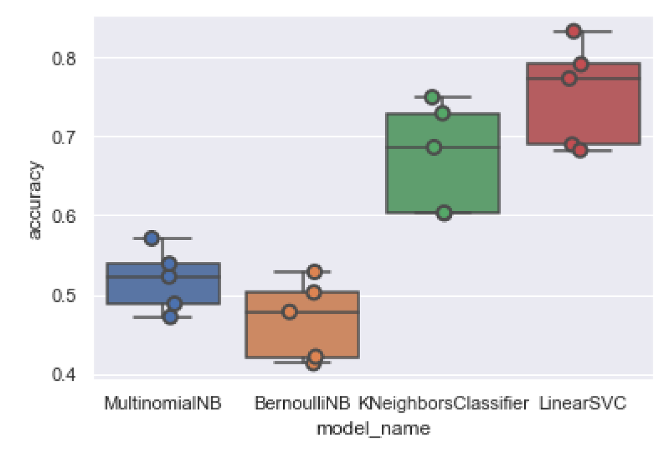
In [21]:
```python
# Remove duplicate words in the text entry
# Duplicate words introduce artifical bias in the context of our data se
t

uniqueConcat = []
for string in trainText.str.split():
    uniqueWords = []
    for word in string:
        if word not in uniqueWords:
            uniqueWords.append(word)
    uniqueString = ' '.join(uniqueWords)
    uniqueConcat.append(uniqueString)

uniqueConcat = pd.Series(uniqueConcat)

full_data['text'] = uniqueConcat
```

In [24]:
```python
# Set Up TF-IDF Before Fitting Classifier

from sklearn.feature_extraction.text import TfidfVectorizer

tfidf = TfidfVectorizer(sublinear_tf=True,
                        min_df=10,
                        norm='l2',
                        encoding='latin-1',
                        ngram_range=(1, 2),
                        stop_words='english'
                       )

features = tfidf.fit_transform(full_data['text']).toarray()
labels = full_data['y_labels']
```

In [138]:
```python
# Evaluate Several Classifiers' Speed and Accuracy

from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier
from sklearn.svm import LinearSVC

from sklearn.neighbors import KNeighborsClassifier
from sklearn.naive_bayes import GaussianNB, MultinomialNB, BernoulliNB
from sklearn.linear_model import LogisticRegression, SGDClassifier
from sklearn.svm import SVC, LinearSVC, NuSVC

from sklearn.model_selection import cross_val_score

from sklearn.metrics import accuracy_score, classification_report

import time

models = [
#     RandomForestClassifier(n_estimators = 200, max_depth = 2),
    MultinomialNB(),
#     GaussianNB(),
    BernoulliNB(),
    KNeighborsClassifier(n_neighbors=1),
#     LogisticRegression(),
    LinearSVC()
#     SGDClassifier(),
#     SVC(),
#     NuSVC()
]

times = {}

CV = 5
cv_df = pd.DataFrame(index=range(CV * len(models)))
entries = []
for model in models:
    start_time = time.time()

    model_name = model.__class__.__name__
    accuracies = cross_val_score(model, features, labels, scoring='accur
acy', cv=CV)

    end_time = time.time() - start_time
    times[model_name] = end_time

    for fold_idx, accuracy in enumerate(accuracies):
        entries.append((model_name, fold_idx, accuracy))
cv_df = pd.DataFrame(entries, columns=['model_name', 'fold_idx', 'accura
cy'])

import seaborn as sns
sns.boxplot(x='model_name', y='accuracy', data=cv_df)
sns.stripplot(x='model_name', y='accuracy', data=cv_df,
              size=8, jitter=True, edgecolor="gray", linewidth=2)
plt.figure(figsize=(10, 10))
plt.show()

print('')
print(CV, '- fold Cross Validation Run Times: ')
print('')

for model in list(times):
    print(model, ':', times[model], 'seconds')
```




```
5 - fold Cross Validation Run Times:

MultinomialNB : 3.6596739292144775 seconds
BernoulliNB : 6.0347511768341064 seconds
KNeighborsClassifier : 18.383825302124023 seconds
LinearSVC : 7.801565885543823 seconds
```

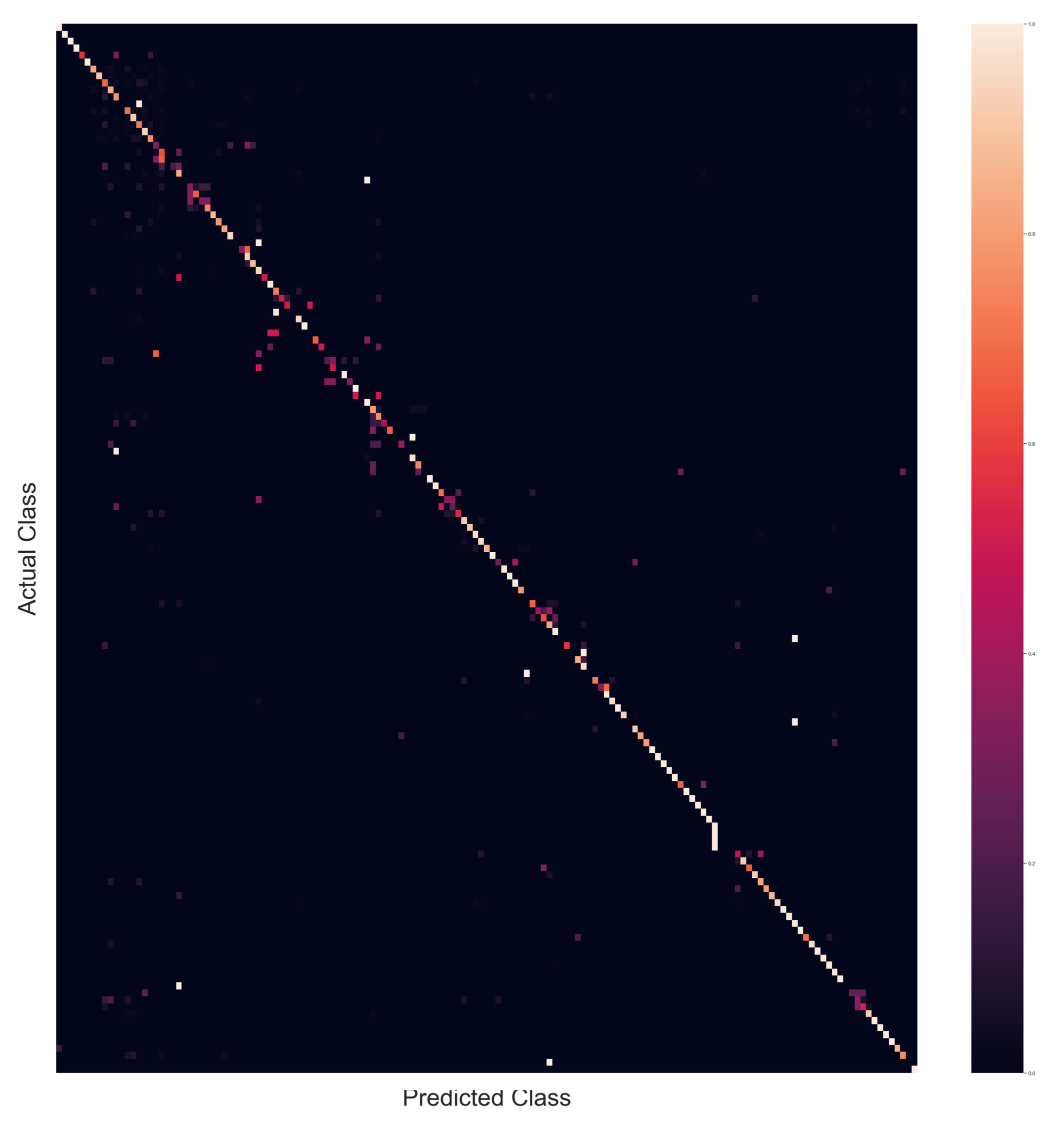
In [47]:
```python
# Create Detailed CV Accuracy Report
# Including precision, recall, f1, and support

from sklearn.model_selection import StratifiedKFold, KFold, RepeatedStra
tifiedKFold
from sklearn import preprocessing

k_folds = 5

skf = RepeatedStratifiedKFold(n_splits = k_folds, n_repeats = 2)

# Mask true Class identities for Confidentiality Purposes
label_mask = dict(zip(labels.unique(), np.arange(len(labels.unique()))))
labels = pd.Series([label_mask[label] for label in labels])

model_scores = []
adf = pd.DataFrame(columns = ['Class', 'precision', 'recall', 'average c
ount in test set'])
for train_index, test_index in skf.split(features, labels):
    X_train, X_test = features[train_index], features[test_index]
    y_train, y_test = labels.iloc[train_index], labels.iloc[test_index]

    model.fit(X_train, y_train)
    y_pred = model.predict(X_test)

    cv_score = model.score(X_test, y_test)
    model_scores.append(cv_score)

    accuracy_df = pd.DataFrame({'predicted' : y_pred,
                   'actual' : y_test })

    for Class in accuracy_df['actual'].unique():
        actual = accuracy_df[accuracy_df['actual'] == Class]['actual']
        predicted =  accuracy_df[accuracy_df['actual'] == Class]['predic
ted']

        true_pos = np.sum(actual == predicted)

        predicted_true_df = accuracy_df[accuracy_df['predicted'] == Clas
s]
        false_pos = np.sum(predicted_true_df['predicted'] != predicted_t
rue_df['actual'])

        actual_true_df = accuracy_df[accuracy_df['actual'] == Class]
        false_neg = np.sum(actual_true_df['actual'] != actual_true_df['p
redicted'])

        # High Precision indicates low rate of False Positives
        # i.e. Not OVERClassIFYING
        # Precision of NaN indicates NO Positive Predictions for that Cl
ass (to avoid ZeroDivisionError)

        if (true_pos + false_pos) == 0.0:
            precision = np.NaN
        else:
            precision = true_pos / (true_pos + false_pos)

        # Recall indicates the percent of entries in a Class that were c
orrectly identified
        # i.e. Not UNDERClassIFYING
        # Recall of NaN indicates Class is not present in the test set
 (to avoid ZeroDivisionError)

        if (true_pos + false_neg) == 0.0:
            recall = np.NaN
        else:
            recall = true_pos / (true_pos + false_neg)

        Class_stats = [Class, precision, recall, int(len(actual))]
        adf.loc[len(adf.index)] = Class_stats


adf['average count in test set'] = adf['average count in test set'].asty
pe(int)

Class_accuracy_df = adf.groupby(['Class']).agg(np.nanmean)

precision = Class_accuracy_df['precision']
recall = Class_accuracy_df['recall']
Class_accuracy_df['F1 score'] = 2 * (precision * recall) / (precision +
recall)

Class_accuracy_df.sort_values('average count in test set', ascending = F
alse, inplace = True)

Class_accuracy_df = Class_accuracy_df[['precision', 'recall', 'F1 score'
, 'average count in test set']]

print('Stratified', k_folds, '- Fold Cross Validation Accuracy : ', '\n'
, np.mean(model_scores))
display(Class_accuracy_df.dropna())
```

```
Stratified 5 - Fold Cross Validation Accuracy :
 0.8658949527326223
```

Class	precision	recall	F1 score	average count in test set
6.0	0.989229	0.962053	0.975452	171.2
1.0	0.969146	0.984564	0.976794	149.0
11.0	0.962732	0.988596	0.975493	114.0
0.0	0.986163	0.960927	0.973381	102.4
55.0	0.952737	0.973009	0.962766	74.2
...	...	...	...	...
35.0	0.900000	0.625000	0.737705	1.0
144.0	0.700000	0.500000	0.583333	1.0
136.0	1.000000	1.000000	1.000000	1.0
137.0	1.000000	0.700000	0.823529	1.0
138.0	0.629630	0.800000	0.704663	1.0

In [118]:
```python
# Visualize accuracy by class through confusion matrix heat map

import numpy as np; np.random.seed(0)
import seaborn as sns; sns.set_theme()
from sklearn.metrics import plot_confusion_matrix
from sklearn.metrics import confusion_matrix

conf_mat = confusion_matrix(y_test, y_pred, normalize = 'true')

uniform_data = np.random.rand(10, 12)
fig, ax = plt.subplots(figsize = (40,40))

ax = sns.heatmap(conf_mat,
                )

ax.set_xlabel('Predicted Class', fontsize = 50)
ax.set_ylabel('Actual Class', fontsize = 50)
```

Out[118]:



In [ ]:
```python
# MODEL OUTPUT ADDITIONS TO PROVIDE PRACTICAL INSIGHTS
```

In [123]:
```python
# Extract Keywords and Phrases for every class: Top-N most correlated
# Include with predictions to provide insights to the team

from sklearn.feature_selection import chi2
import numpy as np

keyword_uni = {}
keyword_bi = {}

N = 5
display_max = 5
counter = 0

for level in sorted(labels.unique()):
    features_chi2 = chi2(features, labels == level)
    indices = np.argsort(features_chi2[0])
    feature_names = np.array(tfidf.get_feature_names())[indices]

    unigrams = [v for v in feature_names if len(v.split(' ')) == 1]
    bigrams = [v for v in feature_names if len(v.split(' ')) == 2]

    n_best_unigrams = unigrams[-N:][::-1]
    n_best_bigrams = bigrams[-N:][::-1]

    if counter < display_max:
        print("# '{}':".format(level))
        print("  . Most correlated unigrams:\n. {}".format('\n. '.join(n
_best_unigrams)))
        print("  . Most correlated bigrams:\n. {}".format('\n. '.join(n_
best_bigrams)))
        counter += 1

    keyword_uni[level] = n_best_unigrams
    keyword_bi[level] = n_best_bigrams
```

In [82]:
```python
# Include Classifier's prediction probabilites for Top n most probable c
lasses
# Provides a "Confidence" score for every prediction and shows the next
 n most likely predictions for every entry
# Output Prediction Table can now be sorted by confidence

test_proba = model.predict_proba(input_features)
model_classes = model.classes_

confidence_series = []
n_classes = 2

for i in np.arange(len(inputDf)):
    top_classes = model_classes[test_proba[i, :].argsort()[::-1][:n_clas
ses]]
    class_probailities = np.sort(test_proba[i, :])[::-1][:n_classes]

    classes_with_probabilities = dict(zip(top_classes, class_probailitie
s))
    confidence_series.append(classes_with_probabilities)

confidence_series = pd.Series(confidence_series)

prob_series = pd.Series([list(inputDf['class_probabilities'][i].values
())[0] for i in np.arange(len(inputDf))])

index_sortby_prob = prob_series.argsort()[::-1]
```