In [ ]:
```
#Group 8 - Wk4 - Ensemble Methods
#Group Member: Athena Zhang, Pratheek Praveen Kumar, Weifeng Li, Wenke Yu, Ziq
iao Wei
```

Import packages

Make sure you installed ***eli5***, ***sklearn***, ***matplotlib*** and ***numpy*** if you use your local machine

In [2]:
```
!pip install eli5
```

```
Collecting eli5
  Downloading eli5-0.11.0-py2.py3-none-any.whl (106 kB)
Requirement already satisfied: jinja2 in c:\users\student\anaconda3\lib\site-
packages (from eli5) (2.11.1)
Collecting graphviz
  Downloading graphviz-0.20-py3-none-any.whl (46 kB)
Collecting tabulate>=0.7.7
  Downloading tabulate-0.8.9-py3-none-any.whl (25 kB)
Requirement already satisfied: attrs>16.0.0 in c:\users\student\anaconda3\lib
\site-packages (from eli5) (19.3.0)
Requirement already satisfied: scipy in c:\users\student\anaconda3\lib\site-p
ackages (from eli5) (1.4.1)
Requirement already satisfied: six in c:\users\student\anaconda3\lib\site-pac
kages (from eli5) (1.14.0)
Requirement already satisfied: scikit-learn>=0.20 in c:\users\student\anacond
a3\lib\site-packages (from eli5) (0.22.1)
Requirement already satisfied: numpy>=1.9.0 in c:\users\student\anaconda3\lib
\site-packages (from eli5) (1.18.1)
Requirement already satisfied: MarkupSafe>=0.23 in c:\users\student\anaconda3
\lib\site-packages (from jinja2->eli5) (1.1.1)
Requirement already satisfied: joblib>=0.11 in c:\users\student\anaconda3\lib
\site-packages (from scikit-learn>=0.20->eli5) (0.14.1)
Installing collected packages: graphviz, tabulate, eli5
Successfully installed eli5-0.11.0 graphviz-0.20 tabulate-0.8.9
```

In [3]:
```
import eli5
import matplotlib.pyplot as plt
import numpy as np
import sklearn
from sklearn import datasets
from sklearn.pipeline import FeatureUnion
from sklearn.pipeline import Pipeline, make_pipeline
from sklearn.base import BaseEstimator, TransformerMixin
from sklearn.feature_extraction import DictVectorizer
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.metrics import confusion_matrix, precision_score, precision_recal
l_curve, recall_score, f1_score
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
```

```
C:\Users\student\anaconda3\lib\site-packages\sklearn\utils\deprecation.py:14
4: FutureWarning: The sklearn.feature_selection.base module is  deprecated in
version 0.22 and will be removed in version 0.24. The corresponding classes /
functions should instead be imported from sklearn.feature_selection. Anything
that cannot be imported from sklearn.feature_selection is now part of the pri
vate API.
  warnings.warn(message, FutureWarning)
```

Compare Logistic Regression and Decision Tree

Prepare dataset and Pick two classes

Your two classes should be similar, but opposite in some sense

In [4]:
```
# categories = ['alt.atheism', 'soc.religion.christian']
categories = ['comp.sys.ibm.pc.hardware', 'comp.sys.mac.hardware']
# categories = ['rec.sport.baseball', 'rec.sport.hockey']
# 'alt.atheism','comp.graphics','comp.os.ms-windows.misc','comp.sys.ibm.pc.har
dware',
# 'comp.sys.mac.hardware','comp.windows.x', 'misc.forsale', 'rec.autos',
# 'rec.motorcycles',  'rec.sport.baseball', 'rec.sport.hockey', 'sci.crypt',
# 'sci.electronics', 'sci.med', 'sci.space', 'soc.religion.christian', 'talk.p
olitics.guns',
# 'talk.politics.mideast', 'talk.politics.misc', 'talk.religion.misc'
train = sklearn.datasets.fetch_20newsgroups(subset='train', categories=categor
ies, remove=('headers', 'footers', 'quotes'),)
test = sklearn.datasets.fetch_20newsgroups(subset='test', categories=categorie
s, remove=('headers', 'footers', 'quotes'),)
print('train data size:', len(train.data))
print('test data size:', len(test.data))
```

```
train data size: 1168
test data size: 777
```

Compare Logistic Regression and Decision Tree models

In [5]:
```
lr_model = LogisticRegression(C=1, solver='newton-cg')
lr_features = CountVectorizer()
lr_classifier = make_pipeline(lr_features, lr_model)
lr_classifier.fit(train.data, train.target)

dt_model = DecisionTreeClassifier(min_samples_split=0.4)
dt_features = CountVectorizer()
dt_classifier = make_pipeline(dt_features, dt_model)
dt_classifier.fit(train.data, train.target)

#Compare accuracy of the two models
lr_train_preds = lr_classifier.predict(train.data)
lr_train_f1 = f1_score(train.target, lr_train_preds, average='micro')
lr_test_preds = lr_classifier.predict(test.data)
lr_test_f1 = f1_score(test.target, lr_test_preds, average='micro')
print("Train/test F1 for Logistic Regression: ", lr_train_f1, lr_test_f1)

dt_train_preds = dt_classifier.predict(train.data)
dt_train_f1 = f1_score(train.target, dt_train_preds, average='micro')
dt_test_preds = dt_classifier.predict(test.data)
dt_test_f1 = f1_score(test.target, dt_test_preds, average='micro')
print("Train/test F1 for Decision Tree: ", dt_train_f1, dt_test_f1)
```

```
Train/test F1 for Logistic Regression:  0.9897260273972602 0.8095238095238095
Train/test F1 for Decision Tree:  0.7696917808219178 0.7438867438867439
```

In [6]:
```
eli5.show_weights(lr_classifier, top=20, target_names=test.target_names)
```

Out[6]:

y=comp.sys.mac.hardware top features

Weight?	Feature
+1.980	mac
+1.555	apple
+1.055	centris
+1.011	quadra
+0.882	se
+0.825	lc
+0.780	nubus
+0.772	want
+0.760	adb
… 6966 more positive …	
… 4946 more negative …	
-0.766	motherboard
-0.830	ide
-0.837	bios
-0.839	gateway
-0.853	vlb
-0.861	help
-0.868	windows
-0.922	dos
-0.981	486
-1.144	controller
-1.357	pc

In [7]:
```
eli5.show_weights(dt_classifier, top=10, target_names=test.target_names)
#Play with the min_samples_split parameter when creating dt_classifier and see how the tree changes
#TODO FOR STUDENT: What happens when to the tree when you modify the min_samples_split_parameter
```

Out[7]:

Weight	Feature
0.2449	mac
0.2005	apple
0.0837	controller
0.0800	dos
0.0622	pc
0.0534	quadra
0.0438	windows
0.0363	set
0.0327	help
0.0315	vlb
… 11921 more …	

```
mac <= 0.500  (86.0%)
    apple <= 0.500  (78.2%)
        controller <= 0.500  (71.0%)
            dos <= 0.500  (65.8%)
                pc <= 0.500  (61.3%)
                    quadra <= 0.500  (59.2%)
                        windows <= 0.500  (56.1%)
                            set <= 0.500  (52.7%)
                                vlb <= 0.500  (51.3%)
                                    centris <= 0.500  (50.0%)
                                        lc <= 0.500  (48.8%)
                                            powerbook <= 0.500  (47.6%)
                                                help <= 0.500  (43.4%)
                                                    se <= 0.500  (42.2%)
                                                        card <= 0.500  (37.
8%)  ---> 0.511
                                                        card > 0.500  (4.4%)
---> 0.255
                                                    se > 0.500  (1.2%)  --->
1.000
                                                help > 0.500  (4.2%)  ---> 0.
184
                                            powerbook > 0.500  (1.2%)  --->
1.000
                                        lc > 0.500  (1.2%)  ---> 1.000
                                    centris > 0.500  (1.3%)  ---> 1.000
                                vlb > 0.500  (1.5%)  ---> 0.000
                            set > 0.500  (3.3%)  ---> 0.128
                        windows > 0.500  (3.1%)  ---> 0.056
                    quadra > 0.500  (2.1%)  ---> 1.000
                pc > 0.500  (4.5%)  ---> 0.057
            dos > 0.500  (5.1%)  ---> 0.000
        controller > 0.500  (7.2%)  ---> 0.024
    apple > 0.500  (7.9%)  ---> 0.946
mac > 0.500  (14.0%)  ---> 0.914
```

In [8]:
```
idx = 2
x = test.data[idx]
print(test.target_names[test.target[idx]])
eli5.show_prediction(lr_model, test.data[idx], vec=lr_features, target_names=test.target_names)
```

```
comp.sys.ibm.pc.hardware
```

Out[8]: **y=comp.sys.ibm.pc.hardware** (probability **0.987**, score **-4.350**) top features

Contribution?	Feature
+4.876	Highlighted in text (sum)
-0.526	<BIAS>

[remainder deleted] i don't have my copy of the manual with me right now, but i can offer the following in the interim: 1) the card uses port addresses 0x2e0 and 0x2e8 (which are not configurable). these addresses, incidentally, were inadvertantly omitted from my version of the manual. 2) i believe there is a dip that controls whether or not to enable irq 2 (for cga or ega support??!?). lance hartmann (lance%hartmann.austin.ibm.com@ibmpa.awdpa.ibm.com) yes, that is a '%' (percent sign) in my network address.

In [9]:
```
eli5.show_prediction(dt_model, test.data[idx], vec=dt_features, target_names=test.target_names)
```

Out[9]: **y=comp.sys.ibm.pc.hardware** (probability **0.745**) top features

Contribution?	Feature
+0.505	<BIAS>
+0.230	Highlighted in text (sum)
+0.068	mac
+0.052	apple
+0.019	quadra
+0.014	se
+0.013	powerbook
+0.013	centris
+0.012	lc
-0.014	vlb
-0.022	windows
-0.022	set
-0.028	help
-0.029	pc
-0.032	dos
-0.036	controller

[remainder deleted] i don't have my copy of the manual with me right now, but i can offer the following in the interim: 1) the card uses port addresses 0x2e0 and 0x2e8 (which are not configurable). these addresses, incidentally, were inadvertantly omitted from my version of the manual. 2) i believe there is a dip that controls whether or not to enable irq 2 (for cga or ega support??!?). lance hartmann (lance%hartmann.austin.ibm.com@ibmpa.awdpa.ibm.com) yes, that is a '%' (percent sign) in my network address.

Ensemble Methods

In [10]:
```
from sklearn.ensemble import VotingClassifier

features = CountVectorizer()

lr_model = LogisticRegression(C=1, solver='lbfgs')
lr_classifier = make_pipeline(features, lr_model)
lr_classifier.fit(train.data, train.target)

#TODO FOR STUDENT: Try playing with the min_samples_split to see how it affect
the ensemble score
dt_model = DecisionTreeClassifier(min_samples_split=0.2)
dt_classifier = make_pipeline(features, dt_model)
dt_classifier.fit(train.data, train.target)

#Compare accuracy of the two models
lr_train_preds = lr_classifier.predict(train.data)
lr_train_f1 = f1_score(train.target, lr_train_preds, average='micro')
lr_test_preds = lr_classifier.predict(test.data)
lr_test_f1 = f1_score(test.target, lr_test_preds, average='micro')
print("Train/test F1 for Logistic Regression: ", lr_train_f1, lr_test_f1)

dt_train_preds = dt_classifier.predict(train.data)
dt_train_f1 = f1_score(train.target, dt_train_preds, average='micro')
dt_test_preds = dt_classifier.predict(test.data)
dt_test_f1 = f1_score(test.target, dt_test_preds, average='micro')
print("Train/test F1 for Decision Tree: ", dt_train_f1, dt_test_f1)

#Look at classifier agreement
print("\n% Cases where the two classifiers agree on test data: ", np.sum(lr_te
st_preds == dt_test_preds)/len(lr_test_preds))
print("% Cases where one of the two classifiers has correct answer: ", np.sum(
np.logical_or(lr_test_preds == test.target, dt_test_preds == test.target)/len(
lr_test_preds)))

#Try to build an ensemble combing both models
#TODO FOR STUDENT: Modify the weights parameter which give different weight to
each of the classifiers
ensemble_classifier = make_pipeline(lr_features, VotingClassifier(estimators=
[('lr', lr_model), ('dt', dt_model)], voting='soft', weights=[3,2]))
ensemble_classifier.fit(train.data, train.target)

ensemble_train_preds = ensemble_classifier.predict(train.data)
ensemble_train_f1 = f1_score(train.target, ensemble_train_preds, average='micr
o')
ensemble_test_preds = ensemble_classifier.predict(test.data)
ensemble_test_f1 = f1_score(test.target, ensemble_test_preds, average='micro')
print("\nTrain/test F1 for Ensemble: ", ensemble_train_f1, ensemble_test_f1)
```

```
Train/test F1 for Logistic Regression:  0.9897260273972602 0.8082368082368082
Train/test F1 for Decision Tree:  0.8655821917808221 0.7902187902187903

% Cases where the two classifiers agree on test data:  0.7915057915057915
% Cases where one of the two classifiers has correct answer:  0.9034749034749
034

Train/test F1 for Ensemble:  0.988013698630137 0.8288288288288288
```

Bagging

In [11]:
```
from sklearn.ensemble import RandomForestClassifier

#TODO FOR STUDENT: Try playing with n_estimators and min_samples_split
ensemble_classifier = make_pipeline(lr_features, RandomForestClassifier(n_esti
mators=1000, min_samples_split=0.01))
ensemble_classifier.fit(train.data, train.target)

ensemble_train_preds = ensemble_classifier.predict(train.data)
ensemble_train_f1 = f1_score(train.target, ensemble_train_preds, average='micr
o')
ensemble_test_preds = ensemble_classifier.predict(test.data)
ensemble_test_f1 = f1_score(test.target, ensemble_test_preds, average='micro')
print("\nTrain/test F1 for Ensemble: ", ensemble_train_f1, ensemble_test_f1)
```

```
Train/test F1 for Ensemble:  0.9888698630136986 0.8378378378378378
```

Boosting

In [12]:
```
from sklearn.ensemble import AdaBoostClassifier
ensemble_classifier = make_pipeline(lr_features, AdaBoostClassifier(n_estimato
rs=100, learning_rate=1.0))
ensemble_classifier.fit(train.data, train.target)

ensemble_train_preds = ensemble_classifier.predict(train.data)
ensemble_train_f1 = f1_score(train.target, ensemble_train_preds, average='micr
o')
ensemble_test_preds = ensemble_classifier.predict(test.data)
ensemble_test_f1 = f1_score(test.target, ensemble_test_preds, average='micro')
print("\nTrain/test F1 for Ensemble: ", ensemble_train_f1, ensemble_test_f1)
```

```
Train/test F1 for Ensemble:  0.9392123287671232 0.833976833976834
```

In [13]:
```
from sklearn.ensemble import GradientBoostingClassifier

#TODO FOR STUDENT: Try playing with n_estimators and min_samples_split
ensemble_classifier = make_pipeline(lr_features, GradientBoostingClassifier(n_
estimators=500, min_samples_split=0.05))
ensemble_classifier.fit(train.data, train.target)

ensemble_train_preds = ensemble_classifier.predict(train.data)
ensemble_train_f1 = f1_score(train.target, ensemble_train_preds, average='micr
o')
ensemble_test_preds = ensemble_classifier.predict(test.data)
ensemble_test_f1 = f1_score(test.target, ensemble_test_preds, average='micro')
print("\nTrain/test F1 for Ensemble: ", ensemble_train_f1, ensemble_test_f1)
```

```
Train/test F1 for Ensemble:  0.988013698630137 0.8314028314028314
```

In [14]:
```
from sklearn.ensemble import GradientBoostingClassifier

#TODO FOR STUDENT: Try playing with n_estimators and min_samples_split
ensemble_classifier = make_pipeline(lr_features, GradientBoostingClassifier(n_
estimators=500, min_samples_split=0.05))
ensemble_classifier.fit(train.data, train.target)

ensemble_train_preds = ensemble_classifier.predict(train.data)
ensemble_train_f1 = f1_score(train.target, ensemble_train_preds, average='micr
o')
ensemble_test_preds = ensemble_classifier.predict(test.data)
ensemble_test_f1 = f1_score(test.target, ensemble_test_preds, average='micro')
print("\nTrain/test F1 for Ensemble: ", ensemble_train_f1, ensemble_test_f1)
```

```
Train/test F1 for Ensemble:  0.988013698630137 0.842985842985843
```

Comparing Bagging and Boosting

In [15]:
```
for n_est in range(50,500,50):
  ensemble_classifier = make_pipeline(lr_features, RandomForestClassifier(n_es
timators=n_est, min_samples_split=0.05))
  ensemble_classifier.fit(train.data, train.target)

  ensemble_train_preds = ensemble_classifier.predict(train.data)
  ensemble_train_f1 = f1_score(train.target, ensemble_train_preds, average='mi
cro')
  ensemble_test_preds = ensemble_classifier.predict(test.data)
  ensemble_test_f1 = f1_score(test.target, ensemble_test_preds, average='micr
o')
  print(n_est, "Train/test F1 for Ensemble: ", ensemble_train_f1, ensemble_tes
t_f1)
```

```
50 Train/test F1 for Ensemble:  0.9743150684931506 0.8314028314028314
100 Train/test F1 for Ensemble:  0.978595890410959 0.8301158301158301
150 Train/test F1 for Ensemble:  0.978595890410959 0.842985842985843
200 Train/test F1 for Ensemble:  0.9768835616438356 0.8326898326898327
250 Train/test F1 for Ensemble:  0.978595890410959 0.8301158301158301
300 Train/test F1 for Ensemble:  0.978595890410959 0.8378378378378378
350 Train/test F1 for Ensemble:  0.9803082191780822 0.8391248391248392
400 Train/test F1 for Ensemble:  0.9768835616438356 0.8326898326898327
450 Train/test F1 for Ensemble:  0.9803082191780822 0.8416988416988417
```

In [16]:
```
for n_est in range(50,500,50):
  ensemble_classifier = make_pipeline(lr_features, RandomForestClassifier(n_es
timators=n_est, min_samples_split=0.5))
  ensemble_classifier.fit(train.data, train.target)

  ensemble_train_preds = ensemble_classifier.predict(train.data)
  ensemble_train_f1 = f1_score(train.target, ensemble_train_preds, average='mi
cro')
  ensemble_test_preds = ensemble_classifier.predict(test.data)
  ensemble_test_f1 = f1_score(test.target, ensemble_test_preds, average='micr
o')
  print(n_est, "Train/test F1 for Ensemble: ", ensemble_train_f1, ensemble_tes
t_f1)
```

```
50 Train/test F1 for Ensemble:  0.853595890410959 0.7824967824967825
100 Train/test F1 for Ensemble:  0.8878424657534246 0.806949806949807
150 Train/test F1 for Ensemble:  0.8801369863013698 0.8120978120978121
200 Train/test F1 for Ensemble:  0.8861301369863014 0.8198198198198198
250 Train/test F1 for Ensemble:  0.8981164383561644 0.824967824967825
300 Train/test F1 for Ensemble:  0.8972602739726028 0.821106821106821
350 Train/test F1 for Ensemble:  0.8929794520547946 0.8288288288288288
400 Train/test F1 for Ensemble:  0.8998287671232876 0.8301158301158301
450 Train/test F1 for Ensemble:  0.8921232876712328 0.8262548262548263
```

In [17]:
```
for n_est in range(50,500,50):
  ensemble_classifier = make_pipeline(lr_features, GradientBoostingClassifier(
n_estimators=n_est, min_samples_split=0.05))
  ensemble_classifier.fit(train.data, train.target)

  ensemble_train_preds = ensemble_classifier.predict(train.data)
  ensemble_train_f1 = f1_score(train.target, ensemble_train_preds, average='mi
cro')
  ensemble_test_preds = ensemble_classifier.predict(test.data)
  ensemble_test_f1 = f1_score(test.target, ensemble_test_preds, average='micr
o')
  print(n_est, "Train/test F1 for Ensemble: ", ensemble_train_f1, ensemble_tes
t_f1)
```

```
50 Train/test F1 for Ensemble:  0.8373287671232876 0.788931788931789
100 Train/test F1 for Ensemble:  0.9238013698630136 0.8314028314028314
150 Train/test F1 for Ensemble:  0.9511986301369864 0.8365508365508365
200 Train/test F1 for Ensemble:  0.9683219178082192 0.8378378378378378
250 Train/test F1 for Ensemble:  0.9837328767123288 0.8314028314028314
300 Train/test F1 for Ensemble:  0.9845890410958904 0.8391248391248392
350 Train/test F1 for Ensemble:  0.9845890410958904 0.8365508365508365
400 Train/test F1 for Ensemble:  0.9863013698630136 0.8468468468468469
450 Train/test F1 for Ensemble:  0.985445205479452 0.8468468468468469
```

In [18]:
```
for n_est in range(50,500,50):
  ensemble_classifier = make_pipeline(lr_features, GradientBoostingClassifier(
n_estimators=n_est, min_samples_split=0.5))
  ensemble_classifier.fit(train.data, train.target)

  ensemble_train_preds = ensemble_classifier.predict(train.data)
  ensemble_train_f1 = f1_score(train.target, ensemble_train_preds, average='mi
cro')
  ensemble_test_preds = ensemble_classifier.predict(test.data)
  ensemble_test_f1 = f1_score(test.target, ensemble_test_preds, average='micr
o')
  print(n_est, "Train/test F1 for Ensemble: ", ensemble_train_f1, ensemble_tes
t_f1)
```

```
50 Train/test F1 for Ensemble:  0.8210616438356164 0.7850707850707851
100 Train/test F1 for Ensemble:  0.877568493150685 0.8095238095238095
150 Train/test F1 for Ensemble:  0.9357876712328768 0.824967824967825
200 Train/test F1 for Ensemble:  0.9537671232876712 0.8352638352638352
250 Train/test F1 for Ensemble:  0.9803082191780822 0.8404118404118404
300 Train/test F1 for Ensemble:  0.9828767123287672 0.8404118404118404
350 Train/test F1 for Ensemble:  0.9837328767123288 0.8365508365508365
400 Train/test F1 for Ensemble:  0.985445205479452 0.8326898326898327
450 Train/test F1 for Ensemble:  0.9837328767123288 0.8352638352638352
```