COMPSCI 671 Machine Learning

HW 2

Yiteng Lu

1. Maximum Entropy Configuration

1.a

For coin 1: $H(\mathbf{p}) = -(0.5 \cdot \log_2 0.5 + 0.5 \cdot \log_2 0.5) = 1$
For coin 2: $H(\mathbf{p}) = -(0.7 \cdot \log_2 0.7 + 0.3 \cdot \log_2 0.3) = 0.8813$
For coin 3: $H(\mathbf{p}) = -(0.1 \cdot \log_2 0.1 + 0.9 \cdot \log_2 0.9) = 0.4690$

So, the first coin has the highest entropy.

1.b

Given $H(\mathbf{p}) = -\sum_{i=1}^{n} P(x)\log_2(P(x))$

$= \mathbb{E}\left[\log_2(1/P(X))\right]$

$\log_2 x$ is a concave function, then $-\log_2 x$ is a convex function:

Proof:

$$\frac{d^2}{dx^2}(-\log_2 x) = \frac{1}{x^2 \log(2)} > 0$$

According to Jensen's Inequality:

$$-\log_2(\mathbb{E}[X]) \leq \mathbb{E}\left[-\log_2(X)\right] \implies \mathbb{E}\left[\log_2(X)\right] \leq \log_2(\mathbb{E}[X])$$

then, $H(\mathbf{p}) = \mathbb{E}\left[\log_2(1/P(X))\right] \leq \log_2(\mathbb{E}[1/P(X)])$

$$= \log_2 \sum_{i=1}^{n} P(x_i) * 1/P(x_i) = \log_2 \sum_{i=1}^{n} 1 = \log_2 n$$

Therefore, the maximum entropy will be find at $H(\mathbf{p}) = \log n$

1.c

$$H(g(x)) = -\int_{-\infty}^{+\infty} g(x)log(g(x))dx$$

$$= -\int_{-\infty}^{+\infty} \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{(x-\mu)^2}{2\sigma^2}} \cdot log(\frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{(x-\mu)^2}{2\sigma^2}})dx$$

$$= -\int_{-\infty}^{+\infty} \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{(x-\mu)^2}{2\sigma^2}} \cdot (log(\frac{1}{\sqrt{2\pi\sigma^2}}) + log(e^{-\frac{(x-\mu)^2}{2\sigma^2}}))dx$$

$$\int_{-\infty}^{+\infty} \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{(x-\mu)^2}{2\sigma^2}} \cdot log(\frac{1}{\sqrt{2\pi\sigma^2}}) + \int_{-\infty}^{+\infty} \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{(x-\mu)^2}{2\sigma^2}} \cdot log(e^{-\frac{(x-\mu)^2}{2\sigma^2}})$$

$$= -\int_{-\infty} \frac{}{\sqrt{2\pi\sigma^2}} e^{\ 2\sigma^2} \cdot \iota og \frac{}{\sqrt{2\pi\sigma^2}} ax - \int_{-\infty} \frac{}{\sqrt{2\pi\sigma^2}} e^{\ 2\sigma^2} \cdot \iota og(e^{\ 2\sigma^2}) ax$$

$$= -log \frac{1}{\sqrt{2\pi\sigma^2}} \int_{-\infty}^{+\infty} \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{(x-\mu)^2}{2\sigma^2}} dx - \int_{-\infty}^{+\infty} -\frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{(x-\mu)^2}{2\sigma^2}} \frac{(x-\mu)^2}{2\sigma^2} dx$$

$$= -log \frac{1}{\sqrt{2\pi\sigma^2}} (1) + \frac{1}{2\sigma^2} \int_{-\infty}^{+\infty} \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{(x-\mu)^2}{2\sigma^2}} (x-\mu)^2 dx$$

$$= -log \frac{1}{\sqrt{2\pi\sigma^2}} + \frac{1}{2\sigma^2} E[(x-\mu)^2]$$

$$= -log \frac{1}{\sqrt{2\pi\sigma^2}} + \frac{1}{2\sigma^2} \mathrm{Var}[x]$$

$$= -log \frac{1}{\sqrt{2\pi\sigma^2}} + \frac{1}{2\sigma^2} \cdot \sigma^2$$

$$= -log \frac{1}{\sqrt{2\pi\sigma^2}} + \frac{1}{2}$$

In addition:

$$
\begin{aligned}
-\int_{-\infty}^{\infty} f(x)log(g(x))dx &= -\int_{-\infty}^{\infty} f(x)log(\frac{1}{\sqrt{2\pi\sigma^2}}e^{-\frac{(x-\mu)^2}{2\sigma^2}})dx \\
&= -\int_{-\infty}^{\infty} f(x)(log(\frac{1}{\sqrt{2\pi\sigma^2}}) + log(e^{-\frac{(x-\mu)^2}{2\sigma^2}}))dx \\
&= -\int_{-\infty}^{+\infty} f(x)\cdot log\frac{1}{\sqrt{2\pi\sigma^2}}dx - \int_{-\infty}^{+\infty} f(x)\cdot log(e^{-\frac{(x-\mu)^2}{2\sigma^2}})dx \\
&= -log\frac{1}{\sqrt{2\pi\sigma^2}}\int_{-\infty}^{+\infty} f(x)dx - \int_{-\infty}^{+\infty} -f(x)\frac{(x-\mu)^2}{2\sigma^2}dx \\
&= -log\frac{1}{\sqrt{2\pi\sigma^2}}(1) + \frac{1}{2\sigma^2}\int_{-\infty}^{+\infty} f(x)(x-\mu)^2dx \\
&= -log\frac{1}{\sqrt{2\pi\sigma^2}} + \frac{1}{2\sigma^2}E[(x-\mu)^2] \\
&= -log\frac{1}{\sqrt{2\pi\sigma^2}} + \frac{1}{2\sigma^2}\mathrm{Var}[x] \\
&= -log\frac{1}{\sqrt{2\pi\sigma^2}} + \frac{1}{2\sigma^2}\sigma^2 \\
&= -log\frac{1}{\sqrt{2\pi\sigma^2}} + \frac{1}{2}
\end{aligned}
$$

Therefore:

$H(g(x)) = -\int_{-\infty}^{\infty} f(x)log(g(x))dx = -log\frac{1}{\sqrt{2\pi\sigma^2}} + \frac{1}{2}$

$$
\begin{aligned}
H(g(x)) - H(f(x)) &= -\int_{-\infty}^{\infty} f(x)log(g(x))dx + \int_{-\infty}^{\infty} f(x)log(f(x))dx \\
&= -(\int_{-\infty}^{\infty} f(x)log(g(x))dx - \int_{-\infty}^{\infty} f(x)log(f(x))dx) \\
&= -(\int_{-\infty}^{\infty} (f(x)log(g(x)) - f(x)log(f(x))dx) \\
&= \int_{-\infty}^{\infty} f(x)(-log\frac{g(x)}{f(x)})dx
\end{aligned}
$$

As proved in 1.b, $\phi(x) = -logx$ is a convex function and we can use the Jensen's inequality:

$$\int_{-\infty}^{\infty} f(x)(-log\frac{g(x)}{f(x)})dx \geq -log(\int_{-\infty}^{\infty} f(x)\frac{g(x)}{f(x)}dx) \geq -log(\int_{-\infty}^{\infty} g(x)dx) \geq -log(1)$$
$$\geq 0$$
$$\Rightarrow H(g(x)) \geq H(f(x))$$

Therefore,Gaussian distribution gives the maximum differential entropy among all arbitrary probability density functions $f(x)$

2. Bias and Variance

2.a

$$
\begin{aligned}
E[(y - f\hat{(}x))^2] &= E[(y - E[f\hat{(}x)] + E[f\hat{(}x)] - f\hat{(}x))^2] \\
&= E[(y - E[f\hat{(}x)])^2 + (E[f\hat{(}x)] - f\hat{(}x))^2 + 2(y - E[f\hat{(}x)]) \cdot (E[f\hat{(} \\
&= E[(y - E[f\hat{(}x)])^2] + E[(E[f\hat{(}x)] - f\hat{(}x))^2] + 2E[y - E[f \\
&\qquad \cdot E[E[f\hat{(}x)] - f\hat{(}x)] \\
&= E[(y - E[f\hat{(}x)])^2] + E[(E[f\hat{(}x)] - f\hat{(}x))^2] + 2E[y - E[f \\
&\qquad \cdot (E[f\hat{(}x)] - E[f\hat{(}x)]) \\
&= E[(y - E[f\hat{(}x)])^2] + E[(E[f\hat{(}x) - f\hat{(}x))^2] \\
&= \underbrace{E[(y - E[f\hat{(}x)])^2]}_{\Downarrow} + \mathrm{Var}[f\hat{(}x)] \qquad \text{(for Var[} \\
&\qquad = E[(f\hat{(}x) - E[f\hat{(}x)])^2]) \\
&\quad E[(y - f(x) + f(x) - E[f\hat{(}x)])^2] \\
&= E[(y - f(x))^2 + (f(x) - E[f\hat{(}x)])^2 + 2(y - f(x)) \cdot (f(x) - E[f\acute{(} \\
&= E[(y - f(x))^2] + E[(f(x) - E[f\hat{(}x)])^2] + 2\underbrace{E[y - f(x)]}_{E[\epsilon]=0} \cdot E[f(x) \\
&= E[(y - f(x))^2] + E[(f(x) - E[f\hat{(}x)])^2] \\
&\qquad = E[\epsilon^2] \\
&\quad + (f(x) - E[f\hat{(}x)])^2 \qquad \text{(for } f(x) \text{ and } E[f\hat{(}x)] \text{ are dete} \\
&\qquad = \mathrm{Var}[\epsilon] + (E[\epsilon])^2 \\
&\qquad + (E[f(x) - E[f\hat{(}x)])^2 \qquad \text{(for } f(x) \text{ is determini} \\
&= \sigma^2 + 0 + (E[f\hat{(}x)] - E[f(x)])^2 \\
&= \sigma^2 + (E[f\hat{(}x) - f(x)])^2 \\
&= \sigma^2 + (\mathrm{Bias}[f\hat{(}x)])^2
\end{aligned}
$$

Therefore, $E[(y - f\hat{(}x))^2] = (\mathrm{Bias}[f\hat{(}x)])^2 + \mathrm{Var}[f\hat{(}x)] + \sigma^2$

2.b

Assume n = 1, x_1 forms up a tree with only one level on depth because no matter what spliting algorithm we use for growing the decision tree, once the node only contains one class, the spliting will terminate; therefore, since only one data point exists, the very first feature we consider will contain the only data point which will also determine the resultant label of this node -- the label of x_1.In such case, when we test on the trainning, x_1 will fall into the node based on its first feature and get correctly labeled, thus the training error is zero.

Things become different when n =2, if x_2 has the same feature values as x_1, it will eventually end up into the same leaf node as x_1, as metioned in the problem prompt, if two data points have the exactly same features, they will have the same kind of label (no collision). On the other hand, if x_2 has one or more different feature values, the tree starts to split and grow given the values of x_2. x_2 will has its own path of features and generate a different leaf node which contains itself. When we test on the training set, we will find x_2 will either goes into the same leaf node that x_1 goes or it will go into a different leaf node will generated by itself. If x_2 goes into the same leaf node as x_1 (given x_2 has all same feature values as x_1), it will be predicted as the same label as x_1, in such case the training error is zero. If it goes into the node generated by itself (given different feature values as x_1), it will be predicted the label in the leaf node it created itself. In such case, it will be predicted into the label which x_2 places itself at the time the tree created and of course it will get the correct label, the training error is zero.

After talking about these two base cases, assume n= k, the training error is zero. when n = k+1, if x_{k+1} has the same feature values as one (or more) data point(s) as before, x_{k+1} would follow the same feature path to the leaf node which contains all the same feature values as x_{k+1}. Since when n = k, these data points in the that particluar leaf node are correctly classified and x_{k+1} must have the same label with them because they share all the feature values, x_{k+1} will also be correctly classified. If x_{k+1} has one or more different feature value(s) from those points from x_1 to x_k, x_{k+1} would end up into a leaf node created by itself. In that case, when test on x_1 to x_{k+1} the first k Xs would be correctly classified as we assume there are no training error, and x_{k+1} would be end up into the correct leaf node as well following along the feature path, in that condition, the training error is zero.

Therefore, by induction, we can tell there always existing a decision for data set with no data point colliding has 0 training error.

2.c

Given:
X_i has variance: $\mathrm{Var}[X_i] = \sigma^2$, For i from $1, 2, \ldots N$
X_i and X_j has correlation ρ, For $i \neq j$
$\bar{X} = \sum_{i=1}^{N} X_i$, For i from $1, 2, \ldots N$
Looking for $\mathrm{Var}[\bar{X}]$.

Derivation:

$$\begin{aligned}\mathrm{Var}[\bar{X}] &= \mathrm{Var}\left[\frac{\sum_{i=1}^{N} X_i}{N}\right] \\ &= \frac{1}{N^2}\mathrm{Var}\left[\sum_{i=1}^{N} X_i\right] \\ &= \frac{1}{N^2}\left(\sum_{i=1}^{N}\mathrm{Var}[X_i] + 2\sum_{i\neq j}\mathrm{Cov}(X_i, X_j)\right)\end{aligned}$$

$$\mathrm{Cov}(X_i, X_j) = \rho\sqrt{Var[X_i]Var[X_j]} = \rho\sigma^2$$

$$\sum_{i\neq j}\mathrm{Cov}(X_i, X_j) = \binom{N}{2}\rho\sigma^2 = \frac{N!}{2!(N-2)!}\rho\sigma^2 = \frac{N(N-1)}{2}\rho\sigma^2$$

$$\begin{aligned}\mathrm{Var}[\bar{X}] &= \frac{1}{N^2}\left(\sum_{i=1}^{N}\mathrm{Var}[X_i] + 2\sum_{i\neq j}\mathrm{Cov}(X_i, X_j)\right) \\ &= \frac{1}{N^2}\left(N\sigma^2 + 2\cdot\frac{N(N-1)}{2}\rho\sigma^2\right) \\ &= \frac{\sigma^2}{N} + \frac{N-1}{N}\rho\sigma^2\end{aligned}$$

2.d

Insteading of using the entire data set as training set, Bagging will only use a certain percent of the overall data to train the model but in mupltile times, in such case, the variance in each subset of data will be smaller the variance of the entire sample data set. By doing so, when the sample set has noise inside it, using baggging would help to avoid overfitting the noisy data and gain more accuracy for the random forest which averaged the trained base models (decision tree).
Just like bagging for the sample set, using random subspaces would make the features randomly sampled which will reduce the correlation if we just train them on the random features instead of the entire feature set. By Looking into the equation derived from 2.c, the using small sample set would have smaller variance in between the data points and also using random subspace would reduce the correlation ρ, therefore the variance of the averaged models would have very small variance.
By using decision tree, the nonlinear relationships between the variables will have less influence on the overall performance. In the most situation, the preprocess of data is not that important, we do not need to standardize the data first and construct the tree. Decision trees are easy to implement and understand; it classifies the data according to their attributes, then people could explicitly see how the classification works.

3. Variable Importance of Recidivism Predicction

3.a

In [305]:
```
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import f1_score
import sys
import warnings
if not sys.warnoptions:
    warnings.simplefilter("ignore")
data = pd.read_csv('ProPublica_COMPAS_preprocessed.csv')
feature_list = list(pd.read_csv('ProPublica_COMPAS_preprocessed.csv',nrows=0
))
feature_list.remove(feature_list[0])
feature_list.remove(feature_list[-1])
data1 = data.drop(data.columns[0], axis=1).values
training,testing = train_test_split(data1, test_size=0.2)  # 4/5 train 1/5 f
or test
```

In [306]:
```
# RANDOM FOREST
rf = RandomForestClassifier()
# train the model
rf.fit(training[:,:-1],training[:,-1:].ravel())
# predict the labels and report accuracy
hard_pred = rf.predict(testing[:,:-1])
acc = np.isclose(hard_pred,testing[:,-1:].ravel()).sum()/len(hard_pred)
print("Test Accuracy: {}".format(acc))
f1 = f1_score(testing[:,-1:],hard_pred)
print('F1 score is:', f1)
```

```
Test Accuracy: 0.650087260034904
F1 score is: 0.4328147100424328
```

3.b

In [307]:

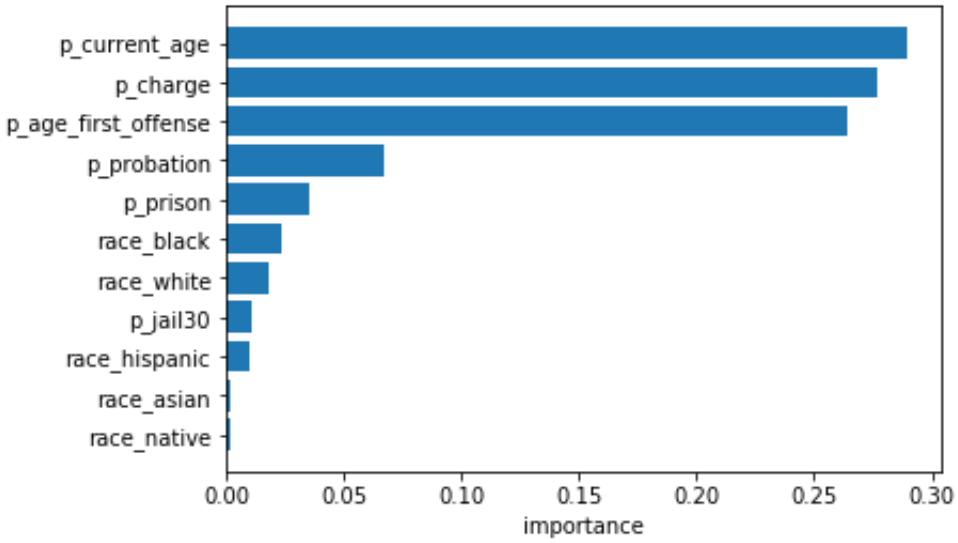
```
import operator
import matplotlib.pyplot as plt
importance = rf.feature_importances_
rank_dic={}
for i, element in enumerate (importance):
    rank_dic[element] = feature_list[i]
imp_dic = {}
imp_dic['age']  = importance[0]     ## the age importance
imp_dic['criminal_history'] = sum(importance[1:6])  ## add up all the crimin
al history importance
imp_dic['race'] = sum(importance[6:])   ## add up all the race importance
print('feature importance are:')
print(importance)
print()
for (key, value) in imp_dic.items():
    if value == max(imp_dic.values()):
        print("The feature has the greatest importance among 'age', 'crimina
l history','race' is" , key)
print()
print("relative_importance of race:{}" .format(imp_dic['race']))
print()
sorted_rank_dic = sorted(rank_dic.items(), key=lambda x: x[0],reverse=True)
print("The ranking of feature importance:")
ranked_list = []
for elements in sorted_rank_dic:
    ranked_list.append(elements[1])
    print(elements[1])
ranked_list.reverse()
y_pos = np.arange(len(feature_list))
importance_bar = np.sort(importance)
plt.barh(y_pos, importance_bar,  align='center')
plt.yticks(y_pos, ranked_list)
plt.xlabel('importance')
plt.show()
```

```
feature importance are:
[0.28932    0.26412337 0.27678141 0.01065174 0.03553274 0.06729952
 0.02366377 0.01860876 0.00993601 0.00211651 0.00196616]

The feature has the greatest importance among 'age', 'criminal history',
'race' is criminal_history

relative_importance of race:0.05629121040695936

The ranking of feature importance:
p_current_age
p_charge
p_age_first_offense
p_probation
p_prison
race_black
race_white
p_jail30
race_hispanic
race_asian
race_native
```



According to the data we printed out and plotted out, we can see the race is not that important. First of all, the race only have 0.0562 variable importance given the feature of age and criminal history when we add the related features together. From the plot, we can also tell that most of the race features are ranked in the later half among all the features.

3.c

•Remark: the two feature deleted with highest importance are 'p_charge' and 'p_current_age' in the first part of the problem

In [308]:
```
training2 = training[:]
testing2 = testing[:]
training2= np.delete(training2,0,1)
training2 = np.delete(training2,1,1)
testing2 =np.delete(testing2,0,1)
testing2 =np.delete(testing2,1,1)
rf.fit(training2[:,:-1],training2[:,-1:].ravel())
hard_pred2 = rf.predict(testing2[:,:-1])
acc2 = np.isclose(hard_pred2,testing2[:,-1:].ravel()).sum()/len(hard_pred2)
print("Accuracy: {}".format(acc2))
```

```
Accuracy: 0.6465968586387435
```

In [309]:
```
importance2 = rf.feature_importances_
imp_dic2 = {}
imp_dic2['criminal_history'] = sum(importance2[:4])
imp_dic2['race'] = sum(importance2[4:])
for (key, value) in imp_dic2.items():
    if value == max(imp_dic2.values()):
        print("The feature has the greatest importance among 'age', 'crimina
l history','race' is",key)
print()
print("relative_importance of race:{}" .format(imp_dic2['race']))
```

```
The feature has the greatest importance among 'age', 'criminal history',
'race' is criminal_history

relative_importance of race:0.08134021187440726
```

Once we removed the two featuers with highest importance, the relative importance for race increased.
•Remark: In the second part of the problem , I remove the feature 'p_charge'

In [310]:
```
training3= training[:]
testing3=testing[:]
training3=np.delete(training3,1,1)
testing3=np.delete(testing3,1,1)
rf.fit(training3[:,:-1],training3[:,-1:].ravel())
hard_pred3=rf.predict(testing3[:,:-1])
acc3 = np.isclose(hard_pred3,testing3[:,-1:].ravel()).sum()/len(hard_pred)
print("Accuracy: {}".format(acc3))
```

```
Accuracy: 0.6326352530541012
```

In [311]:
```
importance3 = rf.feature_importances_
imp_dic3 = {}
imp_dic3['age'] = importance3[0]
imp_dic3['criminal_history'] = sum(importance3[1:5])
imp_dic3['race'] = sum(importance3[5:])
for (key, value) in imp_dic3.items():
    if value == max(imp_dic3.values()):
        print("The feature has the greatest importance among 'age', 'crimina
l history','race' is",key)
print()
print("relative_importance of race:{}" .format(imp_dic3['race']/sum(imp_dic2
.values())))
```

```
The feature has the greatest importance among 'age', 'criminal history',
'race' is criminal_history

relative_importance of race:0.04837107940784228
```

The importance of race does not have much change but it decreased once we only delete one of the two most important variables compared with the very first one

3.d

In [312]:
```
from sklearn.linear_model import LogisticRegression
from sklearn.preprocessing import StandardScaler
lr1 = LogisticRegression(solver='lbfgs',max_iter=10000)
# train the model
lr1.fit(training[:,:-1],training[:,-1:].ravel())
coeff_before = lr.coef_[0]
```

```
print('coefficient of features (before standardized):')
print(coeff_before)
print()
sc = StandardScaler()
train_std = sc.fit_transform(training)
lr2 = LogisticRegression(solver='lbfgs',max_iter=10000)
lr2.fit(train_std[:,:-1],training[:,-1:].ravel())
coeff_after = lr2.coef_[0]
print('coefficient of features(after standardized):')
print(coeff_after)
print()
rank_coef_b = {}
for k, sub_feature in enumerate (coeff_before):
    rank_coef_b[sub_feature] = feature_list[k]
sorted_rank_coef_b = sorted(rank_coef_b.items(), key=lambda x: abs(x[0]),rev
erse = True)
ranked_coeff_before = []
print("ranking of coefficient of features (before standardized):")
for elements in sorted_rank_coef_b:
    ranked_coeff_before.append(elements[1])
    print(elements[1])
ranked_coeff_a=[]
rank_coef_a={}
for j, sub_feature in enumerate (coeff_after):
    rank_coef_a[sub_feature] = feature_list[j]
sorted_rank_coef_a = sorted(rank_coef_a.items(), key=lambda x: abs(x[0]),rev
erse = True)
ranked_coeff_after = []
print()
print("ranking of coefficient of features (after standardized):")
for elements in sorted_rank_coef_a:
    ranked_coeff_after.append(elements[1])
    print(elements[1])


coeff_before =abs(coeff_before)
coeff_before_sorted = sorted(coeff_before)
ranked_coeff_before.reverse()
plt.barh(y_pos, coeff_before_sorted,  align='center')
plt.yticks(y_pos, ranked_coeff_before)
plt.xlabel('coeffient')
plt.title('coefficient of features (before)')
plt.show()
coeff_after =abs(coeff_after)
coeff_after_sorted = sorted(coeff_after)
```

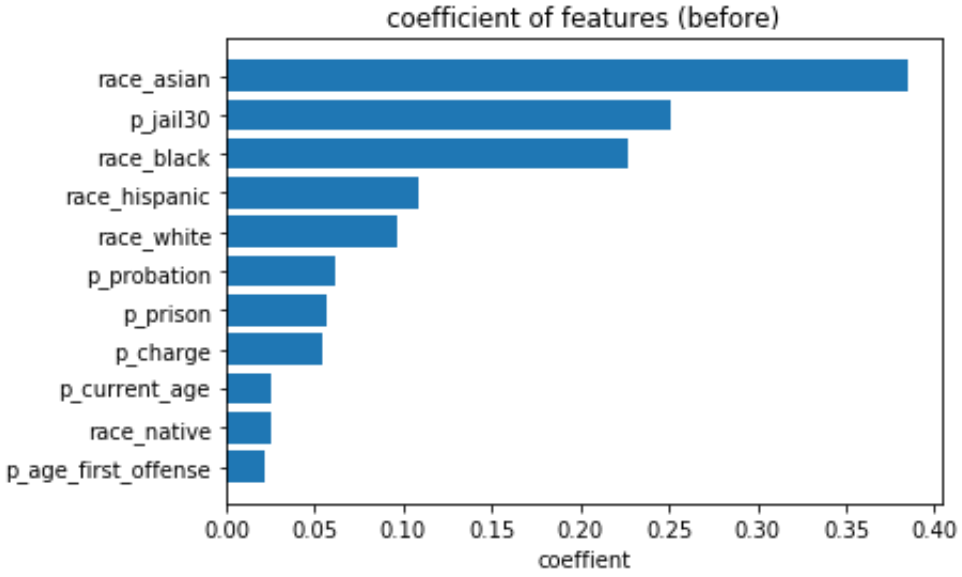
```
ranked_coeff_after.reverse()
plt.barh(y_pos, coeff_after_sorted,  align='center')
plt.yticks(y_pos, ranked_coeff_after)
plt.xlabel('coeffient')
plt.title('coefficient of features (after)')
plt.show()
```

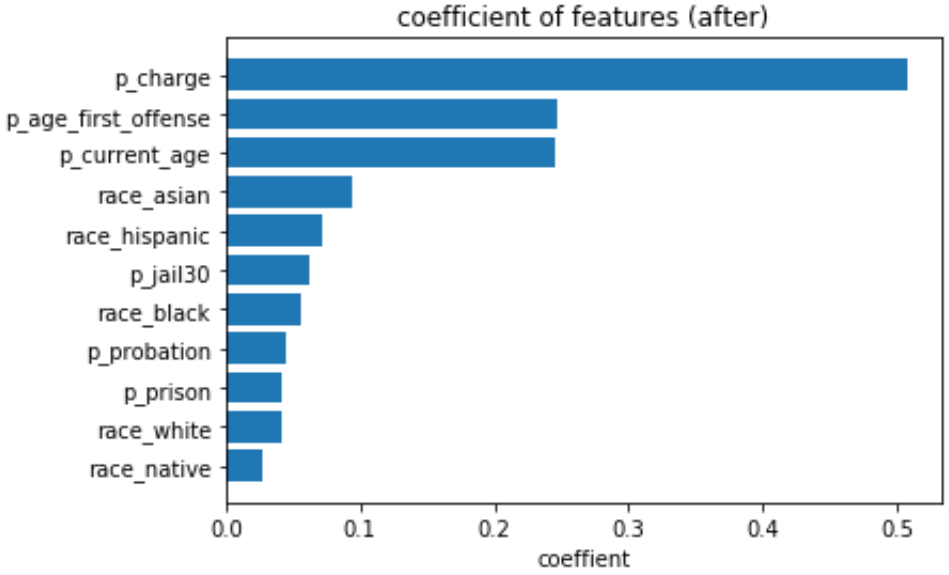
```
coefficient of features (before standardized):
[-0.02568038 -0.02156813  0.0539297  -0.25097748 -0.05703521  0.06106266
  0.22635064  0.09616959 -0.10899033 -0.38475942  0.02531225]

coefficient of features(after standardized):
[-0.24566361 -0.24760593  0.50792064 -0.06219097 -0.04224698  0.04405448
  0.05566429 -0.04183352 -0.07209308 -0.09477024 -0.02655613]

ranking of coefficient of features (before standardized):
race_asian
p_jail30
race_black
race_hispanic
race_white
p_probation
p_prison
p_charge
p_current_age
race_native
p_age_first_offense

ranking of coefficient of features (after standardized):
p_charge
p_age_first_offense
p_current_age
race_asian
race_hispanic
p_jail30
race_black
p_probation
p_prison
race_white
race_native
```


coefficient of features (before)
race_asian
p_jail30
race_black
race_hispanic
race_white
p_probation
p_prison
p_charge
p_current_age
race_native
p_age_first_offense
0.00
0.05
0.10
0.15
0.20
0.25
0.30
0.35
0.40
coeffient


coefficient of features (after)
p_charge
p_age_first_offense
p_current_age
race_asian
race_hispanic
p_jail30
race_black
p_probation
p_prison
race_white
race_native
0.0
0.1
0.2
0.3
0.4
0.5
coeffient

The ranking of feature importance:
p_charge p_current_age
p_age_first_offense
p_probation
p_prison
race_black
race_white
race_hispanic
p_jail30
race_native
race_asian

After standardized the the data and run a linear classifier, the overall order of the coefficient of variable are similar to the feature importance generated by the random forest. The order is not 100% percent consistent but they both indicates that the criminal history have much more importance than the other two. And the race has the least importance among these three. However ,the data without standardized say the otherwise. I think we should rely on the second part.

3.e

What ProPublica says is that someone should get arrested highly dependent on the race, however, my models totally disagree the analysis both for the random forest and for the linear classifier. In problem 3.d, it mentioned that Propublica used coefficients of variables generated by a linear classifier and went into that conclusion. When I replicated the test, before I standardrized the data, I ran into the same conclusion. Therefore, I think ProPublica did the simliar test which is running the linear classifier without standandrizing the data into a certain scale which would possibly overfit the data because linear classifer would tend to match every data points in the set but random forest would use bagging and random spaces to reduce the variance between samples sets and features. Therefore, even if we do not standardrize the data for the RF, we still get the more reasonable result.

In [ ]: