In [2]:
```
import pandas as pd
import numpy as np
from sklearn.ensemble import *
from sklearn.cross_validation import *
from sklearn.preprocessing import *

wd = 'C:/Documents and Settings/e68321/Application Data/adult/data'
train = pd.io.parsers.read_csv(wd + '/adult.data')
test = pd.io.parsers.read_csv(wd + '/adult.test')
```

```
Some slight preprocessing of the data is required: remove whitespaces, unify the train and test class labels.
Although there is an already defined train/test split in the original dataset, we will stack both parts and
only consider the 3-fold cross-validation results over the full dataset from now on.
```

In [3]:
```
for c in train.columns.tolist():
    if train[c].dtype == object:
        train[c] = train[c].map(lambda s: s.strip())
        test[c] = test[c].map(lambda s: s.strip())
        if c == 'income':
            test[c] = test[c].map(lambda s: s[:-1]) # remove "."
        if c == 'native-country': continue
        assert set(train[c]) == set(test[c])

full = pd.concat((train, test))
print 'dataset size:', full.shape
```

```
dataset size: (48842, 15)
```

We can see that the features are a nice balance of numeric and categorical attributes.

In [3]:
```
for c in full.columns.tolist():
    is_categorical = full[c].dtype == object
    print '%s (%s): %s' % (c, 'categorical' if is_categorical  else 'numeric',
                           ', '.join(set(full[c])) if is_categorical else '%s to %s' % (min(full[c]), max(full[c
```

```
age (numeric): 17 to 90
workclass (categorical): Self-emp-inc, State-gov, Without-pay, Private, Local-gov, Self-emp-not-inc, Federal-g
ov, Never-worked, ?
fnlwgt (numeric): 12285 to 1490400
education (categorical): Masters, Prof-school, 12th, Assoc-voc, 1st-4th, Assoc-acdm, HS-grad, Bachelors, 9th,
5th-6th, Some-college, 11th, 10th, Doctorate, Preschool, 7th-8th
education-num (numeric): 1 to 16
marital-status (categorical): Separated, Widowed, Divorced, Married-spouse-absent, Never-married, Married-AF-s
pouse, Married-civ-spouse
occupation (categorical): Farming-fishing, Armed-Forces, Craft-repair, Other-service, Transport-moving, Prof-s
pecialty, Sales, Exec-managerial, Handlers-cleaners, ?, Adm-clerical, Protective-serv, Tech-support, Priv-hous
e-serv, Machine-op-inspct
relationship (categorical): Own-child, Wife, Unmarried, Other-relative, Husband, Not-in-family
race (categorical): Asian-Pac-Islander, Amer-Indian-Eskimo, White, Other, Black
sex (categorical): Male, Female
capital-gain (numeric): 0 to 99999
capital-loss (numeric): 0 to 4356
hours-per-week (numeric): 1 to 99
native-country (categorical): Canada, Hong, Dominican-Republic, Italy, Ireland, Outlying-US(Guam-USVI-etc), Sc
otland, Cambodia, France, Peru, Laos, Ecuador, Iran, Cuba, Guatemala, Germany, Thailand, Haiti, Poland, ?, Hol
and-Netherlands, Philippines, Vietnam, Hungary, England, South, Jamaica, Honduras, Portugal, Mexico, El-Salvad
or, India, Puerto-Rico, China, Yugoslavia, United-States, Trinadad&Tobago, Greece, Japan, Taiwan, Nicaragua, C
olumbia
income (categorical): <=50K, >50K
```

To be fed into the sklearn implementation of RFs, the categorical features must be numerically encoded.

In [3]:
```
for c in [c for c in full.columns.tolist() if full[c].dtype == object]:
    full[c] = LabelEncoder().fit_transform(full[c])
```

It's easy to match the ~85% accuracy reported in the dataset documentation, with a vanilla RF, and no hyperparameter tweaking whatsoever.

In [4]:
```
X = full.iloc[:,:-1] # everything except last column
y = full.iloc[:,-1]  # last column

clf = RandomForestClassifier()
print 'accuracy:', np.mean(cross_val_score(clf, X, y, scoring='accuracy'))
print 'AUC:', np.mean(cross_val_score(clf, X, y, scoring='roc_auc'))
```

```
accuracy: 0.850067533235
AUC: 0.882580780957
```

It's possible to slightly increase the performance with a higher number of trees.

In [65]:
```
clf = RandomForestClassifier(n_estimators=50)
print 'accuracy:', np.mean(cross_val_score(clf, X, y, scoring='accuracy'))
print 'AUC:', np.mean(cross_val_score(clf, X, y, scoring='roc_auc'))
```

```
accuracy: 0.854449031674
AUC: 0.900744507122
```

The CART algorithm (on which this particular implementation of RF rests) actually treats the categorical variables as ordinal: for a feature $x = \{a, b, c, d\}$ for instance, the possible (binary) splits can only be $\{a\}/\{b, c, d\}$, $\{a, b\}/\{c, d\}$ and $\{a, b, c\}/\{d\}$, because the values are implicitly ordered, meaning that $\{a, c\}/\{b, d\}$ would not be possible. In theory, one way to mitigate this is to use one-hot encoding.. but in practice, it doesn't really seem to help.

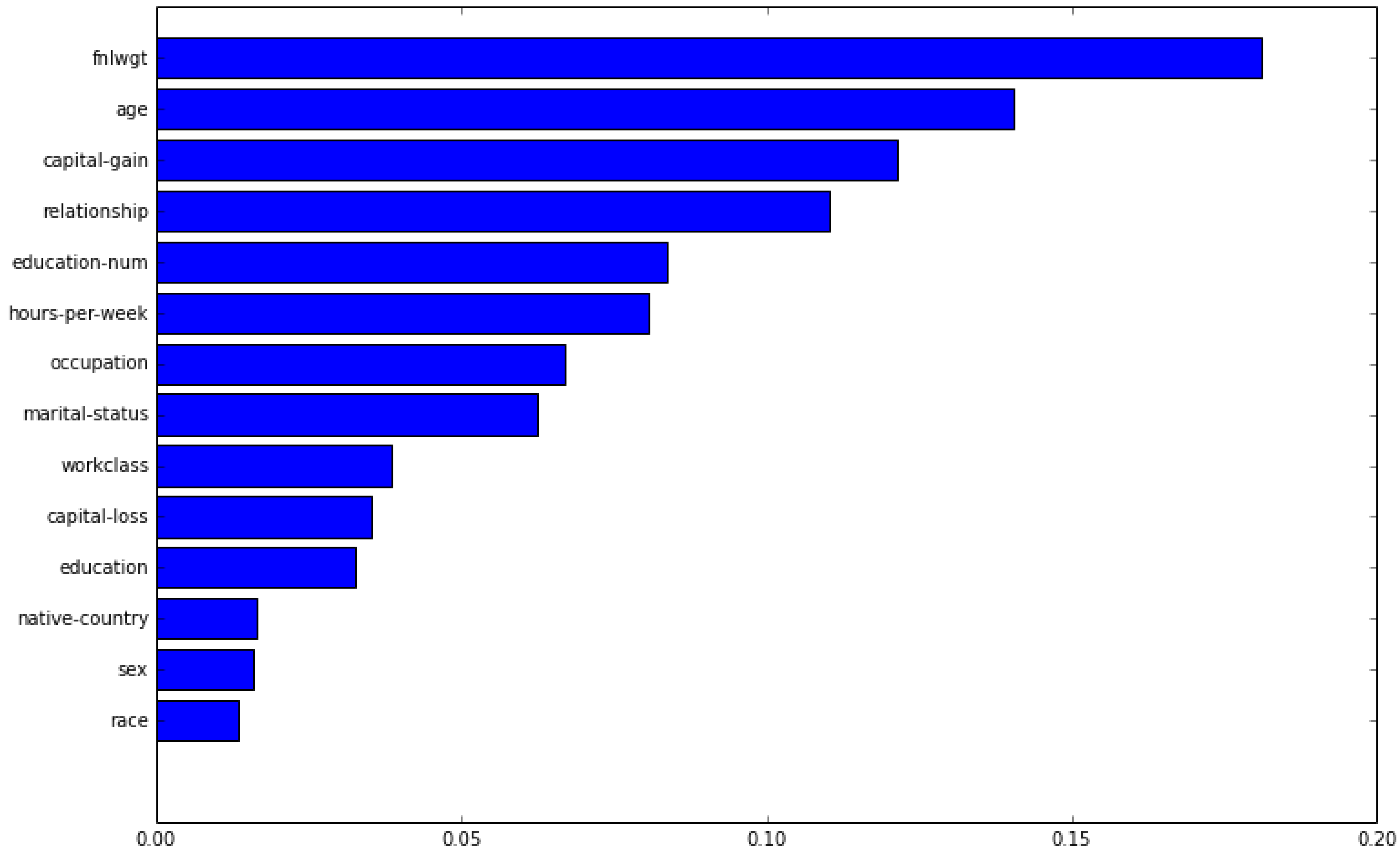
In [64]:
```
X = np.asarray(full)
ohe = OneHotEncoder(categorical_features=[i for i, c in enumerate(train.columns.tolist()[:-1]) if train[c].dtype
X = ohe.fit_transform(X[:,:-1]).todense()

print 'features:', X.shape
print 'accuracy:', np.mean(cross_val_score(clf, X, y, scoring='accuracy'))
print 'AUC:', np.mean(cross_val_score(clf, X, y, scoring='roc_auc'))
```

```
features: (48842, 108)
accuracy: 0.849330518015
AUC: 0.878599687512
```

A nice aspect of an RF model is that it provides a direct way to assess the importance of features (in terms of their predictive power).

In [19]:
```
clf = RandomForestClassifier()
X = full.iloc[:,:-1] # back to 15 features (i.e. non one-hot encoded)
clf.fit(X, y)
fi = clf.feature_importances_
fic = sorted(zip(fi, train.columns.tolist()[:-1]))
pylab.rcParams['figure.figsize'] = 12, 8
barh(range(len(fi)), [v for v, c in fic], align='center')
yticks(range(len(fi)), [c for v, c in fic]);
```



We can easily verify this by looking at the performance of models trained with the worst and best set of 4 features (the results are not striking though, for some reason).

In [31]:

```
clf = RandomForestClassifier(max_features=None) # the default is sqrt(n_features), here we want to use them all
worst = ['education', 'native-country', 'sex', 'race']
best = ['fnlwgt', 'age', 'capital-gain', 'relationship']
print 'accuracy (worst 4 features):', np.mean(cross_val_score(clf, X[worst], y, scoring='accuracy'))
print 'AUC (worst 4 features):', np.mean(cross_val_score(clf, X[worst], y, scoring='roc_auc'))
print 'accuracy (best 4 features):', np.mean(cross_val_score(clf, X[best], y, scoring='accuracy'))
print 'AUC (best 4 features):', np.mean(cross_val_score(clf, X[best], y, scoring='roc_auc'))
```

```
accuracy (worst 4 features): 0.785123450754
AUC (worst 4 features): 0.758914590442
accuracy (best 4 features): 0.792535113887
AUC (best 4 features): 0.800357087358
```

In [2]:
```
import pandas as pd
import numpy as np
from sklearn.ensemble import *
from sklearn.cross_validation import *
from sklearn.preprocessing import *

wd = 'C:/Documents and Settings/e68321/Application Data/adult/data'
train = pd.io.parsers.read_csv(wd + '/adult.data')
test = pd.io.parsers.read_csv(wd + '/adult.test')
```

```
Some slight preprocessing of the data is required: remove whitespaces, unify the train and test class labels.
Although there is an already defined train/test split in the original dataset, we will stack both parts and
only consider the 3-fold cross-validation results over the full dataset from now on.
```

In [3]:
```
for c in train.columns.tolist():
    if train[c].dtype == object:
        train[c] = train[c].map(lambda s: s.strip())
        test[c] = test[c].map(lambda s: s.strip())
        if c == 'income':
            test[c] = test[c].map(lambda s: s[:-1]) # remove "."
        if c == 'native-country': continue
        assert set(train[c]) == set(test[c])

full = pd.concat((train, test))
print 'dataset size:', full.shape
```

```
dataset size: (48842, 15)
```

We can see that the features are a nice balance of numeric and categorical attributes.

In [3]:
```
for c in full.columns.tolist():
    is_categorical = full[c].dtype == object
    print '%s (%s): %s' % (c, 'categorical' if is_categorical  else 'numeric',
                           ', '.join(set(full[c])) if is_categorical else '%s to %s' % (min(full[c]), max(full[c
```

```
age (numeric): 17 to 90
workclass (categorical): Self-emp-inc, State-gov, Without-pay, Private, Local-gov, Self-emp-not-inc, Federal-g
ov, Never-worked, ?
fnlwgt (numeric): 12285 to 1490400
education (categorical): Masters, Prof-school, 12th, Assoc-voc, 1st-4th, Assoc-acdm, HS-grad, Bachelors, 9th,
5th-6th, Some-college, 11th, 10th, Doctorate, Preschool, 7th-8th
education-num (numeric): 1 to 16
marital-status (categorical): Separated, Widowed, Divorced, Married-spouse-absent, Never-married, Married-AF-s
pouse, Married-civ-spouse
occupation (categorical): Farming-fishing, Armed-Forces, Craft-repair, Other-service, Transport-moving, Prof-s
pecialty, Sales, Exec-managerial, Handlers-cleaners, ?, Adm-clerical, Protective-serv, Tech-support, Priv-hous
e-serv, Machine-op-inspct
relationship (categorical): Own-child, Wife, Unmarried, Other-relative, Husband, Not-in-family
race (categorical): Asian-Pac-Islander, Amer-Indian-Eskimo, White, Other, Black
sex (categorical): Male, Female
capital-gain (numeric): 0 to 99999
capital-loss (numeric): 0 to 4356
hours-per-week (numeric): 1 to 99
native-country (categorical): Canada, Hong, Dominican-Republic, Italy, Ireland, Outlying-US(Guam-USVI-etc), Sc
otland, Cambodia, France, Peru, Laos, Ecuador, Iran, Cuba, Guatemala, Germany, Thailand, Haiti, Poland, ?, Hol
and-Netherlands, Philippines, Vietnam, Hungary, England, South, Jamaica, Honduras, Portugal, Mexico, El-Salvad
or, India, Puerto-Rico, China, Yugoslavia, United-States, Trinadad&Tobago, Greece, Japan, Taiwan, Nicaragua, C
olumbia
income (categorical): <=50K, >50K
```

To be fed into the sklearn implementation of RFs, the categorical features must be numerically encoded.

In [3]:
```
for c in [c for c in full.columns.tolist() if full[c].dtype == object]:
    full[c] = LabelEncoder().fit_transform(full[c])
```

It's easy to match the ~85% accuracy reported in the dataset documentation, with a vanilla RF, and no hyperparameter tweaking whatsoever.

In [4]:
```
X = full.iloc[:,:-1] # everything except last column
y = full.iloc[:,-1]  # last column

clf = RandomForestClassifier()
print 'accuracy:', np.mean(cross_val_score(clf, X, y, scoring='accuracy'))
print 'AUC:', np.mean(cross_val_score(clf, X, y, scoring='roc_auc'))
```

```
accuracy: 0.850067533235
AUC: 0.882580780957
```

It's possible to slightly increase the performance with a higher number of trees.

In [65]:
```
clf = RandomForestClassifier(n_estimators=50)
print 'accuracy:', np.mean(cross_val_score(clf, X, y, scoring='accuracy'))
print 'AUC:', np.mean(cross_val_score(clf, X, y, scoring='roc_auc'))
```

```
accuracy: 0.854449031674
AUC: 0.900744507122
```

The CART algorithm (on which this particular implementation of RF rests) actually treats the categorical variables as ordinal: for a feature $x = \{a, b, c, d\}$ for instance, the possible (binary) splits can only be $\{a\}/\{b, c, d\}$, $\{a, b\}/\{c, d\}$ and $\{a, b, c\}/\{d\}$, because the values are implicitly ordered, meaning that $\{a, c\}/\{b, d\}$ would not be possible. In theory, one way to mitigate this is to use one-hot encoding.. but in practice, it doesn't really seem to help.

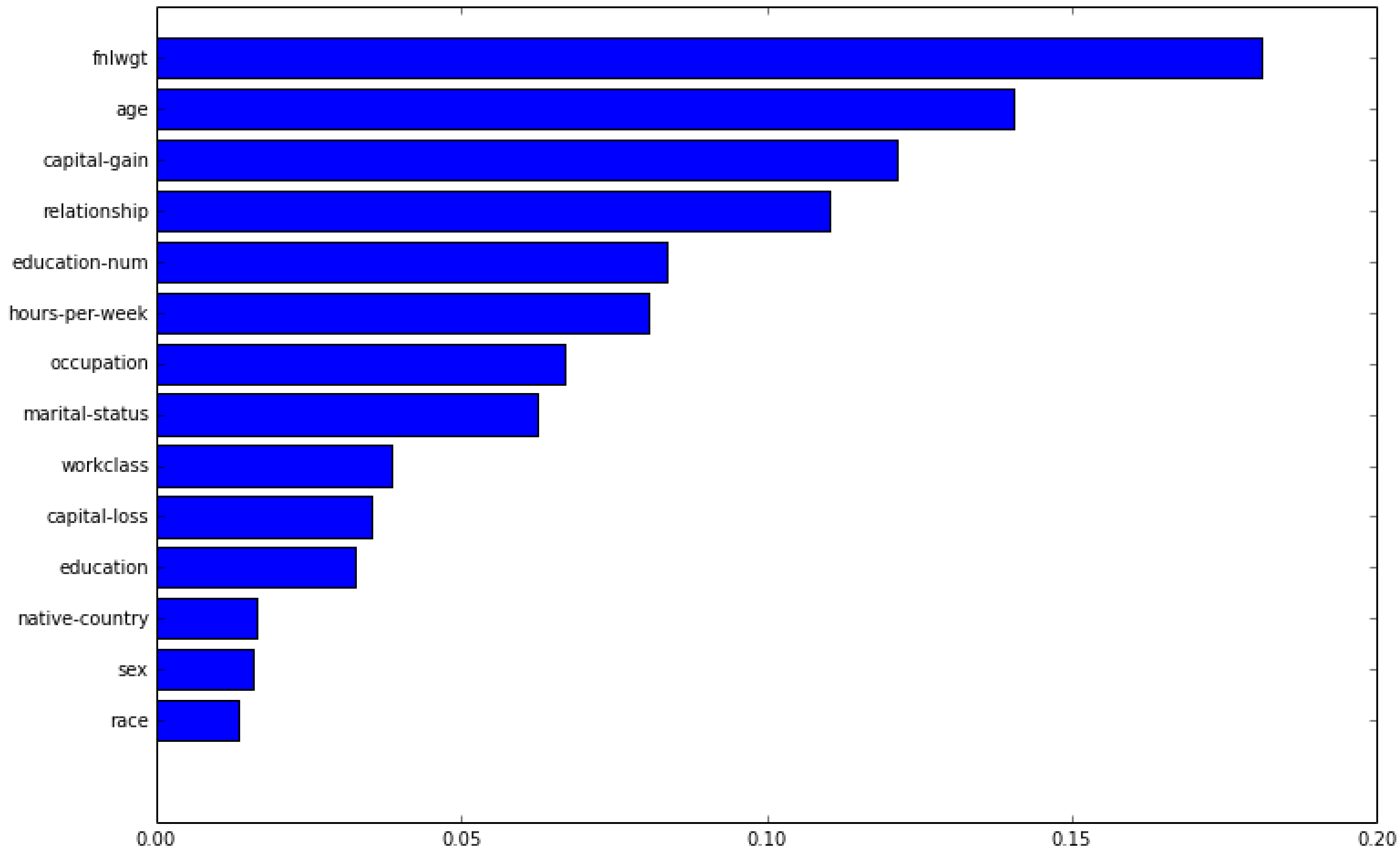
In [64]:
```
X = np.asarray(full)
ohe = OneHotEncoder(categorical_features=[i for i, c in enumerate(train.columns.tolist()[:-1]) if train[c].dtype
X = ohe.fit_transform(X[:,:-1]).todense()

print 'features:', X.shape
print 'accuracy:', np.mean(cross_val_score(clf, X, y, scoring='accuracy'))
print 'AUC:', np.mean(cross_val_score(clf, X, y, scoring='roc_auc'))
```

```
features: (48842, 108)
accuracy: 0.849330518015
AUC: 0.878599687512
```

A nice aspect of an RF model is that it provides a direct way to assess the importance of features (in terms of their predictive power).

In [19]:
```
clf = RandomForestClassifier()
X = full.iloc[:,:-1] # back to 15 features (i.e. non one-hot encoded)
clf.fit(X, y)
fi = clf.feature_importances_
fic = sorted(zip(fi, train.columns.tolist()[:-1]))
pylab.rcParams['figure.figsize'] = 12, 8
barh(range(len(fi)), [v for v, c in fic], align='center')
yticks(range(len(fi)), [c for v, c in fic]);
```



We can easily verify this by looking at the performance of models trained with the worst and best set of 4 features (the results are not striking though, for some reason).

In [31]:

```
clf = RandomForestClassifier(max_features=None) # the default is sqrt(n_features), here we want to use them all
worst = ['education', 'native-country', 'sex', 'race']
best = ['fnlwgt', 'age', 'capital-gain', 'relationship']
print 'accuracy (worst 4 features):', np.mean(cross_val_score(clf, X[worst], y, scoring='accuracy'))
print 'AUC (worst 4 features):', np.mean(cross_val_score(clf, X[worst], y, scoring='roc_auc'))
print 'accuracy (best 4 features):', np.mean(cross_val_score(clf, X[best], y, scoring='accuracy'))
print 'AUC (best 4 features):', np.mean(cross_val_score(clf, X[best], y, scoring='roc_auc'))
```

```
accuracy (worst 4 features): 0.785123450754
AUC (worst 4 features): 0.758914590442
accuracy (best 4 features): 0.792535113887
AUC (best 4 features): 0.800357087358
```

In [2]:
```
import pandas as pd
import numpy as np
from sklearn.ensemble import *
from sklearn.cross_validation import *
from sklearn.preprocessing import *

wd = 'C:/Documents and Settings/e68321/Application Data/adult/data'
train = pd.io.parsers.read_csv(wd + '/adult.data')
test = pd.io.parsers.read_csv(wd + '/adult.test')
```

```
Some slight preprocessing of the data is required: remove whitespaces, unify the train and test class labels.
Although there is an already defined train/test split in the original dataset, we will stack both parts and
only consider the 3-fold cross-validation results over the full dataset from now on.
```

In [3]:
```
for c in train.columns.tolist():
    if train[c].dtype == object:
        train[c] = train[c].map(lambda s: s.strip())
        test[c] = test[c].map(lambda s: s.strip())
        if c == 'income':
            test[c] = test[c].map(lambda s: s[:-1]) # remove "."
        if c == 'native-country': continue
        assert set(train[c]) == set(test[c])

full = pd.concat((train, test))
print 'dataset size:', full.shape
```

```
dataset size: (48842, 15)
```

We can see that the features are a nice balance of numeric and categorical attributes.

In [3]:
```
for c in full.columns.tolist():
    is_categorical = full[c].dtype == object
    print '%s (%s): %s' % (c, 'categorical' if is_categorical  else 'numeric',
                           ', '.join(set(full[c])) if is_categorical else '%s to %s' % (min(full[c]), max(full[c
```

```
age (numeric): 17 to 90
workclass (categorical): Self-emp-inc, State-gov, Without-pay, Private, Local-gov, Self-emp-not-inc, Federal-g
ov, Never-worked, ?
fnlwgt (numeric): 12285 to 1490400
education (categorical): Masters, Prof-school, 12th, Assoc-voc, 1st-4th, Assoc-acdm, HS-grad, Bachelors, 9th,
5th-6th, Some-college, 11th, 10th, Doctorate, Preschool, 7th-8th
education-num (numeric): 1 to 16
marital-status (categorical): Separated, Widowed, Divorced, Married-spouse-absent, Never-married, Married-AF-s
pouse, Married-civ-spouse
occupation (categorical): Farming-fishing, Armed-Forces, Craft-repair, Other-service, Transport-moving, Prof-s
pecialty, Sales, Exec-managerial, Handlers-cleaners, ?, Adm-clerical, Protective-serv, Tech-support, Priv-hous
e-serv, Machine-op-inspct
relationship (categorical): Own-child, Wife, Unmarried, Other-relative, Husband, Not-in-family
race (categorical): Asian-Pac-Islander, Amer-Indian-Eskimo, White, Other, Black
sex (categorical): Male, Female
capital-gain (numeric): 0 to 99999
capital-loss (numeric): 0 to 4356
hours-per-week (numeric): 1 to 99
native-country (categorical): Canada, Hong, Dominican-Republic, Italy, Ireland, Outlying-US(Guam-USVI-etc), Sc
otland, Cambodia, France, Peru, Laos, Ecuador, Iran, Cuba, Guatemala, Germany, Thailand, Haiti, Poland, ?, Hol
and-Netherlands, Philippines, Vietnam, Hungary, England, South, Jamaica, Honduras, Portugal, Mexico, El-Salvad
or, India, Puerto-Rico, China, Yugoslavia, United-States, Trinadad&Tobago, Greece, Japan, Taiwan, Nicaragua, C
olumbia
income (categorical): <=50K, >50K
```

To be fed into the sklearn implementation of RFs, the categorical features must be numerically encoded.

In [3]:
```
for c in [c for c in full.columns.tolist() if full[c].dtype == object]:
    full[c] = LabelEncoder().fit_transform(full[c])
```

It's easy to match the ~85% accuracy reported in the dataset documentation, with a vanilla RF, and no hyperparameter tweaking whatsoever.

In [4]:
```
X = full.iloc[:,:-1] # everything except last column
y = full.iloc[:,-1]  # last column

clf = RandomForestClassifier()
print 'accuracy:', np.mean(cross_val_score(clf, X, y, scoring='accuracy'))
print 'AUC:', np.mean(cross_val_score(clf, X, y, scoring='roc_auc'))
```

```
accuracy: 0.850067533235
AUC: 0.882580780957
```

It's possible to slightly increase the performance with a higher number of trees.

In [65]:
```
clf = RandomForestClassifier(n_estimators=50)
print 'accuracy:', np.mean(cross_val_score(clf, X, y, scoring='accuracy'))
print 'AUC:', np.mean(cross_val_score(clf, X, y, scoring='roc_auc'))
```

```
accuracy: 0.854449031674
AUC: 0.900744507122
```

The CART algorithm (on which this particular implementation of RF rests) actually treats the categorical variables as ordinal: for a feature $x = \{a, b, c, d\}$ for instance, the possible (binary) splits can only be $\{a\}/\{b, c, d\}$, $\{a, b\}/\{c, d\}$ and $\{a, b, c\}/\{d\}$, because the values are implicitly ordered, meaning that $\{a, c\}/\{b, d\}$ would not be possible. In theory, one way to mitigate this is to use one-hot encoding.. but in practice, it doesn't really seem to help.

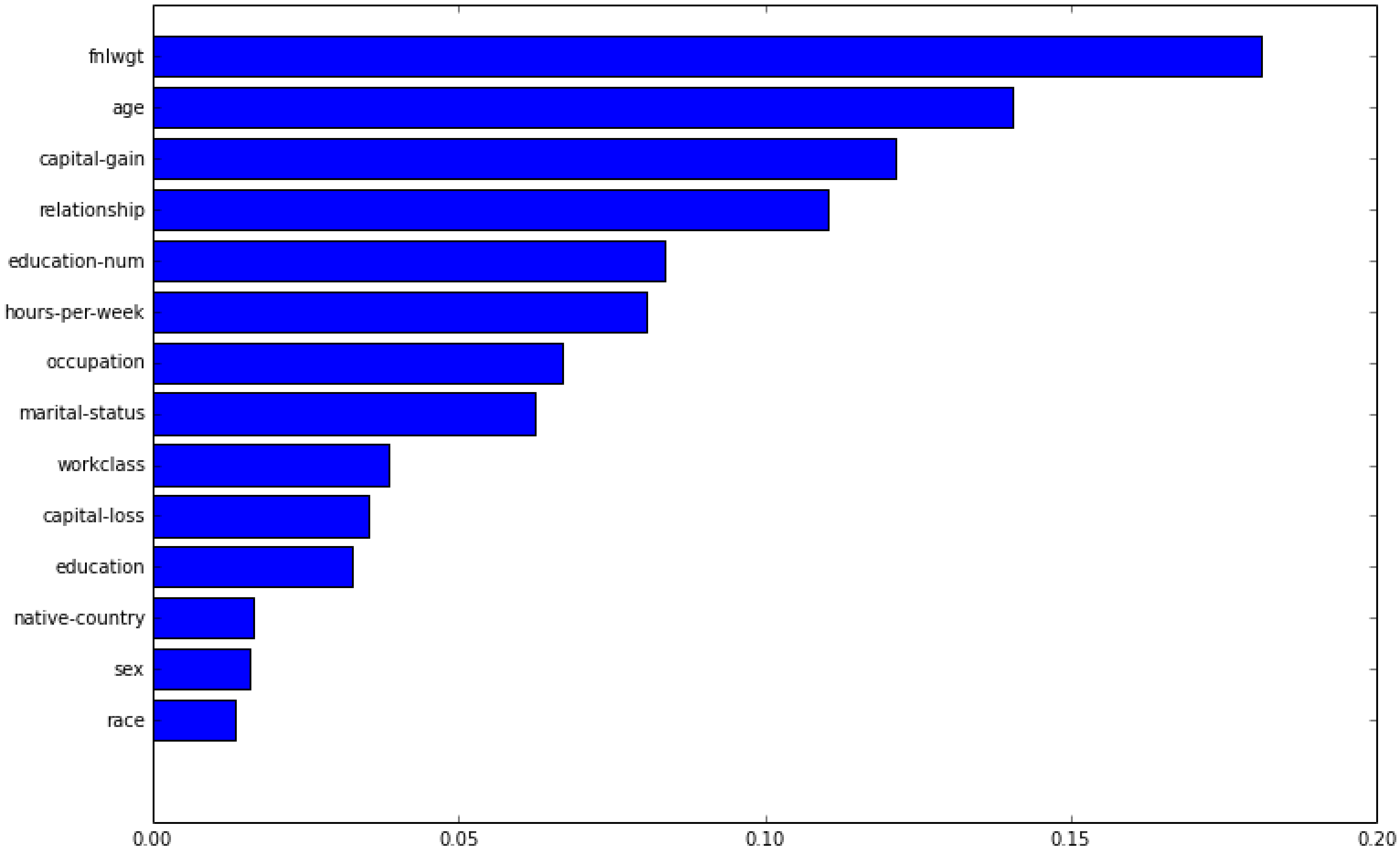
In [64]:
```
X = np.asarray(full)
ohe = OneHotEncoder(categorical_features=[i for i, c in enumerate(train.columns.tolist()[:-1]) if train[c].dtype
X = ohe.fit_transform(X[:,:-1]).todense()

print 'features:', X.shape
print 'accuracy:', np.mean(cross_val_score(clf, X, y, scoring='accuracy'))
print 'AUC:', np.mean(cross_val_score(clf, X, y, scoring='roc_auc'))
```

```
features: (48842, 108)
accuracy: 0.849330518015
AUC: 0.878599687512
```

A nice aspect of an RF model is that it provides a direct way to assess the importance of features (in terms of their predictive power).

In [19]:
```
clf = RandomForestClassifier()
X = full.iloc[:,:-1] # back to 15 features (i.e. non one-hot encoded)
clf.fit(X, y)
fi = clf.feature_importances_
fic = sorted(zip(fi, train.columns.tolist()[:-1]))
pylab.rcParams['figure.figsize'] = 12, 8
barh(range(len(fi)), [v for v, c in fic], align='center')
yticks(range(len(fi)), [c for v, c in fic]);
```



We can easily verify this by looking at the performance of models trained with the worst and best set of 4 features (the results are not striking though, for some reason).

In [31]:
```
clf = RandomForestClassifier(max_features=None) # the default is sqrt(n_features), here we want to use them all
worst = ['education', 'native-country', 'sex', 'race']
best = ['fnlwgt', 'age', 'capital-gain', 'relationship']
print 'accuracy (worst 4 features):', np.mean(cross_val_score(clf, X[worst], y, scoring='accuracy'))
print 'AUC (worst 4 features):', np.mean(cross_val_score(clf, X[worst], y, scoring='roc_auc'))
print 'accuracy (best 4 features):', np.mean(cross_val_score(clf, X[best], y, scoring='accuracy'))
print 'AUC (best 4 features):', np.mean(cross_val_score(clf, X[best], y, scoring='roc_auc'))
```

```
accuracy (worst 4 features): 0.785123450754
AUC (worst 4 features): 0.758914590442
accuracy (best 4 features): 0.792535113887
AUC (best 4 features): 0.800357087358
```