Compare multiple algorithms after blending text and feature based predictions

Import packages and data

In [1]:
```
#import packages
import pandas as pd
import re
from sklearn.ensemble import RandomForestClassifier
from IPython.display import display
import numpy as np
import math
from sklearn import metrics
from pandas.api.types import is_string_dtype, is_numeric_dtype
import matplotlib.pyplot as plt
from sklearn.ensemble import forest
import scipy
from scipy.cluster import hierarchy as hc
from sklearn.metrics import accuracy_score, balanced_accuracy_score, f1_sco
re, classification_report
import xgboost as xgb
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.feature_extraction.text import TfidfTransformer
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.metrics import confusion_matrix
from sklearn.linear_model import LogisticRegression
import lightgbm as lgb
from lightgbm import LGBMModel
#Run xgboost on dataframe
import xgboost as xgb
from xgboost import XGBClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.svm import LinearSVC
```

In [2]:
```
#import data (do only once)
cases = pd.read_excel('incident V2 - Enriched.xlsx')
cases.shape
```

Out[2]: (34564, 45)

In [176]:
```
def plot_confusion_matrix(y_true, y_pred, classes,
                          normalize=False,
                          title=None,
                          cmap=plt.cm.Blues):
    """
    This function prints and plots the confusion matrix.
    Normalization can be applied by setting `normalize=True`.
    """
    if not title:
        if normalize:
            title = 'Normalized confusion matrix'
        else:
            title = 'Confusion matrix, without normalization'

    # Compute confusion matrix
    cm = confusion_matrix(y_true, y_pred)
    # Only use the labels that appear in the data
    #classes = classes[unique_labels(y_true, y_pred)]
    if normalize:
        cm = cm.astype('float') / cm.sum(axis=1)[:, np.newaxis]
        print("Normalized confusion matrix")
    else:
        print('Confusion matrix, without normalization')

    print(cm)

    fig, ax = plt.subplots()
    im = ax.imshow(cm, interpolation='nearest', cmap=cmap)
    ax.figure.colorbar(im, ax=ax)
    # We want to show all ticks...
    ax.set(xticks=np.arange(cm.shape[1]),
           yticks=np.arange(cm.shape[0]),
           # ... and label them with the respective list entries
           xticklabels=classes, yticklabels=classes,
           title=title,
           ylabel='True label',
           xlabel='Predicted label')

    # Rotate the tick labels and set their alignment.
    plt.setp(ax.get_xticklabels(), rotation=45, ha="right",
             rotation_mode="anchor")

    # Loop over data dimensions and create text annotations.
    fmt = '.2f' if normalize else 'd'
    thresh = cm.max() / 2.
    for i in range(cm.shape[0]):
        for j in range(cm.shape[1]):
            ax.text(j, i, format(cm[i, j], fmt),
                    ha="center", va="center",
                    color="white" if cm[i, j] > thresh else "black")
    fig.tight_layout()
    return ax

def rmse(x,y):
    return math.sqrt(((x-y)**2).mean())
```

```
def print_score(m):
    res = [rmse(m.predict(X_train), y_train), rmse(m.predict(X_valid), y_va
lid),
                m.score(X_train, y_train), m.score(X_valid, y_valid)]
    if hasattr(m, 'oob_score_'): res.append(m.oob_score_)
    print(res)

def split_vals(a,n):
    return a[:n].copy(), a[n:].copy()

def get_oob(df):
    m = RandomForestRegressor(n_estimators=40, min_samples_leaf=5, max_feat
ures=0.6, n_jobs=-1, oob_score=True)
    x, _ = split_vals(df, n_trn)
    m.fit(x, y_train)
    return m.oob_score_

def add_datepart(df, fldname, drop=True, time=False):
    fld = df[fldname]
    fld_dtype = fld.dtype
    if isinstance(fld_dtype, pd.core.dtypes.dtypes.DatetimeTZDtype):
        fld_dtype = np.datetime64

    if not np.issubdtype(fld_dtype, np.datetime64):
        df[fldname] = fld = pd.to_datetime(fld, infer_datetime_format=True)
    targ_pre = re.sub('[Dd]ate$', '', fldname)
    attr = ['Year', 'Month', 'Week', 'Day', 'Dayofweek', 'Dayofyear',
            'Is_month_end', 'Is_month_start', 'Is_quarter_end', 'Is_quarter
_start', 'Is_year_end', 'Is_year_start']
    if time: attr = attr + ['Hour', 'Minute', 'Second']
    for n in attr: df[targ_pre + n] = getattr(fld.dt, n.lower())
    df[targ_pre + 'Elapsed'] = fld.astype(np.int64) // 10 ** 9
    if drop: df.drop(fldname, axis=1, inplace=True)

def train_cats(df):
    for n,c in df.items():
        if is_string_dtype(c): df[n] = c.astype('category').cat.as_ordered
()

def fix_missing(df, col, name, na_dict):
    if is_numeric_dtype(col):
        if pd.isnull(col).sum() or (name in na_dict):
            df[name+'_na'] = pd.isnull(col)
            filler = na_dict[name] if name in na_dict else col.median()
            df[name] = col.fillna(filler)
            na_dict[name] = filler
    return na_dict

def proc_df(df, y_fld=None, skip_flds=None, ignore_flds=None, do_scale=Fals
e, na_dict=None,
            preproc_fn=None, max_n_cat=None, subset=None, mapper=None):
    if not ignore_flds: ignore_flds=[]
    if not skip_flds: skip_flds=[]
    if subset: df = get_sample(df,subset)
    else: df = df.copy()
    ignored_flds = df.loc[:, ignore_flds]
    df.drop(ignore_flds, axis=1, inplace=True)
```

```
    if preproc_fn: preproc_fn(df)
    if y_fld is None: y = None
    else:
        if not is_numeric_dtype(df[y_fld]): df[y_fld] = df[y_fld].cat.codes
        y = df[y_fld].values
        skip_flds += [y_fld]
    df.drop(skip_flds, axis=1, inplace=True)

    if na_dict is None: na_dict = {}
    else: na_dict = na_dict.copy()
    na_dict_initial = na_dict.copy()
    for n,c in df.items(): na_dict = fix_missing(df, c, n, na_dict)
    if len(na_dict_initial.keys()) > 0:
        df.drop([a + '_na' for a in list(set(na_dict.keys()) - set(na_dict_
initial.keys()))], axis=1, inplace=True)
    if do_scale: mapper = scale_vars(df, mapper)
    for n,c in df.items(): numericalize(df, c, n, max_n_cat)
    df = pd.get_dummies(df, dummy_na=True)
    df = pd.concat([ignored_flds, df], axis=1)
    res = [df, y, na_dict]
    if do_scale: res = res + [mapper]
    return res

def numericalize(df, col, name, max_n_cat):
    if not is_numeric_dtype(col) and ( max_n_cat is None or col.nunique()>m
ax_n_cat):
        df[name] = col.cat.codes+1
```

Set parameters

In [3]:
```
#Variables
no_of_ags = 15 #Number of AGs to consider when choosing AGs with most frequ
ency
#AGs to exclude from analysis.  Leave blank if none is excluded. This may c
hange depending on years chosen
remove_ags = ['Global Helpdesk - Tier 1', 'Japan Helpdesk Support','Global
 Helpdesk','Global ITSOC - Tier 1' ] #for 2018 and 2019
#Define test and train sets cases['opened_at'].dt.to_period('M')
test_period = ['2019-02', '2019-03', '2019-04']
train_period = ['2018-01', '2018-02', '2018-03', '2018-04','2018-05', '2018
-06', '2018-07', '2018-08','2018-09', '2018-10', '2018-11', '2018-12', '201
9-01']
#rounds = 100 #Number of times RF is run
# Whether to merge AGs or not
merge_ags = 'Y' #Set to 'N' if you dont want to merge AGs
model_params = dict(((k, eval(k)) for k in ('no_of_ags', 'remove_ags', 'tes
t_period','train_period', 'merge_ags' )))
```

Data prep

In [4]:
```
# Use copy so that we dont have import data for every run
df = cases.copy()
print('Full dataset shape:', df.shape)

# Use only closed cases
df = df[df['state'].isin(['Closed', 'Closed (CR Implemented)', 'Closed (Pur
chase Required)', 'Resolved'])].copy()
print('Only closed cases shape:', df.shape)

if merge_ags == 'Y':
    df['ag_merged'] = np.where(df['ag'].isin(['Finance Support','IT BSA - F
inance']), 'Merged Finance Support IT BSA - Finance', df['ag'])
    #Reset column names for convenience
    df.rename(columns={'ag': 'ag_old'},inplace=True)
    df.rename(columns={'ag_merged':'ag'}, inplace=True)\

#Create new text feature
for cols in ['short_description', 'description', 'Requester Cost Center Des
cr', 'Requester Location Desc']:
    df[cols] = df[cols].astype(str)
df['fulltext'] = df['short_description'] + ' ' + df['description'] + ' ' +
df['Requester Cost Center Descr'] + ' ' + df['Requester Location Desc']

#Filter cases based on period chosen
df['opened_at'] = pd.to_datetime(df['opened_at'])
df = df[df['opened_at'].dt.to_period('M').astype(str).isin(test_period + tr
ain_period)].copy()
print('Shape after selecting period', df.shape)

df_size = len(df)

#Filter cases based on AGs
keep_ag = list(df['ag'].value_counts().head(no_of_ags).index)
for i in remove_ags:
    keep_ag.remove(i)
df = df[df['ag'].isin(keep_ag)].copy()
print()
print('Shape of data subset: ', df.shape)


#Percentage of cases considered
print()
print('% of cases considered after taking subset: ', len(df)*100/df_size)

#AGs list and frequency
print()
print('AG list and frequencies')
print(df['ag'].value_counts())

#Change all object type to category
df[df.select_dtypes(['object']).columns] = df.select_dtypes(['object']).app
ly(lambda x: x.astype('category'))

# Display code to category mapping
print()
print('AG to codes mapping')
```

```
class_to_cat_mapping = dict(enumerate(df['ag'].cat.categories))
print(class_to_cat_mapping)

#Change AG to codes
df['ag'] = df['ag'].cat.codes
```

```
Full dataset shape: (34564, 45)
Only closed cases shape: (30874, 45)
Shape after selecting period (5834, 47)

Shape of data subset:  (5453, 47)

% of cases considered after taking subset:  93.46931779225231

AG list and frequencies
Merged Finance Support IT BSA - Finance    3182
IT BSA - Billing C&C                       1171
RevOps Support                              362
Bus - Billing C&C                           203
IT BSA - Singleview Ops                     170
IT BSA - Vertex                             149
Global DBA Support                           74
Hyperion Team                                54
IT BSA - BI Team                             52
Singleview Admin                             23
IT BSA - Client Services                     13
Name: ag, dtype: int64

AG to codes mapping
{0: 'Bus - Billing C&C', 1: 'Global DBA Support', 2: 'Hyperion Team', 3: 'I
T BSA - BI Team', 4: 'IT BSA - Billing C&C', 5: 'IT BSA - Client Services',
6: 'IT BSA - Singleview Ops', 7: 'IT BSA - Vertex', 8: 'Merged Finance Supp
ort IT BSA - Finance', 9: 'RevOps Support', 10: 'Singleview Admin'}
```

In [5]:
```
df.columns
```

Out[5]:
```
Index(['Unnamed: 0', 'number', 'state', 'u_region', 'u_business_priority',
       'u_classification', 'urgency', 'assigned_to', 'opened_at',
       'u_closure_category', 'u_requester', 'u_requested_by_date',
       'short_description', 'description', 'cmdb_ci', 'u_sla_breached',
       'u_sla_breached_reason', 'sla_due', 'sys_updated_on', 'comments',
       'u_bsa_comments', 'u_business_comments', 'u_developer_comments',
       'u_tech_lead_comments', 'work_notes', 'ag_old',
       'u_comments_and_work_notes', 'u_problem_code', 'u_problem_descriptio
n',
       'u_previous_assignment_groups', 'Requester Person ID',
       'Requester User Id', 'Requester Full Name', 'Requester Grade',
       'Requester Supervisor', 'Requester Cost Center Descr',
       'Requester Location Desc', 'Assigned To Person ID',
       'Assigned To User Id', 'Assigned To Full Name', 'Assigned To Grade',
       'Assigned To Supervisor', 'Assigned To Cost Center Descr',
       'Assigned To Location Desc', 'cln_desc1', 'ag', 'fulltext'],
      dtype='object')
```

In [6]:
```
d = df[['u_requester', 'Requester Grade', 'Requester Supervisor', 'Requeste
r Cost Center Descr', 'Requester Location Desc', 'ag', 'fulltext']].copy()
```

In [8]:
```
d.to_csv('df.csv')
```

In [ ]:
```

```

Prep text features

Create important text features

Test train split

In [179]:
```
#Split into test and train sets
test = df[df['opened_at'].dt.to_period('M').astype(str).isin(test_period)].
copy()
train = df.drop(test.index, axis=0)
print()
print('Train shape:', train.shape, 'Test shape:', test.shape)

#Remove stop words in English when creating tf idf vector and create train
 set
vectorizer = TfidfVectorizer(stop_words='english')
tfidf_fit = vectorizer.fit(train['fulltext'].values)
train_tfidf = vectorizer.transform(train['fulltext'].values)
print(train_tfidf.shape)

#Transform test set into tf idf
docs_new = test['fulltext'].values
X_new_tfidf = vectorizer.transform(docs_new)
print()
print('Shape of test tf idf: ', X_new_tfidf.shape)
```

```
Train shape: (4679, 47) Test shape: (774, 47)
(4679, 49076)

Shape of test tf idf:  (774, 49076)
```

In [180]:
```
#Run RF on tf idf and fit and find important features
m = RandomForestClassifier(n_estimators=1000, n_jobs=-1)
m.fit(train_tfidf, train['ag'])

#Use the feature importance to find the most important words
feature_importance = pd.DataFrame({'Feature' : vectorizer.get_feature_names
(), 'Importance' : m.feature_importances_})
feature_importance.sort_values('Importance', ascending=False, inplace=True)
```

In [181]:
```
# Create customer stop words
#Consider words with importance less than 0.0001 as unimportant and remove
 them from tf idf
words_to_remove = feature_importance[feature_importance['Importance'] < 0.0
001]['Feature']

#Add words to remove to stop words and create new tf idf
from sklearn.feature_extraction import text
from sklearn.feature_extraction.text import TfidfVectorizer
my_stop_words = text.ENGLISH_STOP_WORDS.union(words_to_remove)
len(my_stop_words)
```

Out[181]:
```
47892
```

In [182]:
```
#Split into test and train sets and create test and train tf idfs with redu
ced words
test = df[df['opened_at'].dt.to_period('M').astype(str).isin(test_period)].
copy()
train = df.drop(test.index, axis=0)
print()
print('Train shape:', train.shape, 'Test shape:', test.shape)

#Remove stop words in English when creating tf idf vector and create train
 set
vectorizer = TfidfVectorizer(stop_words=my_stop_words)
tfidf_fit = vectorizer.fit(train['fulltext'].values)
train_tfidf = vectorizer.transform(train['fulltext'].values)
print(train_tfidf.shape)

#Transform test set into tf idf
docs_new = test['fulltext'].values
X_new_tfidf = vectorizer.transform(docs_new)
print()
print('Shape of test tf idf: ', X_new_tfidf.shape)
train_tfidf.shape, X_new_tfidf.shape
```

```
Train shape: (4679, 47) Test shape: (774, 47)
(4679, 1502)

Shape of test tf idf:  (774, 1502)
```

Out[182]:
```
((4679, 1502), (774, 1502))
```

Prep column features

In [183]:
```
#Choose columns needed
to_keep = ['u_classification','Requester Grade','Requester Supervisor','Req
uester Cost Center Descr','Requester Location Desc','opened_at','ag']
df_feature = df[to_keep].copy()

#Change opened_at to date parts
#add_datepart(df_feature, 'opened_at', drop=False)
```

In [184]:
```
df_feature.shape
```

Out[184]: (5453, 7)

In [185]:
```
df_feature_ohe = pd.get_dummies(df_feature,columns=['u_classification','Requester Grade','Requester Supervisor','Requester Cost Center Descr','Requester Location Desc'])
```

In [186]:
```
df_feature_ohe.shape
```

Out[186]: (5453, 935)

In [187]:
```
#Split into test and train sets
test = df_feature_ohe[df_feature_ohe['opened_at'].dt.to_period('M').astype(str).isin(test_period)].copy()
train = df_feature_ohe.drop(test.index, axis=0)
print()
print('Train shape:', train.shape, 'Test shape:', test.shape)

test.drop('opened_at', axis=1, inplace=True)
train.drop('opened_at', axis=1, inplace=True)

y_test = test['ag']
test.drop('ag', axis=1, inplace=True)
y_train = train['ag']
train.drop('ag', axis=1, inplace=True)

train.shape, y_train.shape, test.shape, y_test.shape
```

```
Train shape: (4679, 935) Test shape: (774, 935)
```

Out[187]: ((4679, 933), (4679,), (774, 933), (774,))

Concat text and column features

In [188]:
```
#Concatenate train tfidf and train set to create full training set
train_tfidf_df = pd.DataFrame(train_tfidf.todense())
train_tfidf_df.reset_index(drop=True, inplace=True)
train.reset_index(drop=True, inplace=True)
train_full = pd.concat([train_tfidf_df, train], axis=1)

#Concatenate test tfidf and test set to create full training set
test_tfidf_df = pd.DataFrame(X_new_tfidf.todense())
test_tfidf_df.reset_index(drop=True, inplace=True)
test.reset_index(drop=True, inplace=True)
test_full = pd.concat([test_tfidf_df, test], axis=1)

train_full.shape, test_full.shape
```

Out[188]: ((4679, 2435), (774, 2435))

Run algorithms and save reports

using RandomForest

In [189]:
```
#Run RF on tf idf and fit
m = RandomForestClassifier(n_estimators=1000, n_jobs=-1)
m.fit(train_full, y_train)

#Do predictions
pred_label = m.predict(test_full)
pred_probs = m.predict_proba(test_full)
pred_df = pd.DataFrame(pred_probs)

pred_df['first_max_label'] = pred_label
pred_df['first_max_probs'] = pred_probs.max(axis=1)

second_label = []
for i in range(0,len(pred_probs)):
    second_label.append(np.argsort(-pred_probs)[i][1])
pred_df['second_max_label'] = second_label

probs_list = pred_probs.copy()
second_probs = []
for j in range(0,len(probs_list)):
    probs_list[j].sort()
    second_probs.append(probs_list[j][-2])
pred_df['second_max_probs'] = second_probs
pred_df['actual'] = y_test.values
pred_df['model'] = 'all_rf'

#Calculate accuracy
model_params.update({'Accuracy': accuracy_score(y_test, pred_label)})
model_params_df = pd.DataFrame.from_dict(model_params, orient='index').T
model_params_df['model'] = 'all_rf'

#Use the feature importance to find the most important words
feature_importance = pd.DataFrame({'Feature' : vectorizer.get_feature_names
() + list(train.columns.values), 'Importance' : m.feature_importances_})
feature_importance.sort_values('Importance', ascending=False, inplace=True)
feature_importance['model'] = 'all_rf'

#Confusion matrix analysis
cm = confusion_matrix(y_test, pred_label)

#Create cm df
a = []
p = []
c = []
t = []
for i in range(0,cm.shape[0]):
    for j in range(0,cm.shape[1]):
        a.append(i)
        p.append(j)
        c.append(cm[i][j])
        t.append(cm[i].sum())

cm_df = pd.DataFrame({'actual': a, 'predicted':p, 'count': c, 'total_actua
l':t})
cm_df['count%'] = cm_df['count'] * 100 / cm_df['total_actual']
```

```
#Get the code to cat mapping as df
class_to_cat_df = pd.DataFrame.from_dict(class_to_cat_mapping, orient='inde
x')
class_to_cat_df = class_to_cat_df.reset_index()
class_to_cat_df.columns = ['code','ag']

#Merge with confusion df to get names of AGs
confusion = cm_df.merge(class_to_cat_df, left_on='actual', right_on='code',
how='left')
confusion = confusion.merge(class_to_cat_df, left_on='predicted', right_on=
'code', how='left')
confusion = confusion[['ag_x', 'ag_y', 'count', 'total_actual', 'count%']].
copy()
confusion.columns = ['actual', 'predicted', 'count', 'total_actual', 'coun
t%']
#confusion['count%'] = confusion['count']*100/cm.sum()
confusion['model'] = 'all_rf'

with pd.ExcelWriter('all_rf.xlsx') as writer:  # doctest: +SKIP
    pred_df.to_excel(writer, sheet_name='probability')
    feature_importance.head(30).to_excel(writer, sheet_name='feature import
ance')
    model_params_df.to_excel(writer, sheet_name='parameters')
    confusion.to_excel(writer, sheet_name='confusion matrix')
```

Using XGBoost

In [190]:
```
#Run XGB on tf idf and fit
m = XGBClassifier(learning_rate =0.1, n_estimators=1000, max_depth=5, min_c
hild_weight=3, gamma=0.1, subsample=0.75, colsample_bytree=0.7, objective=
'multi:softmax', nthread=4, seed=27, reg_alpha=1 ,n_jobs=-1)
m.fit(train_full, y_train)

#Do predictions
pred_label = m.predict(test_full)
pred_probs = m.predict_proba(test_full)
pred_df = pd.DataFrame(pred_probs)

pred_df['first_max_label'] = pred_label
pred_df['first_max_probs'] = pred_probs.max(axis=1)

second_label = []
for i in range(0,len(pred_probs)):
    second_label.append(np.argsort(-pred_probs)[i][1])
pred_df['second_max_label'] = second_label

probs_list = pred_probs.copy()
second_probs = []
for j in range(0,len(probs_list)):
    probs_list[j].sort()
    second_probs.append(probs_list[j][-2])
pred_df['second_max_probs'] = second_probs
pred_df['actual'] = y_test.values
pred_df['model'] = 'all_xgb'

#Calculate accuracy
model_params.update({'Accuracy': accuracy_score(y_test, pred_label)})
model_params_df = pd.DataFrame.from_dict(model_params, orient='index').T
model_params_df['model'] = 'all_xgb'

#Use the feature importance to find the most important words
feature_importance = pd.DataFrame({'Feature' : vectorizer.get_feature_names
() + list(train.columns.values), 'Importance' : m.feature_importances_})
feature_importance.sort_values('Importance', ascending=False, inplace=True)
feature_importance['model'] = 'all_xgb'

#Confusion matrix analysis
cm = confusion_matrix(y_test, pred_label)

#Create cm df
a = []
p = []
c = []
t = []
for i in range(0,cm.shape[0]):
    for j in range(0,cm.shape[1]):
        a.append(i)
        p.append(j)
        c.append(cm[i][j])
        t.append(cm[i].sum())

cm_df = pd.DataFrame({'actual': a, 'predicted':p, 'count': c, 'total_actua
l':t})
```

```
cm_df['count%'] = cm_df['count'] * 100 / cm_df['total_actual']

#Get the code to cat mapping as df
class_to_cat_df = pd.DataFrame.from_dict(class_to_cat_mapping, orient='inde
x')
class_to_cat_df = class_to_cat_df.reset_index()
class_to_cat_df.columns = ['code','ag']

#Merge with confusion df to get names of AGs
confusion = cm_df.merge(class_to_cat_df, left_on='actual', right_on='code',
how='left')
confusion = confusion.merge(class_to_cat_df, left_on='predicted', right_on=
'code', how='left')
confusion = confusion[['ag_x', 'ag_y', 'count', 'total_actual', 'count%']].
copy()
confusion.columns = ['actual', 'predicted', 'count', 'total_actual', 'coun
t%']
#confusion['count%'] = confusion['count']*100/cm.sum()
confusion['model'] = 'all_xgb'

with pd.ExcelWriter('all_xgb.xlsx') as writer:  # doctest: +SKIP
    pred_df.to_excel(writer, sheet_name='probability')
    feature_importance.head(30).to_excel(writer, sheet_name='feature import
ance')
    model_params_df.to_excel(writer, sheet_name='parameters')
    confusion.to_excel(writer, sheet_name='confusion matrix')
```

Using LightGBM

In [191]:
```
m = LGBMModel(objective='multiclass', n_estimators=1000, n_jobs=-1, num_cla
ss=11, class_weight='balanced', importance_type='gain')
m.fit(train_full, y_train)

#Do predictions
pred_probs = m.predict(test_full)
pred_label = pred_probs.argmax(axis=1)
pred_df = pd.DataFrame(pred_probs)

pred_df['first_max_label'] = pred_label
pred_df['first_max_probs'] = pred_probs.max(axis=1)

second_label = []
for i in range(0,len(pred_probs)):
    second_label.append(np.argsort(-pred_probs)[i][1])
pred_df['second_max_label'] = second_label

probs_list = pred_probs.copy()
second_probs = []
for j in range(0,len(probs_list)):
    probs_list[j].sort()
    second_probs.append(probs_list[j][-2])
pred_df['second_max_probs'] = second_probs
pred_df['actual'] = y_test.values
pred_df['model'] = 'all_lgb'

#Calculate accuracy
model_params.update({'Accuracy': accuracy_score(y_test, pred_label)})
model_params_df = pd.DataFrame.from_dict(model_params, orient='index').T
model_params_df['model'] = 'all_lgb'

#Use the feature importance to find the most important words
feature_importance = pd.DataFrame({'Feature' : vectorizer.get_feature_names
() + list(train.columns.values), 'Importance' : m.feature_importances_})
feature_importance.sort_values('Importance', ascending=False, inplace=True)
feature_importance['model'] = 'all_lgb'

#Confusion matrix analysis
cm = confusion_matrix(y_test, pred_label)

#Create cm df
a = []
p = []
c = []
t = []
for i in range(0,cm.shape[0]):
    for j in range(0,cm.shape[1]):
        a.append(i)
        p.append(j)
        c.append(cm[i][j])
        t.append(cm[i].sum())

cm_df = pd.DataFrame({'actual': a, 'predicted':p, 'count': c, 'total_actua
l':t})
cm_df['count%'] = cm_df['count'] * 100 / cm_df['total_actual']
```

```
#Get the code to cat mapping as df
class_to_cat_df = pd.DataFrame.from_dict(class_to_cat_mapping, orient='inde
x')
class_to_cat_df = class_to_cat_df.reset_index()
class_to_cat_df.columns = ['code','ag']

#Merge with confusion df to get names of AGs
confusion = cm_df.merge(class_to_cat_df, left_on='actual', right_on='code',
how='left')
confusion = confusion.merge(class_to_cat_df, left_on='predicted', right_on=
'code', how='left')
confusion = confusion[['ag_x', 'ag_y', 'count', 'total_actual', 'count%']].
copy()
confusion.columns = ['actual', 'predicted', 'count', 'total_actual', 'coun
t%']
#confusion['count%'] = confusion['count']*100/cm.sum()
confusion['model'] = 'all_lgb'

with pd.ExcelWriter('all_lgb.xlsx') as writer:  # doctest: +SKIP
    pred_df.to_excel(writer, sheet_name='probability')
    feature_importance.head(30).to_excel(writer, sheet_name='feature import
ance')
    model_params_df.to_excel(writer, sheet_name='parameters')
    confusion.to_excel(writer, sheet_name='confusion matrix')
```

Using Logistic regression

In [192]:
```
m = LogisticRegression(multi_class='multinomial', solver='newton-cg', max_i
ter=1000)
m.fit(train_full, y_train)

#Do predictions
pred_label = m.predict(test_full)
pred_probs = m.predict_proba(test_full)
pred_df = pd.DataFrame(pred_probs)

pred_df['first_max_label'] = pred_label
pred_df['first_max_probs'] = pred_probs.max(axis=1)

second_label = []
for i in range(0,len(pred_probs)):
    second_label.append(np.argsort(-pred_probs)[i][1])
pred_df['second_max_label'] = second_label

probs_list = pred_probs.copy()
second_probs = []
for j in range(0,len(probs_list)):
    probs_list[j].sort()
    second_probs.append(probs_list[j][-2])
pred_df['second_max_probs'] = second_probs
pred_df['actual'] = y_test.values
pred_df['model'] = 'all_log'

#Calculate accuracy
model_params.update({'Accuracy': accuracy_score(y_test, pred_label)})
model_params_df = pd.DataFrame.from_dict(model_params, orient='index').T
model_params_df['model'] = 'all_log'

'''
#Use the feature importance to find the most important words
feature_importance = pd.DataFrame({'Feature' : vectorizer.get_feature_names
(), 'Importance' : m.feature_importances_})
feature_importance.sort_values('Importance', ascending=False, inplace=True)
feature_importance['model'] = 'text_log'
'''
feature_importance = pd.DataFrame()

#Confusion matrix analysis
cm = confusion_matrix(y_test, pred_label)

#Create cm df
a = []
p = []
c = []
t = []
for i in range(0,cm.shape[0]):
    for j in range(0,cm.shape[1]):
        a.append(i)
        p.append(j)
        c.append(cm[i][j])
        t.append(cm[i].sum())

cm_df = pd.DataFrame({'actual': a, 'predicted':p, 'count': c, 'total_actua
```

```
l':t})
cm_df['count%'] = cm_df['count'] * 100 / cm_df['total_actual']

#Get the code to cat mapping as df
class_to_cat_df = pd.DataFrame.from_dict(class_to_cat_mapping, orient='inde
x')
class_to_cat_df = class_to_cat_df.reset_index()
class_to_cat_df.columns = ['code','ag']

#Merge with confusion df to get names of AGs
confusion = cm_df.merge(class_to_cat_df, left_on='actual', right_on='code',
how='left')
confusion = confusion.merge(class_to_cat_df, left_on='predicted', right_on=
'code', how='left')
confusion = confusion[['ag_x', 'ag_y', 'count', 'total_actual', 'count%']].
copy()
confusion.columns = ['actual', 'predicted', 'count', 'total_actual', 'coun
t%']
#confusion['count%'] = confusion['count']*100/cm.sum()
confusion['model'] = 'all_log'

with pd.ExcelWriter('all_log.xlsx') as writer:  # doctest: +SKIP
    pred_df.to_excel(writer, sheet_name='probability')
    feature_importance.head(30).to_excel(writer, sheet_name='feature import
ance')
    model_params_df.to_excel(writer, sheet_name='parameters')
    confusion.to_excel(writer, sheet_name='confusion matrix')
```

Using DecisionTreeClassifier

In [193]:
```
m = DecisionTreeClassifier(max_depth=23)
m.fit(train_full, y_train)

#Do predictions
pred_label = m.predict(test_full)
pred_probs = m.predict_proba(test_full)
pred_df = pd.DataFrame(pred_probs)

pred_df['first_max_label'] = pred_label
pred_df['first_max_probs'] = pred_probs.max(axis=1)

second_label = []
for i in range(0,len(pred_probs)):
    second_label.append(np.argsort(-pred_probs)[i][1])
pred_df['second_max_label'] = second_label

probs_list = pred_probs.copy()
second_probs = []
for j in range(0,len(probs_list)):
    probs_list[j].sort()
    second_probs.append(probs_list[j][-2])
pred_df['second_max_probs'] = second_probs
pred_df['actual'] = y_test.values
pred_df['model'] = 'all_dtc'

#Calculate accuracy
model_params.update({'Accuracy': accuracy_score(y_test, pred_label)})
model_params_df = pd.DataFrame.from_dict(model_params, orient='index').T
model_params_df['model'] = 'all_dtc'

'''
#Use the feature importance to find the most important words
feature_importance = pd.DataFrame({'Feature' : vectorizer.get_feature_names
(), 'Importance' : m.feature_importances_})
feature_importance.sort_values('Importance', ascending=False, inplace=True)
feature_importance['model'] = 'text_dtc'
'''
feature_importance = pd.DataFrame()

#Confusion matrix analysis
cm = confusion_matrix(y_test, pred_label)

#Create cm df
a = []
p = []
c = []
t = []
for i in range(0,cm.shape[0]):
    for j in range(0,cm.shape[1]):
        a.append(i)
        p.append(j)
        c.append(cm[i][j])
        t.append(cm[i].sum())

cm_df = pd.DataFrame({'actual': a, 'predicted':p, 'count': c, 'total_actua
l':t})
```

```
cm_df['count%'] = cm_df['count'] * 100 / cm_df['total_actual']

#Get the code to cat mapping as df
class_to_cat_df = pd.DataFrame.from_dict(class_to_cat_mapping, orient='inde
x')
class_to_cat_df = class_to_cat_df.reset_index()
class_to_cat_df.columns = ['code','ag']

#Merge with confusion df to get names of AGs
confusion = cm_df.merge(class_to_cat_df, left_on='actual', right_on='code',
how='left')
confusion = confusion.merge(class_to_cat_df, left_on='predicted', right_on=
'code', how='left')
confusion = confusion[['ag_x', 'ag_y', 'count', 'total_actual', 'count%']].
copy()
confusion.columns = ['actual', 'predicted', 'count', 'total_actual', 'coun
t%']
#confusion['count%'] = confusion['count']*100/cm.sum()
confusion['model'] = 'all_dtc'

with pd.ExcelWriter('all_dtc.xlsx') as writer:  # doctest: +SKIP
    pred_df.to_excel(writer, sheet_name='probability')
    feature_importance.head(30).to_excel(writer, sheet_name='feature import
ance')
    model_params_df.to_excel(writer, sheet_name='parameters')
    confusion.to_excel(writer, sheet_name='confusion matrix')
```

Using SVM

In [194]:
```
#m = LinearSVC()
m = SVC(gamma='scale', decision_function_shape='ovo', probability=True)
m.fit(train_full, y_train)

#Do predictions
pred_label = m.predict(test_full)
pred_probs = m.predict_proba(test_full)
pred_df = pd.DataFrame(pred_probs)

pred_df['first_max_label'] = pred_label
pred_df['first_max_probs'] = pred_probs.max(axis=1)

second_label = []
for i in range(0,len(pred_probs)):
    second_label.append(np.argsort(-pred_probs)[i][1])
pred_df['second_max_label'] = second_label

probs_list = pred_probs.copy()
second_probs = []
for j in range(0,len(probs_list)):
    probs_list[j].sort()
    second_probs.append(probs_list[j][-2])
pred_df['second_max_probs'] = second_probs
pred_df['actual'] = y_test.values
pred_df['model'] = 'all_svm'

#Calculate accuracy
model_params.update({'Accuracy': accuracy_score(y_test, pred_label)})
model_params_df = pd.DataFrame.from_dict(model_params, orient='index').T
model_params_df['model'] = 'all_svm'

'''
#Use the feature importance to find the most important words
feature_importance = pd.DataFrame({'Feature' : vectorizer.get_feature_names
(), 'Importance' : m.feature_importances_})
feature_importance.sort_values('Importance', ascending=False, inplace=True)
feature_importance['model'] = 'text_svm'
'''
feature_importance = pd.DataFrame()

#Confusion matrix analysis
cm = confusion_matrix(y_test, pred_label)

#Create cm df
a = []
p = []
c = []
t = []
for i in range(0,cm.shape[0]):
    for j in range(0,cm.shape[1]):
        a.append(i)
        p.append(j)
        c.append(cm[i][j])
        t.append(cm[i].sum())

cm_df = pd.DataFrame({'actual': a, 'predicted':p, 'count': c, 'total_actua
```

```
l':t})
cm_df['count%'] = cm_df['count'] * 100 / cm_df['total_actual']

#Get the code to cat mapping as df
class_to_cat_df = pd.DataFrame.from_dict(class_to_cat_mapping, orient='inde
x')
class_to_cat_df = class_to_cat_df.reset_index()
class_to_cat_df.columns = ['code','ag']

#Merge with confusion df to get names of AGs
confusion = cm_df.merge(class_to_cat_df, left_on='actual', right_on='code',
how='left')
confusion = confusion.merge(class_to_cat_df, left_on='predicted', right_on=
'code', how='left')
confusion = confusion[['ag_x', 'ag_y', 'count', 'total_actual', 'count%']].
copy()
confusion.columns = ['actual', 'predicted', 'count', 'total_actual', 'coun
t%']
#confusion['count%'] = confusion['count']*100/cm.sum()
confusion['model'] = 'all_svm'

with pd.ExcelWriter('all_svm.xlsx') as writer:  # doctest: +SKIP
    pred_df.to_excel(writer, sheet_name='probability')
    feature_importance.head(30).to_excel(writer, sheet_name='feature import
ance')
    model_params_df.to_excel(writer, sheet_name='parameters')
    confusion.to_excel(writer, sheet_name='confusion matrix')
```

Tensorflow

In [169]:
```
train_full.shape, test_full.shape
```

Out[169]:
```
((4679, 2446), (774, 2446))
```

In [ ]:
```

```

In [170]:
```
# A utility method to create a tf.data dataset from a Pandas Dataframe
def df_to_dataset(dataframe, shuffle=True, batch_size=32):
  dataframe = dataframe.copy()
  labels = dataframe.pop('ag')
  ds = tf.data.Dataset.from_tensor_slices((dict(dataframe), labels))
  if shuffle:
    ds = ds.shuffle(buffer_size=len(dataframe))
  ds = ds.batch(batch_size)
  return ds
```

In [172]:
```
batch_size = 5
train_ds = df_to_dataset(train_full, batch_size=batch_size)
val_ds = df_to_dataset(val, shuffle=False, batch_size=batch_size)
test_ds = df_to_dataset(test_full, shuffle=False, batch_size=batch_size)
```

```
---------------------------------------------------------------------------
KeyError                                  Traceback (most recent call last)
~\AppData\Local\Continuum\anaconda3\lib\site-packages\pandas\core\indexes\b
ase.py in get_loc(self, key, method, tolerance)
   2656             try:
-> 2657                 return self._engine.get_loc(key)
   2658             except KeyError:

pandas/_libs/index.pyx in pandas._libs.index.IndexEngine.get_loc()

pandas/_libs/index.pyx in pandas._libs.index.IndexEngine.get_loc()

pandas/_libs/hashtable_class_helper.pxi in pandas._libs.hashtable.PyObjectH
ashTable.get_item()

pandas/_libs/hashtable_class_helper.pxi in pandas._libs.hashtable.PyObjectH
ashTable.get_item()

KeyError: 'ag'

During handling of the above exception, another exception occurred:

KeyError                                  Traceback (most recent call last)
<ipython-input-172-03cb3aaa1186> in <module>
      1 batch_size = 5
----> 2 train_ds = df_to_dataset(train_full, batch_size=batch_size)
      3 val_ds = df_to_dataset(val, shuffle=False, batch_size=batch_size)
      4 test_ds = df_to_dataset(test_full, shuffle=False, batch_size=batch_
size)

<ipython-input-170-6d79a9443fd4> in df_to_dataset(dataframe, shuffle, batch
_size)
      2 def df_to_dataset(dataframe, shuffle=True, batch_size=32):
      3   dataframe = dataframe.copy()
----> 4   labels = dataframe.pop('ag')
      5   ds = tf.data.Dataset.from_tensor_slices((dict(dataframe), labels)
)
      6   if shuffle:

~\AppData\Local\Continuum\anaconda3\lib\site-packages\pandas\core\generic.p
y in pop(self, item)
    807         3  monkey        NaN
    808         """
--> 809         result = self[item]
    810         del self[item]
    811         try:

~\AppData\Local\Continuum\anaconda3\lib\site-packages\pandas\core\frame.py
 in __getitem__(self, key)
   2925             if self.columns.nlevels > 1:
   2926                 return self._getitem_multilevel(key)
-> 2927             indexer = self.columns.get_loc(key)
   2928             if is_integer(indexer):
   2929                 indexer = [indexer]

~\AppData\Local\Continuum\anaconda3\lib\site-packages\pandas\core\indexes\b
ase.py in get_loc(self, key, method, tolerance)
```

```
   2657                 return self._engine.get_loc(key)
   2658             except KeyError:
-> 2659                 return self._engine.get_loc(self._maybe_cast_indexe
r(key))
   2660         indexer = self.get_indexer([key], method=method, tolerance=
tolerance)
   2661         if indexer.ndim > 1 or indexer.size > 1:

pandas/_libs/index.pyx in pandas._libs.index.IndexEngine.get_loc()

pandas/_libs/index.pyx in pandas._libs.index.IndexEngine.get_loc()

pandas/_libs/hashtable_class_helper.pxi in pandas._libs.hashtable.PyObjectH
ashTable.get_item()

pandas/_libs/hashtable_class_helper.pxi in pandas._libs.hashtable.PyObjectH
ashTable.get_item()

KeyError: 'ag'
```

Blending model outputs

Import all predictions

In [197]:
```
all_rf = pd.read_excel('all_rf.xlsx')
all_xgb = pd.read_excel('all_xgb.xlsx')
all_lgb = pd.read_excel('all_lgb.xlsx')
all_log = pd.read_excel('all_log.xlsx')
all_svm = pd.read_excel('all_svm.xlsx')
all_dtc = pd.read_excel('all_dtc.xlsx')
```

In [198]:
```
label_df = pd.DataFrame({'all_rf':all_rf['first_max_label'],
                         'all_xgb':all_xgb['first_max_label'],
                         'all_lgb':all_lgb['first_max_label'],
                         'all_dtc' : all_dtc['first_max_label'],
                         'all_log' : all_log['first_max_label'],
                         'all_svm' : all_svm['first_max_label']})
```

Mode of labels

In [199]:
```
label_df['label'] = label_df.mode(axis=1)[0].astype(int)
label_df.head()
```

Out[199]:

	all_rf	all_xgb	all_lgb	all_dtc	all_log	all_svm	label
0	7	7	7	8	7	5	7
1	5	0	5	5	5	5	5
2	7	7	7	7	7	7	7
3	5	5	5	5	5	5	5
4	5	5	5	7	8	8	5

In [200]:
```
accuracy_score(y_test, label_df['label'])
```

Out[200]: 0.789405684754522

Consolidate reports

In [201]:
```
probability_consolidated = pd.read_excel('all_rf.xlsx', sheet_name='probabi
lity')
probability_consolidated = probability_consolidated.append([pd.read_excel(
'all_lgb.xlsx', sheet_name='probability'),
                                                            pd.read_excel('a
ll_xgb.xlsx', sheet_name='probability'),
                                                            pd.read_excel('a
ll_log.xlsx', sheet_name='probability'),
                                                            pd.read_excel('a
ll_dtc.xlsx', sheet_name='probability'),
                                                            pd.read_excel('a
ll_svm.xlsx', sheet_name='probability')])
probability_consolidated.drop('Unnamed: 0', axis=1, inplace=True)
```

In [202]:
```
feature_consolidated = pd.read_excel('all_rf.xlsx', sheet_name='feature imp
ortance')
feature_consolidated = feature_consolidated.append([pd.read_excel('all_lgb.
xlsx', sheet_name='feature importance'),
                                                    pd.read_excel('all_xgb.
xlsx', sheet_name='feature importance'),
                                                    pd.read_excel('all_log.
xlsx', sheet_name='feature importance'),
                                                    pd.read_excel('all_dtc.
xlsx', sheet_name='feature importance'),
                                                    pd.read_excel('all_svm.
xlsx', sheet_name='feature importance')])
feature_consolidated.drop('Unnamed: 0', axis=1, inplace=True)
```

In [203]:
```
parameters_consolidated = pd.read_excel('all_rf.xlsx', sheet_name='parameters')
parameters_consolidated = parameters_consolidated.append([pd.read_excel('all_lgb.xlsx', sheet_name='parameters'),
                                                          pd.read_excel('all_xgb.xlsx', sheet_name='parameters'),
                                                          pd.read_excel('all_log.xlsx', sheet_name='parameters'),
                                                          pd.read_excel('all_dtc.xlsx', sheet_name='parameters'),
                                                          pd.read_excel('all_svm.xlsx', sheet_name='parameters')])
parameters_consolidated.drop('Unnamed: 0', axis=1, inplace=True)
```

In [204]:
```
confusion_consolidated = pd.read_excel('all_rf.xlsx', sheet_name='confusion matrix')
confusion_consolidated = confusion_consolidated.append([pd.read_excel('all_lgb.xlsx', sheet_name='confusion matrix'),
                                                        pd.read_excel('all_xgb.xlsx', sheet_name='confusion matrix'),
                                                        pd.read_excel('all_log.xlsx', sheet_name='confusion matrix'),
                                                        pd.read_excel('all_dtc.xlsx', sheet_name='confusion matrix'),
                                                        pd.read_excel('all_svm.xlsx', sheet_name='confusion matrix'),])
confusion_consolidated.drop('Unnamed: 0', axis=1, inplace=True)
```

In [205]:
```
with pd.ExcelWriter('consolidated reports - text and features.xlsx') as writer:
    probability_consolidated.to_excel(writer, sheet_name='probability')
    feature_consolidated.to_excel(writer, sheet_name='feature importance')
    parameters_consolidated.to_excel(writer, sheet_name='parameters')
    confusion_consolidated.to_excel(writer, sheet_name='confusion matrix')
```

In [ ]:
```

```