read the documentation for the package (and reading package documentation in general is a good habit to develop), the authors of the package have also written an excellent blog post on how to use it, showing examples on different algorithms from scikit-learn (the Random Forests example is towards the end of the blog post): https://briangriner.github.io/Partial_Dependence_Plots_presentation-BrianGriner-PrincetonPublicLibrary-4.14.18-updated-4.22.18.html (https://briangriner.github.io/Partial_Dependence_Plots_presentation-BrianGriner-PrincetonPublicLibrary-4.14.18-updated-4.22.18.html)

So, armed with this new knowledge, feel free to pick a few features, and make a couple of Partial Dependence Plots of your own!

```
In [12]: # Your code here!
```

(Optional) Advanced Boosting-Based Algorithms

As explained in the video lectures, the next generation of algorithms after Random Forests (that use Bagging, a.k.a. Bootstrap Aggregation) were developed using Boosting, and the first one of these were Gradient Boosted Machines, which are implemented in scikit-learn (http://scikit-learn.org/stable/modules/ensemble.html#gradient-tree-boosting (http://scikit-learn.org/stable/modules/ensemble.html#gradient-tree-boosting)).

Still, in recent years, a number of variations on GBMs have been developed by different research amd industry groups, all of them bringing improvements, both in speed, accuracy and functionality to the original Gradient Boosting algorithms.

In no order of preference, these are:

1. **XGBoost**: https://xgboost.readthedocs.io/en/latest/ (https://xgboost.readthedocs.io/en/latest/)
2. **CatBoost**: https://tech.yandex.com/catboost/ (https://tech.yandex.com/catboost/)
3. **LightGBM**: https://lightgbm.readthedocs.io/en/latest/ (https://lightgbm.readthedocs.io/en/latest/)

If you're using the Anaconda distribution, these are all very easy to install:

In [ ]:
```
! conda install -c anaconda py-xgboost
```

In [ ]:
```
! conda install -c conda-forge catboost
```

In [ ]:
```
! conda install -c conda-forge lightgbm
```

Your task in this optional section of the mini project is to read the documentation of these three libraries, and apply all of them to the "German Credit" dataset, just like you did in the case of Decision Trees and Random Forests.

The final deliverable of this section should be a table (can be a pandas DataFrame) which shows the accuracy of all the five algorthms taught in this mini project in one place.

Happy modeling! :)

In [ ]:
```

```