In [1]:

```
import numpy as np
import pandas as pd
import os
import sklearn
```

Reading Data

In [2]:

```
def segmentWords(s):
    return s.split()

def readFile(fileName):
    # Function for reading file
    # input: filename as string
    # output: contents of file as list containing single words
    contents = []
    f = open(fileName)
    for line in f:
        contents.append(line)
    f.close()
    result = segmentWords('\n'.join(contents))
    return result
```

Create a Dataframe containing the counts of each word in a file

In [35]:

```
d = []

for c in os.listdir("data_training/train"):
    directory = "data_training/train/" + c
    for f in os.listdir(directory):
        #print(f)
        words = readFile(directory + "/" + f)
        e = {x:words.count(x) for x in words}
        e['__FileID__'] = f
        e['__CLASS__'] = c
        d.append(e)
```

Create a dataframe from d - make sure to fill all the nan values with zeros.

Hint: Consider the `fillna()` function for Dataframes

In [36]:

```
df = pd.DataFrame(d)
df.fillna(0, inplace=True)
```

In [38]:

```
df['__CLASS__'].unique()
```

Out[38]:

```
array(['neg', 'pos'], dtype=object)
```

Split data into training and validation set

- Sample 80% of your dataframe to be the training data
- Let the remaining 20% be the validation data (you can filter out the indicies of the original dataframe that weren't selected for the training data)

In [43]:

```
from sklearn.model_selection import train_test_split
training,testing = train_test_split(df, test_size=0.2)
```

- Split the dataframe for both training and validation data into x and y dataframes - where y contains the labels and x contains the words

Hint: Try looking at the Dataframe `drop()` function

In [55]:

```
y_train = training[['__CLASS__','__FileID__']]
y_test = testing[['__CLASS__','__FileID__']]
x_train = training.drop(['__CLASS__','__FileID__'], axis = 1)
x_test = testing.drop(['__CLASS__','__FileID__'], axis = 1)
```

In [ ]:

```

```

Logistic Regression

Basic Logistic Regression

- Use sklearn's linear_model.LogisticRegression() to create your model.
- Fit the data and labels with your model.
- Score your model with the same data and labels.

In [56]:

```
from sklearn.linear_model import LogisticRegression
```

In [71]:

```
y_train = y_train.drop('__FileID__', axis = 1)
y_test = y_test.drop('__FileID__', axis = 1)
```

In [73]:

```
lr = LogisticRegression()
lr.fit(x_train,y_train)
score = lr.score(x_test,y_test)
score
```

```
/Users/glennparham/anaconda/lib/python3.6/site-packages/sklearn/util
s/validation.py:578: DataConversionWarning: A column-vector y was pa
ssed when a 1d array was expected. Please change the shape of y to (
n_samples, ), for example using ravel().
  y = column_or_1d(y, warn=True)
```

Out[73]:

```
0.83571428571428574
```

Changing Parameters

In [ ]:

```

```

Feature Selection

- In the backward stepsize selection method, you can remove coefficients and the corresponding x columns, where the coefficient is more than a particular amount away from the mean - you can choose how far from the mean is reasonable.

Hint: Numpy's `argwhere()` might be useful here
Hint: Instead of defining a hard-coded constant to determine which features to keep or remove, consider using values relative to the distribution of the weight magnitudes

In [83]:

```
x_train_norm = abs(x_train - x_train.mean()) / abs(x_train.max() - x_train.min()
)
x_train_norm.fillna(0, inplace=True)
```

In [85]:

```
sum(x_train_norm.iloc[1].values)
#remove columns of x_train_norm where it is far from 1  (aka very close to 0) be
cause that means it's not significant
```

Out[85]:

```
298.38781784616151
```

How did you select which features to remove? Why did that reduce overfitting?

WRITE ANSWER TO 4.1 HERE

Single Decision Tree

Basic Decision Tree

- Initialize your model as a decision tree with sklearn.
- Fit the data and labels to the model.

In [93]:

```
from sklearn.model_selection import GridSearchCV
from sklearn.grid_search import RandomizedSearchCV
from sklearn.ensemble import RandomForestClassifier
from sklearn.ensemble import AdaBoostClassifier
from sklearn import tree
```

In [116]:

```
clf = tree.DecisionTreeClassifier()
clf.fit(x_train, y_train)
```

Out[116]:

```
DecisionTreeClassifier(class_weight=None, criterion='gini', max_dept
h=None,
            max_features=None, max_leaf_nodes=None,
            min_impurity_decrease=0.0, min_impurity_split=None,
            min_samples_leaf=1, min_samples_split=2,
            min_weight_fraction_leaf=0.0, presort=False, random_stat
e=None,
            splitter='best')
```

Changing Parameters

- To test out which value is optimal for a particular parameter, you can either loop through various values or look into `sklearn.model_selection.GridSearchCV`

In [121]:

```
dot_data = tree.export_graphviz(clf,
                                feature_names=x_train.columns,
                                class_names=y_train)
```

```
/Users/glennparham/anaconda/lib/python3.6/site-packages/sklearn/tree
/export.py:399: DeprecationWarning: out_file can be set to None star
ting from 0.18. This will be the default in 0.20.
  DeprecationWarning)

---------------------------------------------------------------------
-------
KeyError                                  Traceback (most recent cal
l last)
<ipython-input-121-36c036748431> in <module>()
      1 dot_data = tree.export_graphviz(clf,
      2                                 feature_names=x_train.columns,
----> 3                                 class_names=y_train)

/Users/glennparham/anaconda/lib/python3.6/site-packages/sklearn/tree
/export.py in export_graphviz(decision_tree, out_file, max_depth, fe
ature_names, class_names, label, filled, leaves_parallel, impurity,
node_ids, proportion, rotate, rounded, special_characters, precision
)
    462             recurse(decision_tree, 0, criterion="impurity")
    463         else:
--> 464             recurse(decision_tree.tree_, 0, criterion=decisi
on_tree.criterion)
    465
    466         # If required, draw leaf nodes at same depth as each
other
```

```
/Users/glennparham/anaconda/lib/python3.6/site-packages/sklearn/tree
/export.py in recurse(tree, node_id, criterion, parent, depth)
    330             out_file.write('%d [label=%s'
    331                            % (node_id,
--> 332                               node_to_str(tree, node_id, cri
terion)))
    333
    334             if filled:

/Users/glennparham/anaconda/lib/python3.6/site-packages/sklearn/tree
/export.py in node_to_str(tree, node_id, criterion)
    295                 node_string += 'class = '
    296             if class_names is not True:
--> 297                 class_name = class_names[np.argmax(value)]
    298             else:
    299                 class_name = "y%s%s%s" % (characters[1],

/Users/glennparham/anaconda/lib/python3.6/site-packages/pandas/core/
frame.py in __getitem__(self, key)
   1795             return self._getitem_multilevel(key)
   1796         else:
-> 1797             return self._getitem_column(key)
   1798
   1799     def _getitem_column(self, key):

/Users/glennparham/anaconda/lib/python3.6/site-packages/pandas/core/
frame.py in _getitem_column(self, key)
   1802         # get column
   1803         if self.columns.is_unique:
-> 1804             return self._get_item_cache(key)
   1805
   1806         # duplicate columns & possible reduce dimensionailit
y

/Users/glennparham/anaconda/lib/python3.6/site-packages/pandas/core/
generic.py in _get_item_cache(self, item)
   1082         res = cache.get(item)
   1083         if res is None:
-> 1084             values = self._data.get(item)
   1085             res = self._box_item_values(item, values)
   1086             cache[item] = res

/Users/glennparham/anaconda/lib/python3.6/site-packages/pandas/core/
internals.py in get(self, item, fastpath)
   2849
   2850             if not isnull(item):
-> 2851                 loc = self.items.get_loc(item)
   2852             else:
   2853                 indexer = np.arange(len(self.items))[isnull(
self.items)]

/Users/glennparham/anaconda/lib/python3.6/site-packages/pandas/core/
```

```
index.py in get_loc(self, key, method)
   1570         """
   1571         if method is None:
-> 1572             return self._engine.get_loc(_values_from_object(
key))
   1573
   1574         indexer = self.get_indexer([key], method=method)

pandas/index.pyx in pandas.index.IndexEngine.get_loc (pandas/index.c
:3824)()

pandas/index.pyx in pandas.index.IndexEngine.get_loc (pandas/index.c
:3704)()

pandas/hashtable.pyx in pandas.hashtable.PyObjectHashTable.get_item
(pandas/hashtable.c:12280)()

pandas/hashtable.pyx in pandas.hashtable.PyObjectHashTable.get_item
(pandas/hashtable.c:12231)()

KeyError: 1
```

In [114]:

```
x_train.columns
```

Out[114]:

```
Index(['', 'earth', 'goodies', 'if', 'ripley', 'suspend', 'they',
       'white', '', '',
       ...
       'zukovsky', 'zundel', 'zurg's', 'zweibel', 'zwick', 'zwick's'
,
       'zwigoff's', 'zycie', 'zycie'', '|''],
      dtype='object', length=42774)
```

In [ ]:

```
graph = graphviz.Source(dot_data)
graph
```

In [101]:

```
clf.score(x_train, y_train)
```

Out[101]:

```
1.0
```

In [102]:

```
clf.score(x_test, y_test)
```

Out[102]:

```
0.67500000000000004
```

How did you choose which parameters to change and what value to give to them? Feel free to show a plot.

Why is a single decision tree so prone to overfitting?

In [ ]:

```

```

Random Forest Classifier

Basic Random Forest

- Use sklearn's ensemble.RandomForestClassifier() to create your model.
- Fit the data and labels with your model.
- Score your model with the same data and labels.

In [107]:

```
clf = RandomForestClassifier(n_estimators=100)
clf.fit(x_train, y_train)
```

```
/Users/glennparham/anaconda/lib/python3.6/site-packages/ipykernel/__
main__.py:2: DataConversionWarning: A column-vector y was passed whe
n a 1d array was expected. Please change the shape of y to (n_sample
s,), for example using ravel().
  from ipykernel import kernelapp as app
```

Out[107]:

```
RandomForestClassifier(bootstrap=True, class_weight=None, criterion=
'gini',
            max_depth=None, max_features='auto', max_leaf_nodes=None
,
            min_impurity_decrease=0.0, min_impurity_split=None,
            min_samples_leaf=1, min_samples_split=2,
            min_weight_fraction_leaf=0.0, n_estimators=100, n_jobs=1
,
            oob_score=False, random_state=None, verbose=0,
            warm_start=False)
```

In [108]:

```
clf.score(x_train, y_train)
```

Out[108]:

1.0

In [109]:

```
clf.score(x_test, y_test)
```

Out[109]:

0.778571428571428 58

Changing Parameters

In [ ]:

```

```

What parameters did you choose to change and why?

In [ ]:

```

```

How does a random forest classifier prevent overfitting better than a single decision tree?

In [ ]:

```

```