AG News Topic Classification

CSML 1010 - Winter 2020 - Group 20

Tony Lee, Viswesh Krishnamurthy

YORK
UNIVERSITÉ
UNIVERSITY

PROBLEM SELECTION & DEFINITION

- A text classification problem was chosen from the website https://datasets.quantumstat.com/. We chose the AG News corpus dataset

- The goal of this project is to develop a text classifier model that can accept a news 'headline' and 'content' of the news to classify the news article into one of the 4 following categories
  - World (coded as 1)
  - Sports (2)
  - Business (3)
  - Sci/Tech (4)

YORK
UNIVERSITÉ
UNIVERSITY

PROJECT MILESTONE 1 – 18th Apr 2020

YORK
UNIVERSITÉ
UNIVERSITY

FEATURE ENGINEERING & MODELS - PLAN

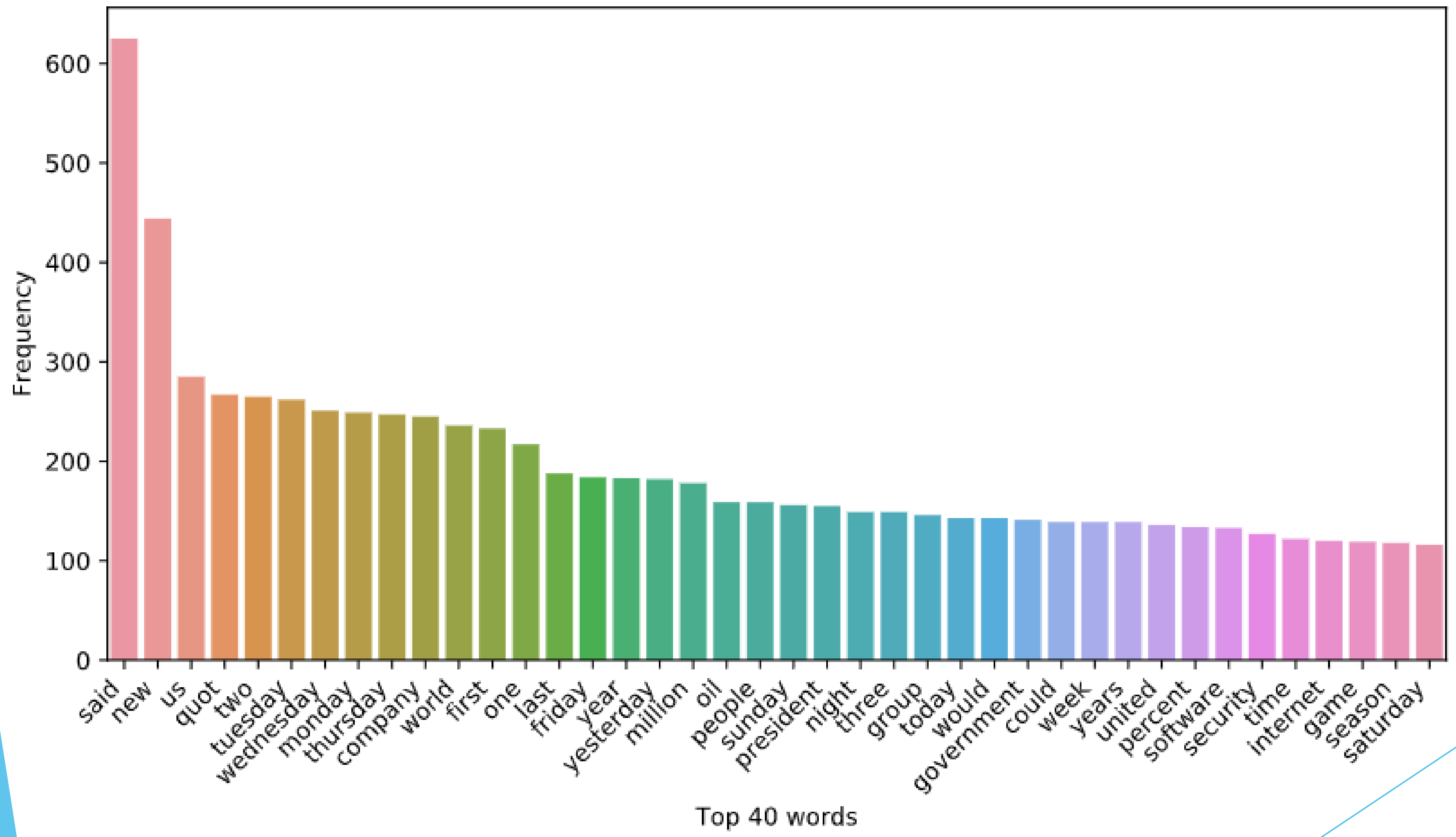
- We have performed feature engineering using the following methods
  - Bag of words, Bag of n-grams, Bag of characters
  - Tf/Idf, Tf/Idf Unigrams, Tf/Idf N-grams, Tf/Idf Characters
  - Word2Vec
- We built the first model with SVM with a sample data of 4000 rows
- We will continue to build more models as seen below,
  - Naive Bayes
  - Logistic Regression
  - Decision Trees
  - GBM (Gradient Boosting)

YORK UNIVERSITÉ UNIVERSITY

FEATURE ENGINEERING – BAG OF WORDS

```
# Use countvectorizer to get a vector of words

cv = CountVectorizer(min_df = 2, lowercase = True,
                     token_pattern=r'\b[A-Za-z]{2,}\b', ngram_range = (1, 1))
x_train_cv = cv.fit_transform(x_train.content_cleaned).toarray()
x_test_cv = cv.transform(x_test.content_cleaned).toarray()

# get all unique words in the corpus

bow_vocab = cv.get_feature_names()

# produce a dataframe including the feature names

x_train_bagofwords = pandas.DataFrame(x_train_cv, columns=bow_vocab)
x_test_bagofwords = pandas.DataFrame(x_test_cv, columns=bow_vocab)
x_train_bagofwords.head()
```

FEATURE ENGINEERING – BAG OF WORDS (Cont'd)

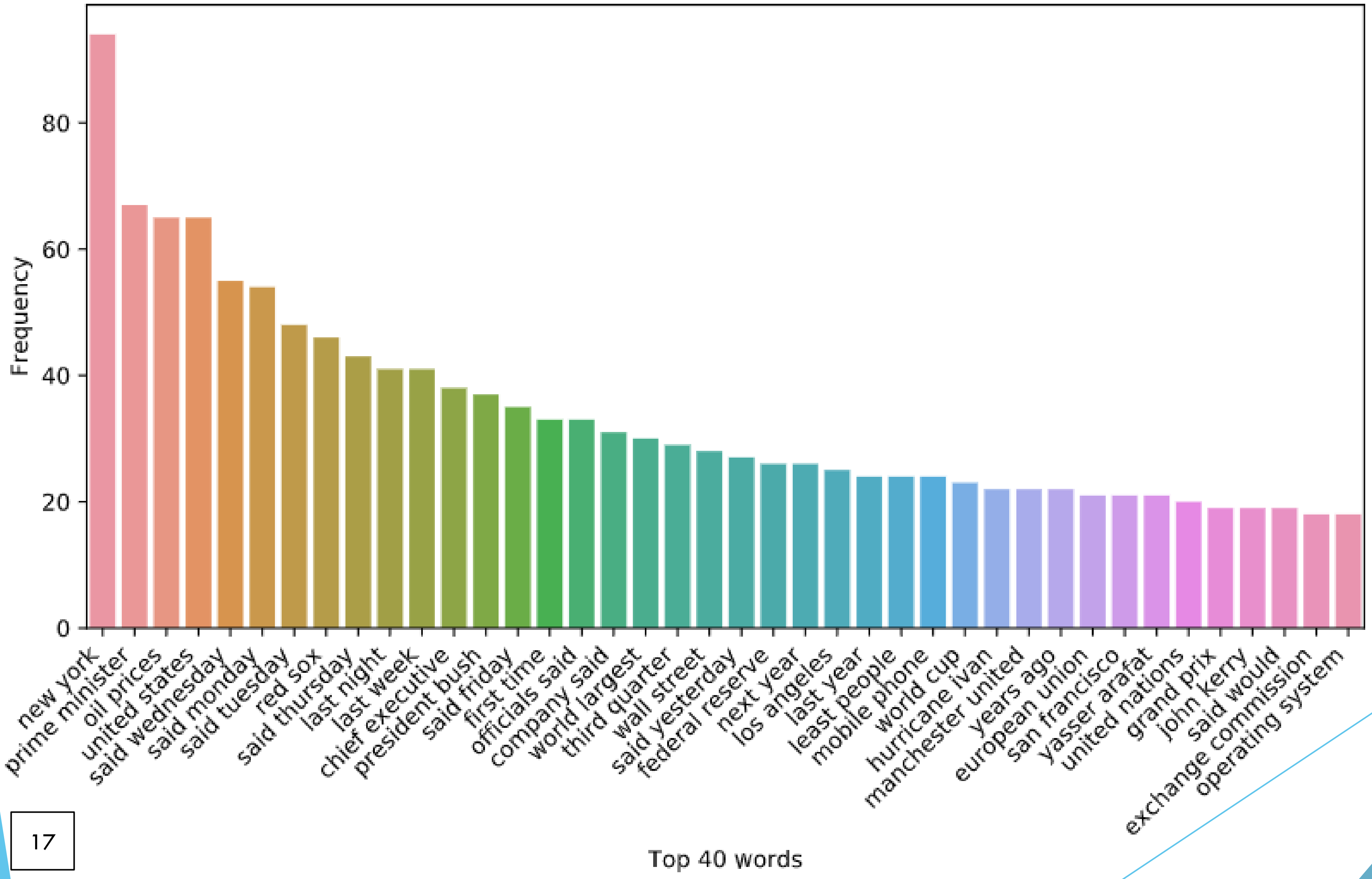


FEATURE ENGINEERING – BAG OF N-GRAMS

```
# Use countvectorizer to get a vector of n-grams

cv = CountVectorizer(min_df = 2, lowercase = True,
                     token_pattern=r'\b[A-Za-z]{2,}\b', ngram_range = (2, 3))
x_train_cv = cv.fit_transform(x_train).toarray()
x_test_cv = cv.transform(x_test).toarray()

# get all unique words in the corpus

ngram_vocab = cv.get_feature_names()

# produce a dataframe including the feature names

x_train_bagofngrams = pandas.DataFrame(x_train_cv, columns=ngram_vocab)
x_test_bagofngrams = pandas.DataFrame(x_test_cv, columns=ngram_vocab)
x_train_bagofngrams.head()
```


YORK UNIVERSITÉ UNIVERSITY

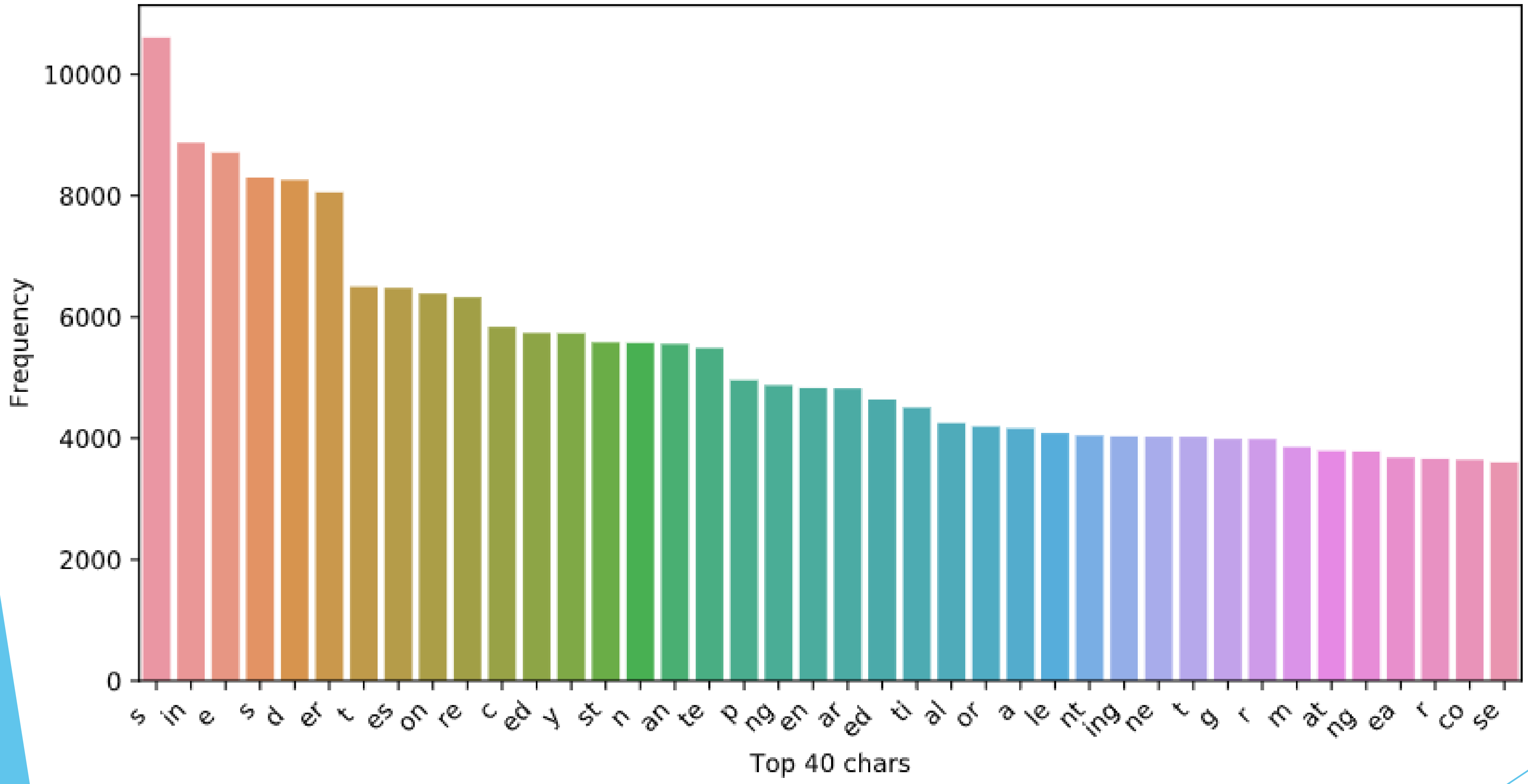
FEATURE ENGINEERING – BAG OF N-GRAMS (Cont'd)



FEATURE ENGINEERING – BAG OF CHARS

```
# Use countvectorizer to get a vector of chars

cv = CountVectorizer(analyzer='char', min_df = 2, ngram_range = (2, 3),
                     token_pattern=r'\b[A-Za-z]{2,}\b')
x_train_cv = cv.fit_transform(x_train).toarray()
x_test_cv = cv.transform(x_test).toarray()

# get all unique words in the corpus

cv_char_vocab = cv.get_feature_names()

# produce a dataframe including the feature names

x_train_cv_char = pandas.DataFrame(x_train_cv, columns = cv_char_vocab)
x_test_cv_char = pandas.DataFrame(x_test_cv, columns=cv_char_vocab)
x_train_cv_char.head()
```

FEATURE ENGINEERING – BAG OF CHARS (Cont'd)



FEATURE ENGINEERING – TF/IDF UNIGRAMS

```
# Use TF/IDF vectorizer to get a vector of unigrams

tfidf_vect = TfidfVectorizer(sublinear_tf = True, min_df = 2, ngram_range = (1, 1),
                             use_idf = True, token_pattern=r'\b[A-Za-z]{2,}\b')
x_train_tfidf_unigram = tfidf_vect.fit_transform(x_train).toarray()
x_test_tfidf_unigram = tfidf_vect.transform(x_test).toarray()

# get all unique words in the corpus

vocab = tfidf_vect.get_feature_names()

# produce a dataframe including the feature names

x_train_tfidf_unigram = pandas.DataFrame(numpy.round(x_train_tfidf_unigram, 2),
columns = vocab)
x_test_tfidf_unigram = pandas.DataFrame(numpy.round(x_test_tfidf_unigram, 2),
columns = vocab)
x_train_tfidf_unigram.head()
```

FEATURE ENGINEERING – TF/IDF N-GRAMS

```
# Use TF/IDF vectorizer to get a vector of n-grams

tfidf_vect = TfidfVectorizer(sublinear_tf = True, min_df = 2, ngram_range = (2, 3),
                              use_idf = True, token_pattern=r'\b[A-Za-z]{2,}\b')
x_train_tfidf_ngram = tfidf_vect.fit_transform(x_train).toarray()
x_test_tfidf_ngram = tfidf_vect.fit_transform(x_test).toarray()

# get all unique words in the corpus

vocab = tfidf_vect.get_feature_names()

# produce a dataframe including the feature names

x_train_tfidf_ngram = pandas.DataFrame(numpy.round(x_train_tfidf_ngram, 2),
columns = vocab)
x_test_tfidf_ngram = pandas.DataFrame(numpy.round(x_test_tfidf_ngram, 2), columns
= vocab)
x_train_tfidf_ngram.head()
```

FEATURE ENGINEERING – TF/IDF CHARS

```
# Use TF/IDF vectorizer to get a vector of chars

tfidf_vect = TfidfVectorizer(analyzer = 'char', sublinear_tf = True, min_df = 2,
                             ngram_range = (2, 3), use_idf = True,
                             token_pattern=r'\b[A-Za-z]{2,}\b')
x_train_tfidf_char = tfidf_vect.fit_transform(x_train).toarray()
x_test_tfidf_char = tfidf_vect.transform(x_test).toarray()

# get all unique words in the corpus

char_vocab = tfidf_vect.get_feature_names()

# produce a dataframe including the feature names

x_train_tfidf_char = pandas.DataFrame(numpy.round(x_train_tfidf_char, 2), columns = char_vocab)
x_test_tfidf_char = pandas.DataFrame(numpy.round(x_test_tfidf_char, 2), columns = char_vocab)
x_train_tfidf_char.head()
```


YORK
UNIVERSITÉ
UNIVERSITY

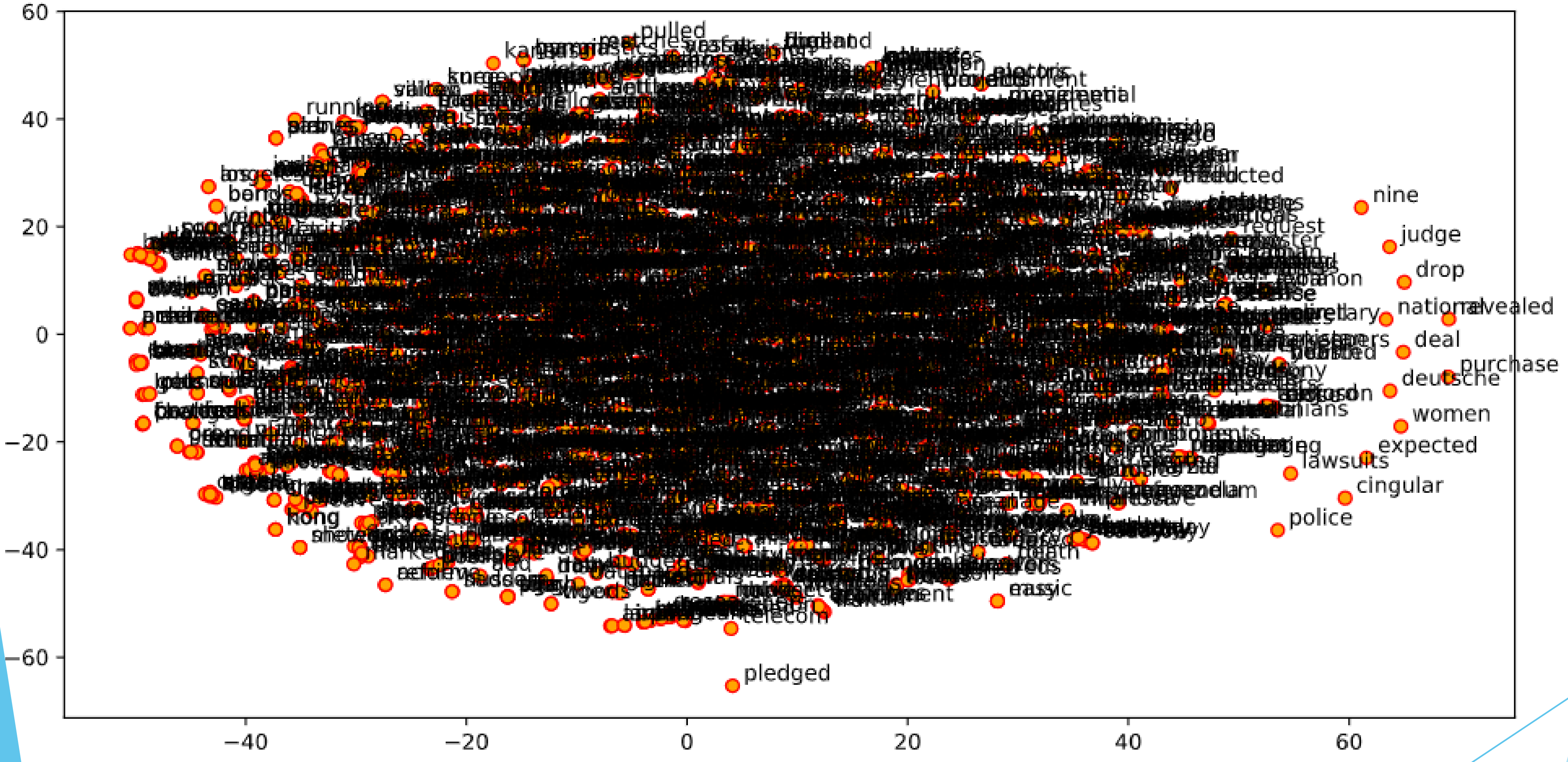
FEATURE ENGINEERING – WORD2VEC

```
# ## Using gensim to build Word2Vec

from gensim.models import word2vec

# tokenize sentences in corpus
wpt = nltk.WordPunctTokenizer()
tokenized_corpus = [wpt.tokenize(document) for document in x_train]

# Set values for various parameters
feature_size = 100    # Word vector dimensionality
window_context = 20          # Context window size
workers = 10
min_word_count = 5   # Minimum word count
sample = 1e-3   # Downsample setting for frequent words

w2v_model = word2vec.Word2Vec(tokenized_corpus, size=feature_size,
                          window=window_context, min_count=min_word_count,
                          sample=sample, iter=500)
```

FEATURE ENGINEERING – WORD2VEC (Cont'd)

```
# ### Visualize Word Embedding

# %%
from sklearn.manifold import TSNE
words = w2v_model.wv.index2word
wvs = w2v_model.wv[words]
tsne = TSNE(n_components=2, random_state=0, n_iter=500, perplexity=2)
numpy.set_printoptions(suppress=True)
T = tsne.fit_transform(wvs)
labels = words
plt.figure(figsize=(12, 6))
plt.scatter(T[:, 0], T[:, 1], c='orange', edgecolors='r')
for label, x, y in zip(labels, T[:, 0], T[:, 1]):
 plt.annotate(label, xy=(x+1, y+1), xytext=(0, 0), textcoords='offset points')
```

FEATURE ENGINEERING – WORD2VEC (Cont'd)



FEATURE ENGINEERING – WORD2VEC (Cont'd)

```
# ### Functions to get document level embeddings

def average_word_vectors(words, model, vocabulary, num_features):
 feature_vector = numpy.zeros((num_features,),dtype="float64")
 nwords = 0.

 for word in words:
    if word in vocabulary:
      nwords = nwords + 1.
      feature_vector = numpy.add(feature_vector, model[word])

 if nwords:
    feature_vector = numpy.divide(feature_vector, nwords)
 return feature_vector


def averaged_word_vectorizer(corpus, model, num_features):
 vocabulary = set(model.wv.index2word)
 features = [average_word_vectors(tokenized_sentence, model, vocabulary, num_features)
            for tokenized_sentence in corpus]
 return numpy.array(features)
```

FEATURE ENGINEERING – WORD2VEC (Cont'd)

```
# ### Obtain document level embeddings

w2v_feature_array = averaged_word_vectorizer(corpus=tokenized_corpus, model=w2v_model,
                                             num_features=feature_size)
pandas.DataFrame(w2v_feature_array)

word_freq_df = pandas.DataFrame(x_train_bagofwords.toarray(), columns=cv.get_feature_names())
top_words_df = pandas.DataFrame(word_freq_df.sum()).sort_values(0, ascending=False)
word_freq_df.head(20)

top_words_df.head(20)
```

MODELLING - SVM

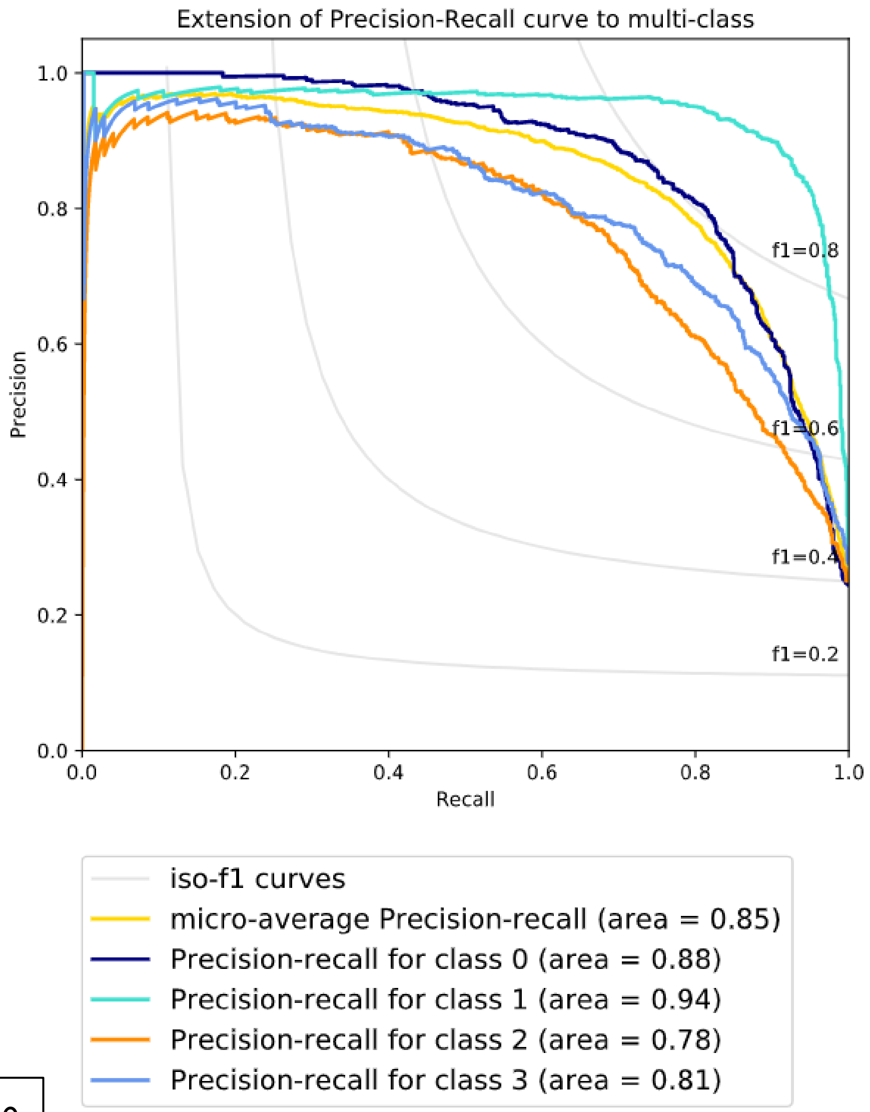
```
# Run classifier

classifier = OneVsRestClassifier(svm.LinearSVC(random_state=1))
classifier.fit(x_train_bagofwords, y_train)
y_score = classifier.decision_function(x_test_bagofwords)

# The average precision score in multi-label settings
# For each class
precision = dict()
recall = dict()
average_precision = dict()
for i in range(n_classes):
    precision[i], recall[i], _ = precision_recall_curve(y_test[:, i], y_score[:, i])
    average_precision[i] = average_precision_score(y_test[:, i], y_score[:, i])

# A "micro-average": quantifying score on all classes jointly
precision["micro"], recall["micro"], _ = precision_recall_curve(y_test.ravel(), y_score.ravel())
average_precision["micro"] = average_precision_score(y_test, y_score, average="micro")
print('Average precision score, micro-averaged over all classes: {0:0.2f}'
      .format(average_precision["micro"]))
```

MODELLING – SVM (Cont'd)

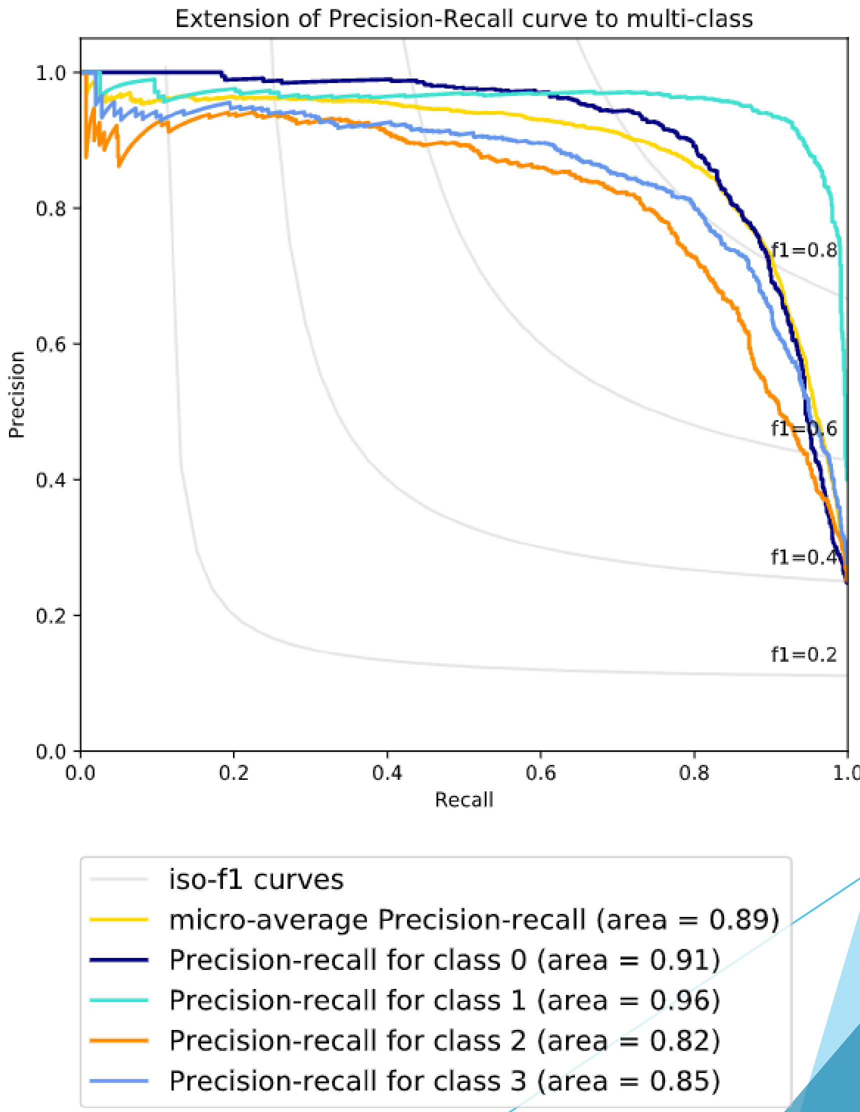
```
# Plot the micro-averaged Precision-Recall curve

plt.figure()
plt.step(recall['micro'], precision['micro'], where='post')

plt.xlabel('Recall')
plt.ylabel('Precision')
plt.ylim([0.0, 1.05])
plt.xlim([0.0, 1.0])
plt.title(
    'Average precision score, micro-averaged over all classes: AP={0:0.2f}'
    .format(average_precision["micro"]))
```

SVM MODEL RESULTS

Precision Recall F1, CV Unigrams

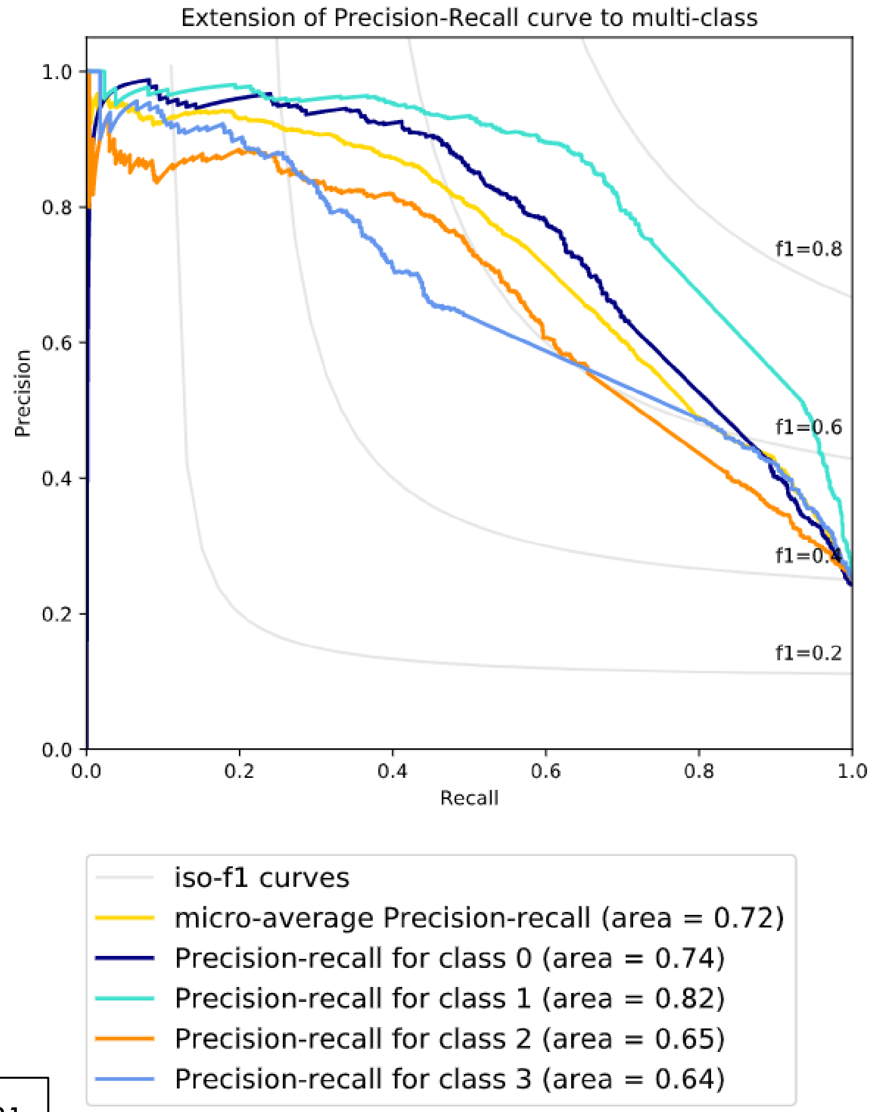


Precision Recall F1, TF/IDF Unigrams

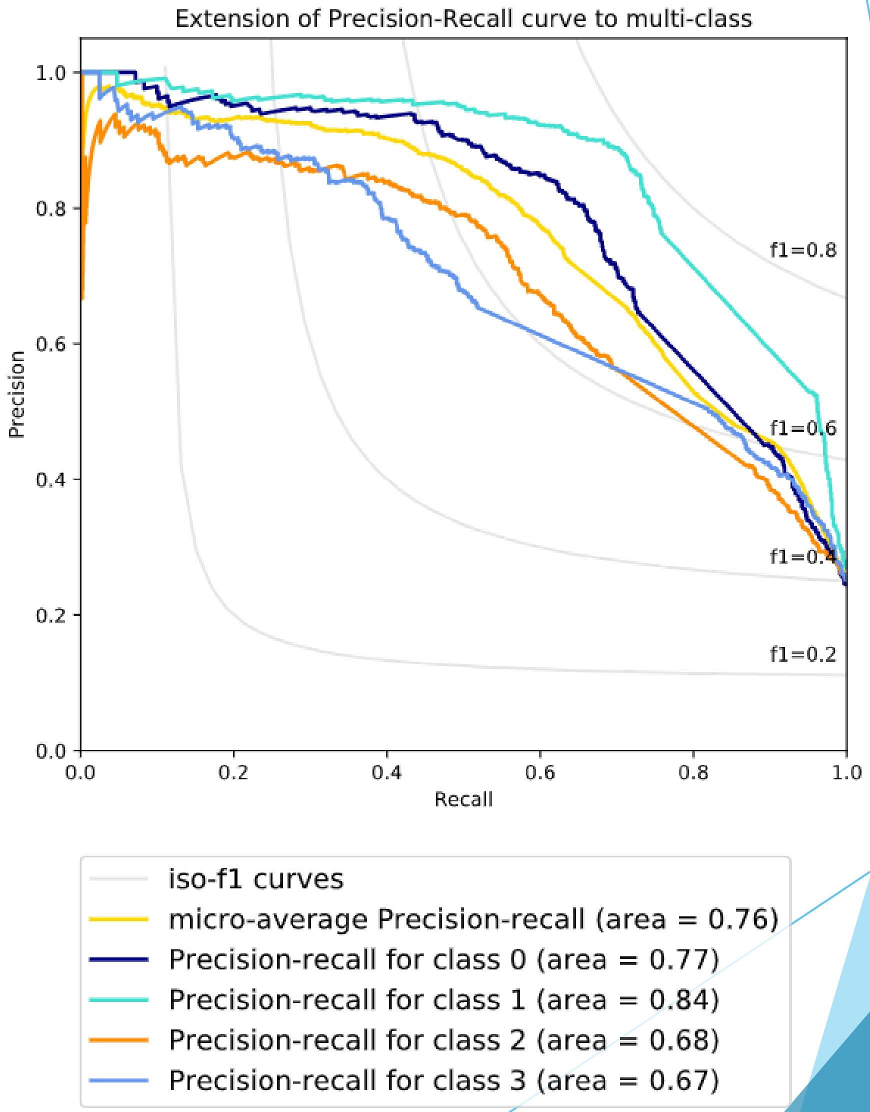


SVM MODEL RESULTS

<u>Precision Recall F1, CV N-grams</u>



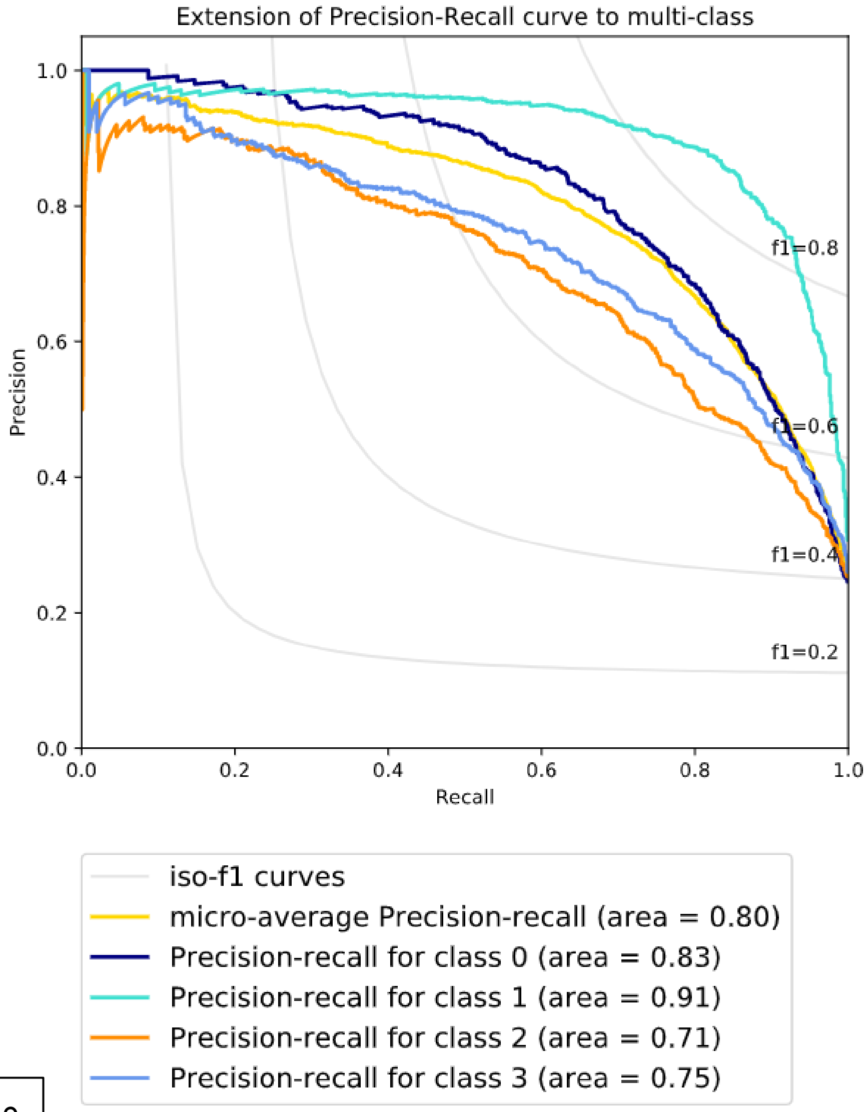
<u>Precision Recall F1, TF/IDF N-grams</u>



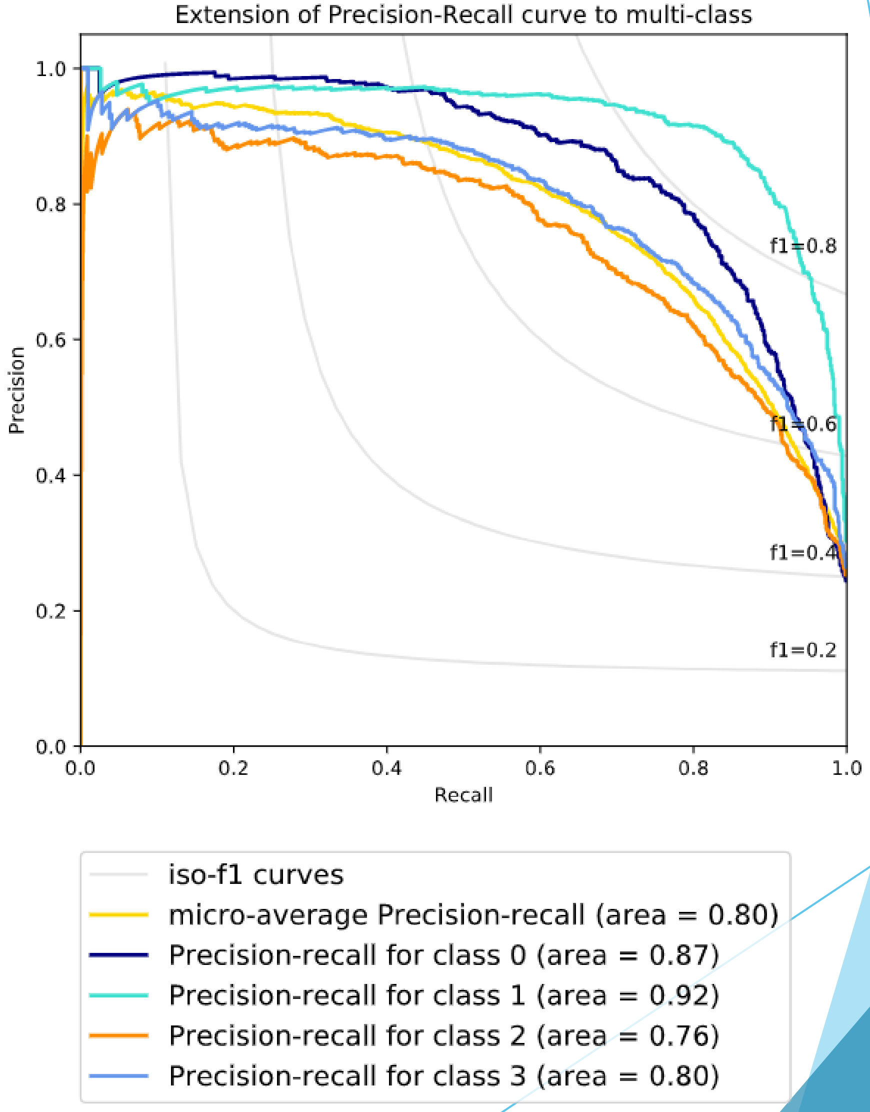
YORK UNIVERSITÉ UNIVERSITY

SVM MODEL RESULTS

Precision Recall F1, CV Chars

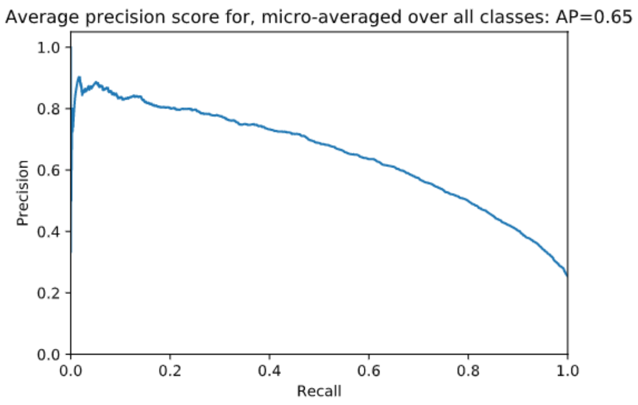


Precision Recall F1, TF/IDF Chars

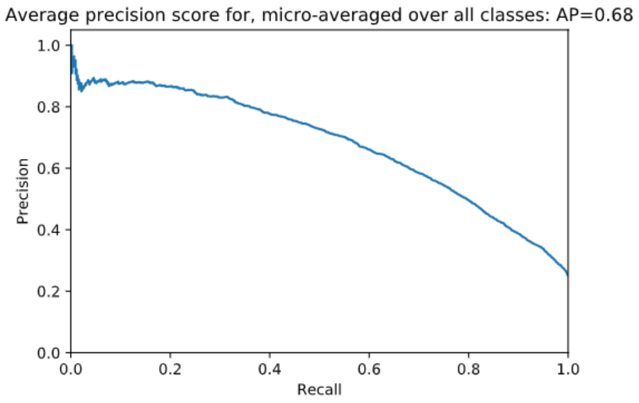


YORK UNIVERSITÉ UNIVERSITY

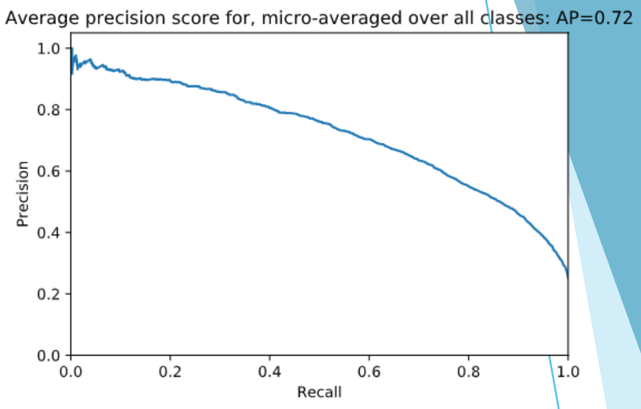
SVM MODEL RESULTS – WORD2VEC FEATURE SIZE COMPARISON



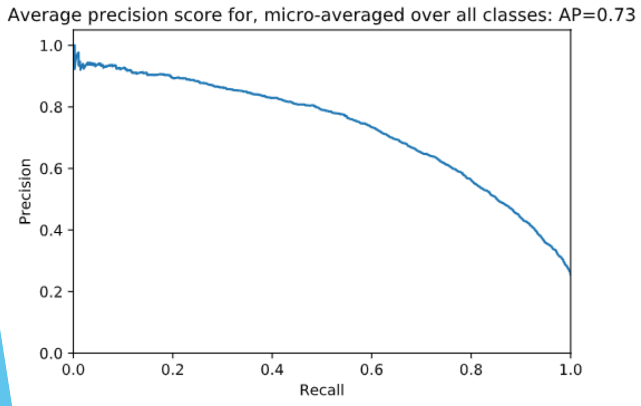
Feature size = 100



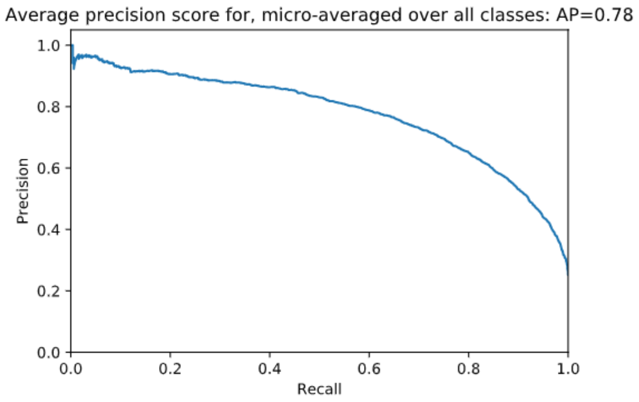
Feature size = 500



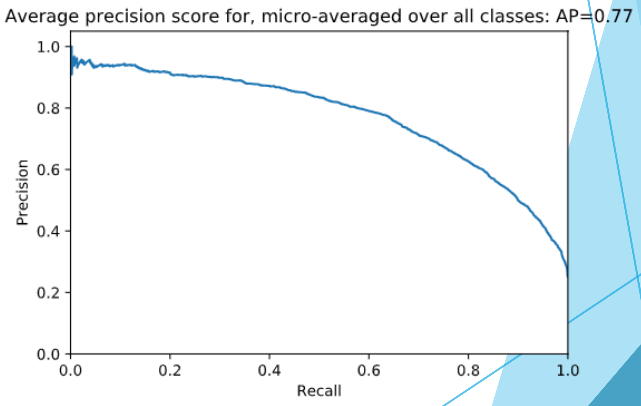
Feature size = 1000



Feature size = 2000

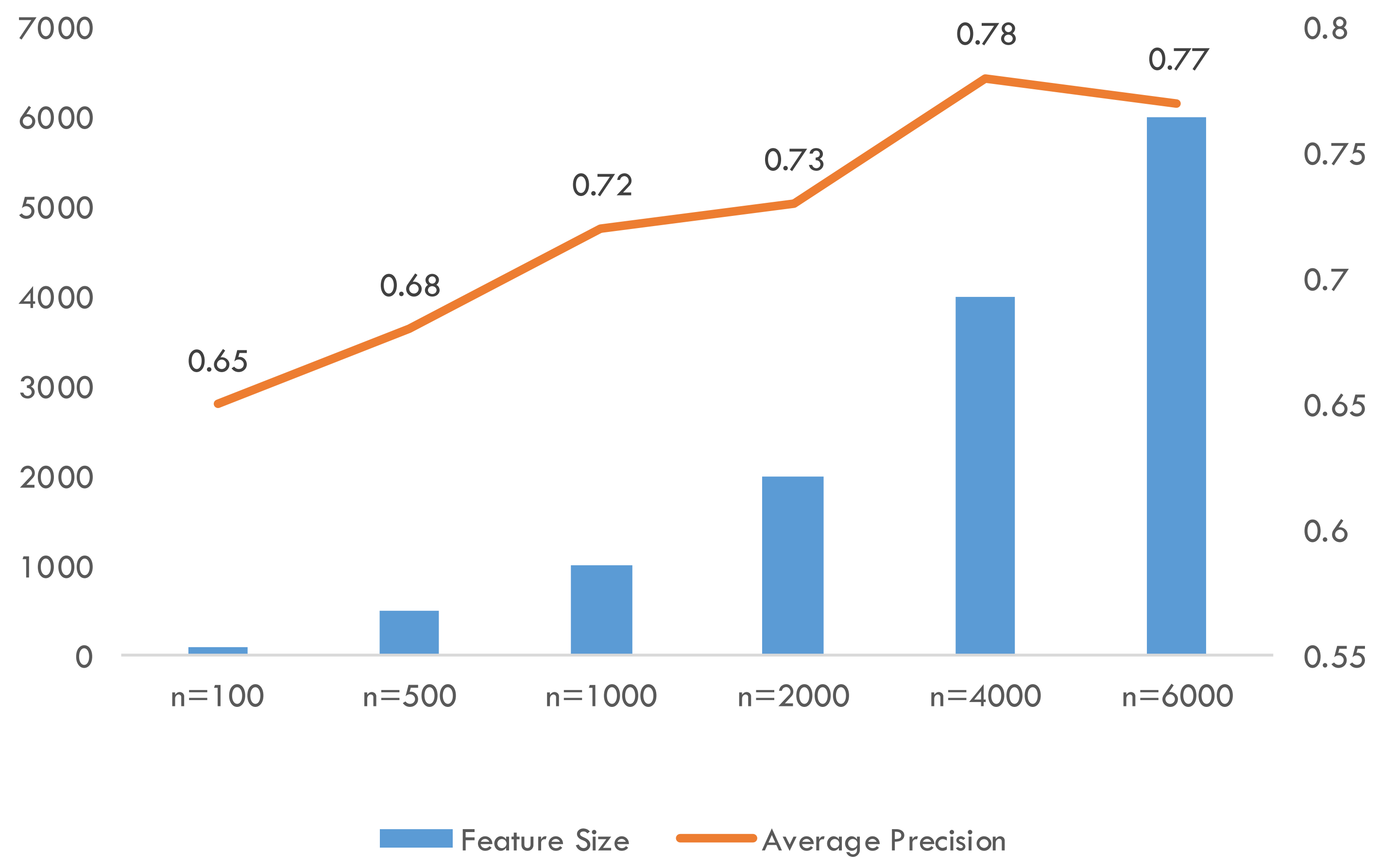


Feature size = 4000



Feature size = 6000

SVM MODEL RESULTS – WORD2VEC FEATURE SIZE COMPARISON



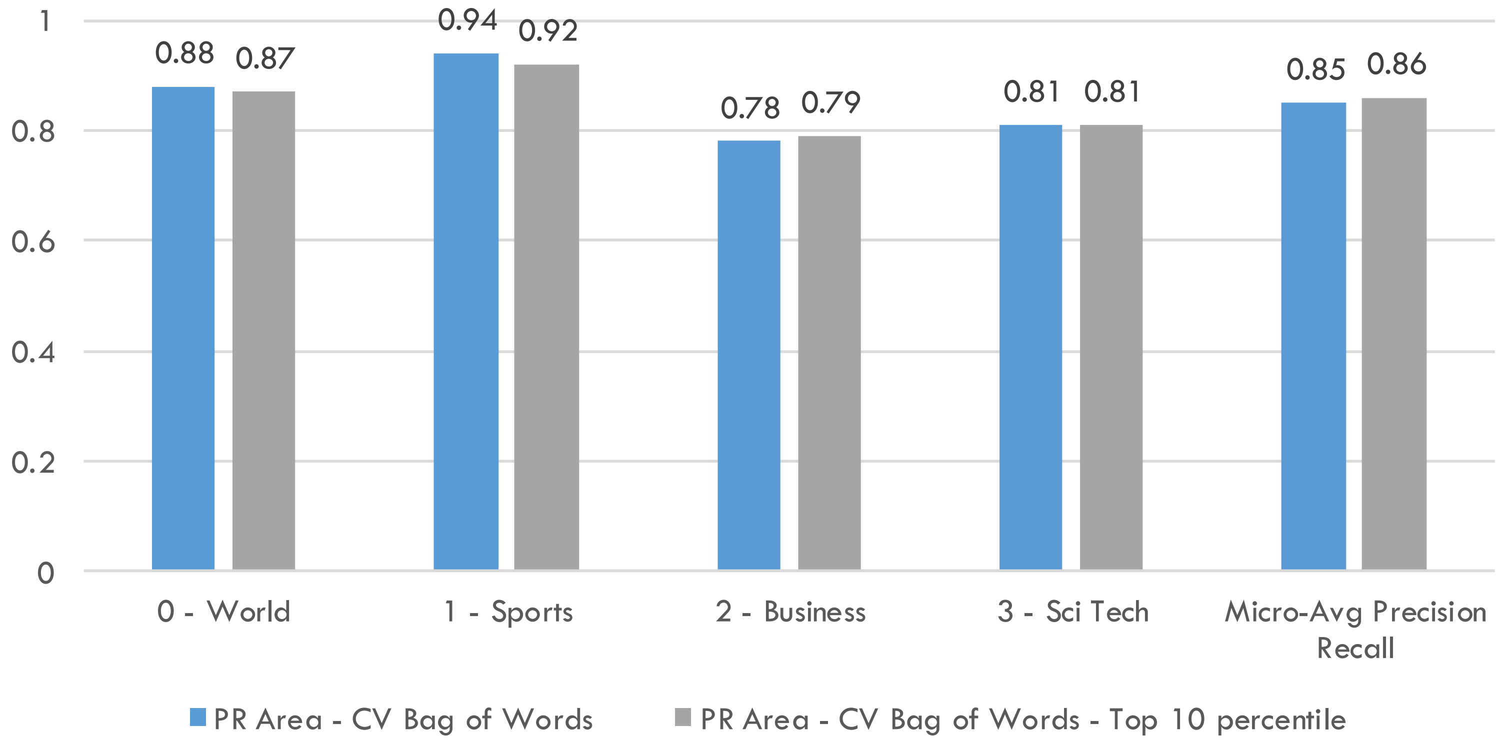
FEATURE SELECTION

- We did feature selection using the package f_class_if from sklearn to select the top 10 percentile from each vectorizer / embedding

```
selector = SelectPercentile(f_classif, percentile=10)
selector.fit(x_train_cv, train_data_sample.category)
x_train_cv_10p = selector.transform(x_train_cv).toarray()
x_test_cv_10p = selector.transform(x_test_cv).toarray()
```

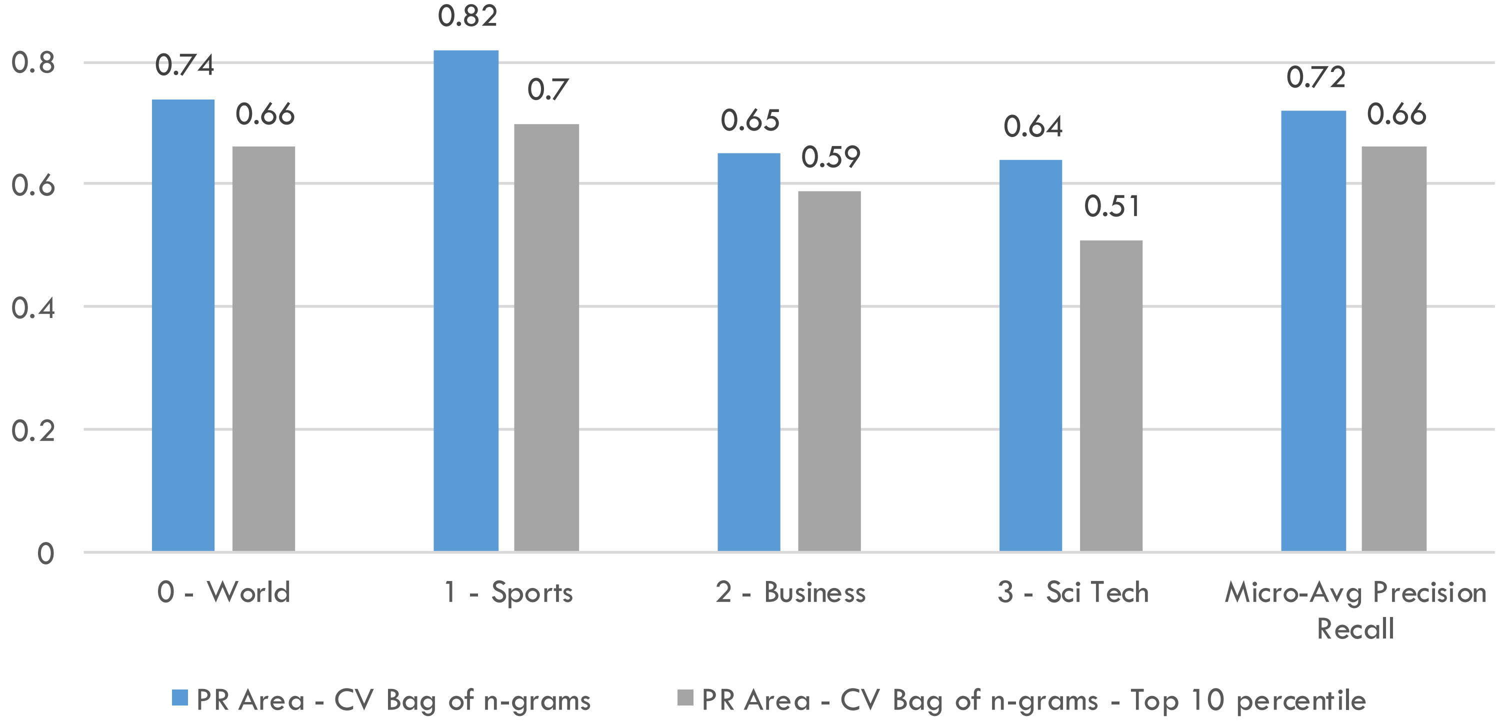
FEATURE SELECTION RESULTS COMPARISON

- We ran SVM with test & train of both actual and feature selected datasets. Below is a comparison of results between CountVectorizer Bag of Words for actual data vs feature selected data



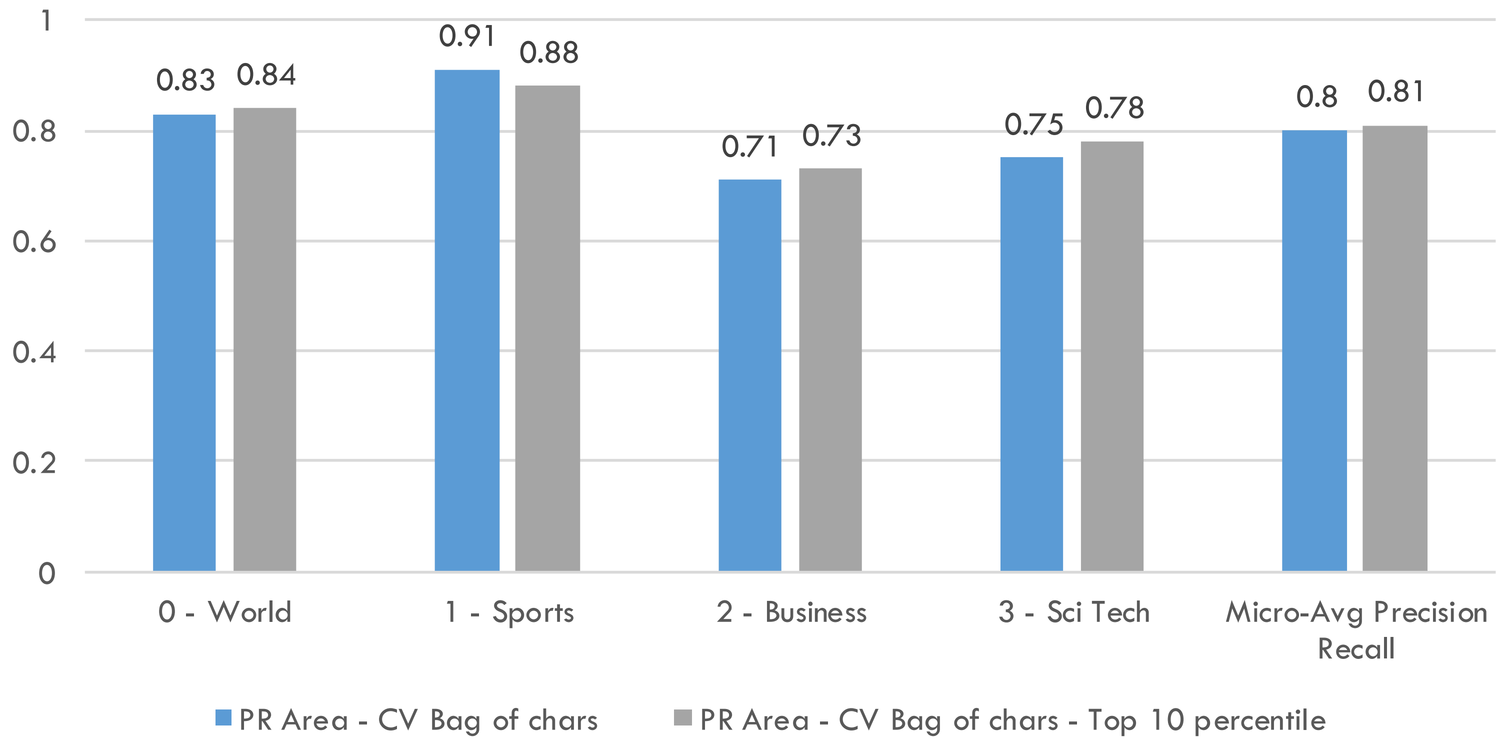
FEATURE SELECTION RESULTS COMPARISON

- Below is a comparison of results between CountVectorizer Bag of N-grams for actual data vs feature selected data



FEATURE SELECTION RESULTS COMPARISON

- Below is a comparison of results between CountVectorizer Bag of chars for actual data vs feature selected data



- We will continue feature selection with other feature selection methods to better determine the significance of it to our project

- We will continue to build other classification models like Logistic Regression, Naïve Bayes & Decision Trees

PROJECT MILESTONE 2 – 5th May 2020

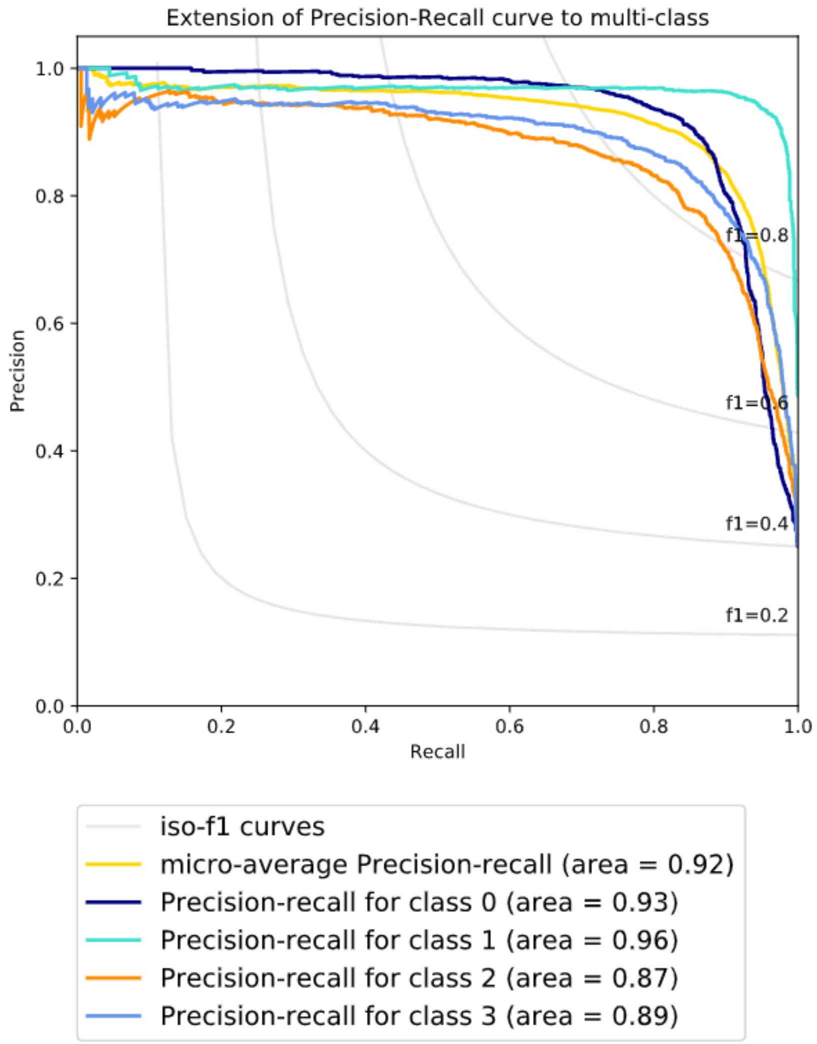
YORK
UNIVERSITÉ
UNIVERSITY

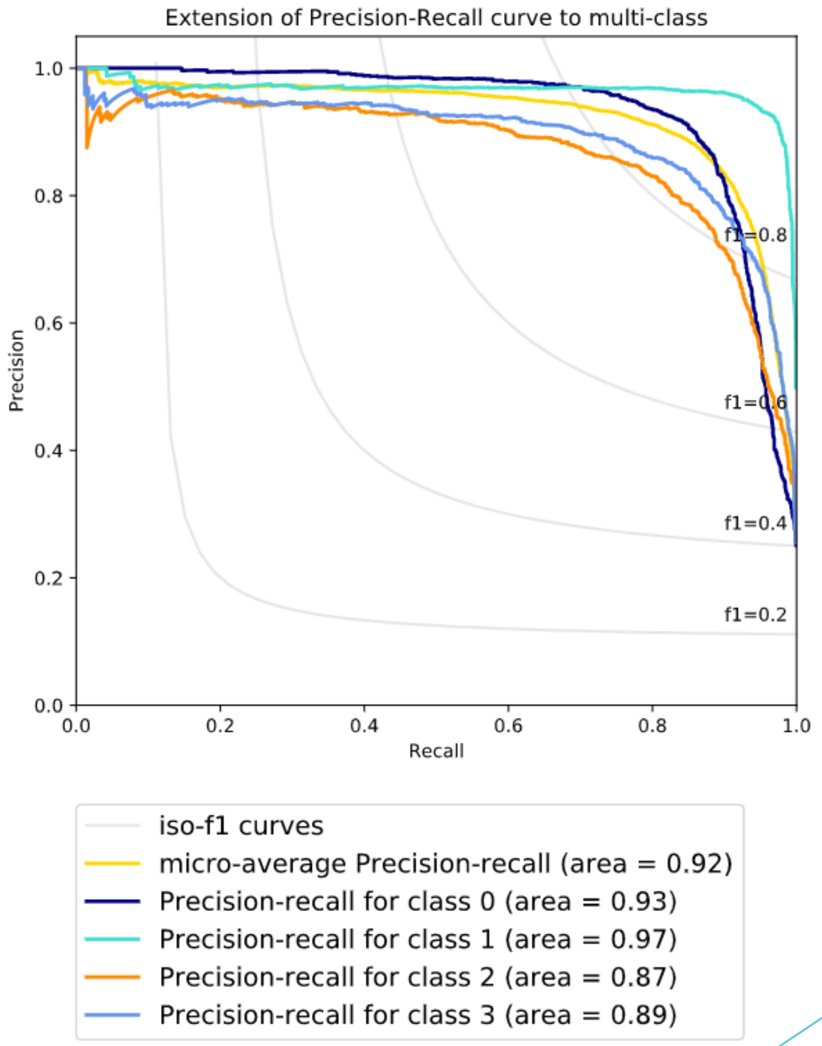
MODELS

- With the previous SVM model as the baseline, the following models were built,
  - Logistic Regression
  - Naïve Bayes
  - Decision Trees

  All the models were built using Word2Vec embedding. The embedding was trained using the 120,000 instances, the entire training set

MODELS – PERFORMANCE COMPARISON

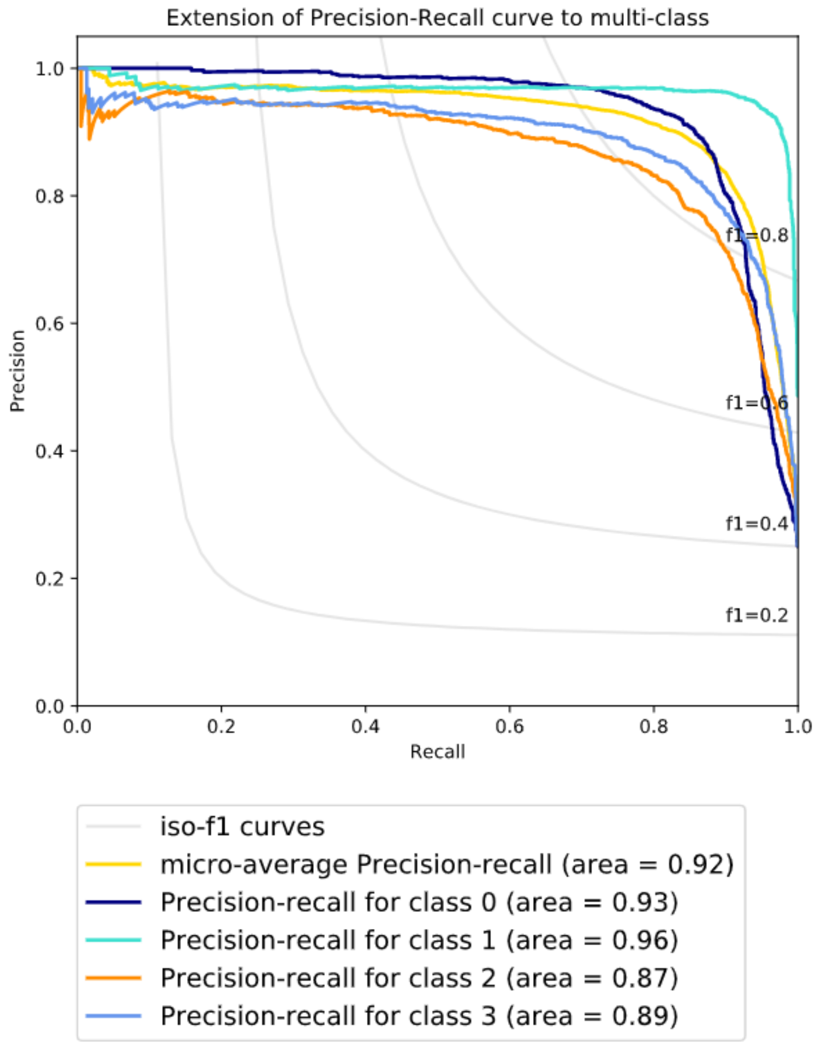


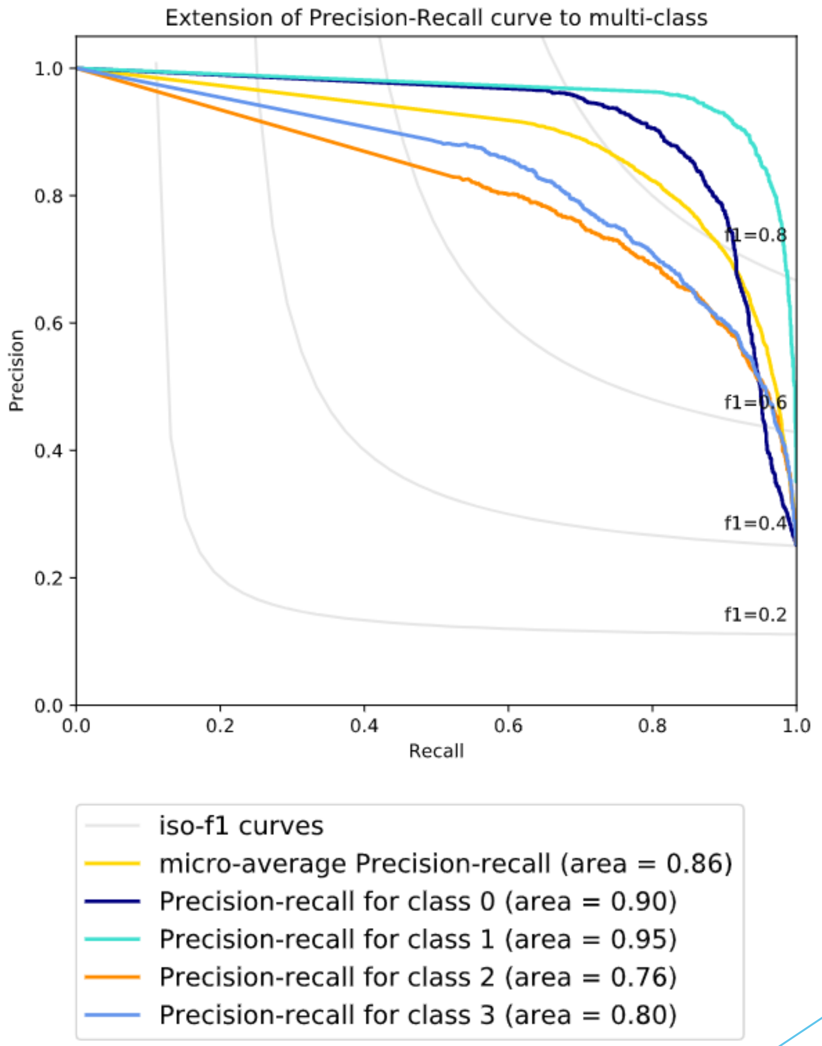


<u>Baseline Model – SVM</u>
- 120,000 instances
- Min word count = 5
- No.of Dimensions = 300

<u>Logistic Regression</u>
- 120,000 instances
- Min word count = 5
- No.of Dimensions = 300

MODELS – PERFORMANCE COMPARISON (Cont'd)

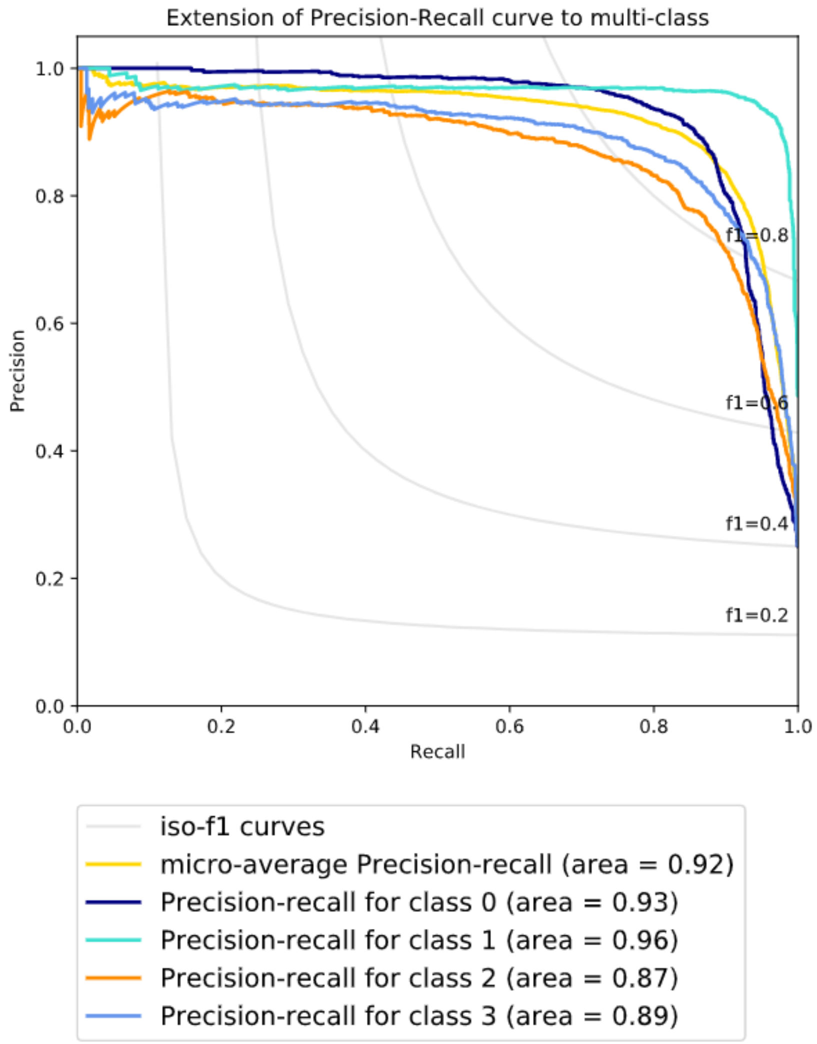


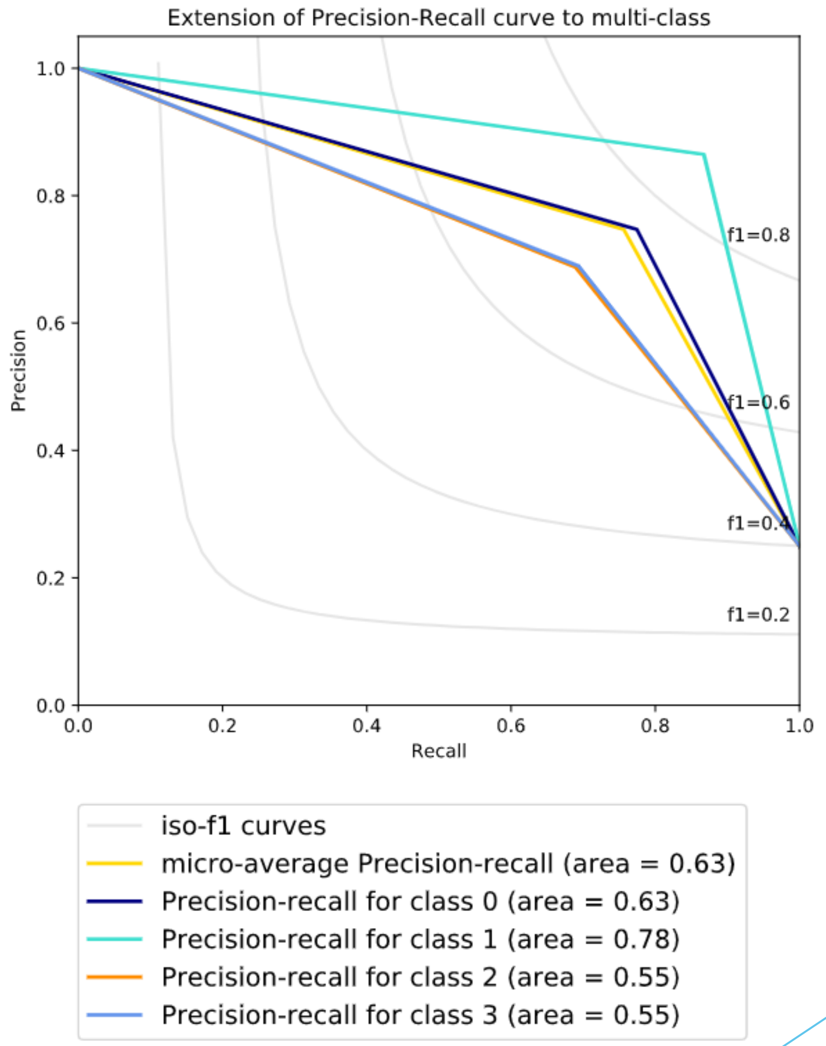


<u>Baseline Model – SVM</u>
- 120,000 instances
- Min word count = 5
- No.of Dimensions = 300

<u>Naïve Bayes</u>
- 120,000 instances
- Min word count = 5
- No.of Dimensions = 300

MODELS – PERFORMANCE COMPARISON (Cont'd)





<u>Baseline Model – SVM</u>

- 120,000 instances
- Min word count = 5
- No.of Dimensions = 300

<u>Decision Trees</u>

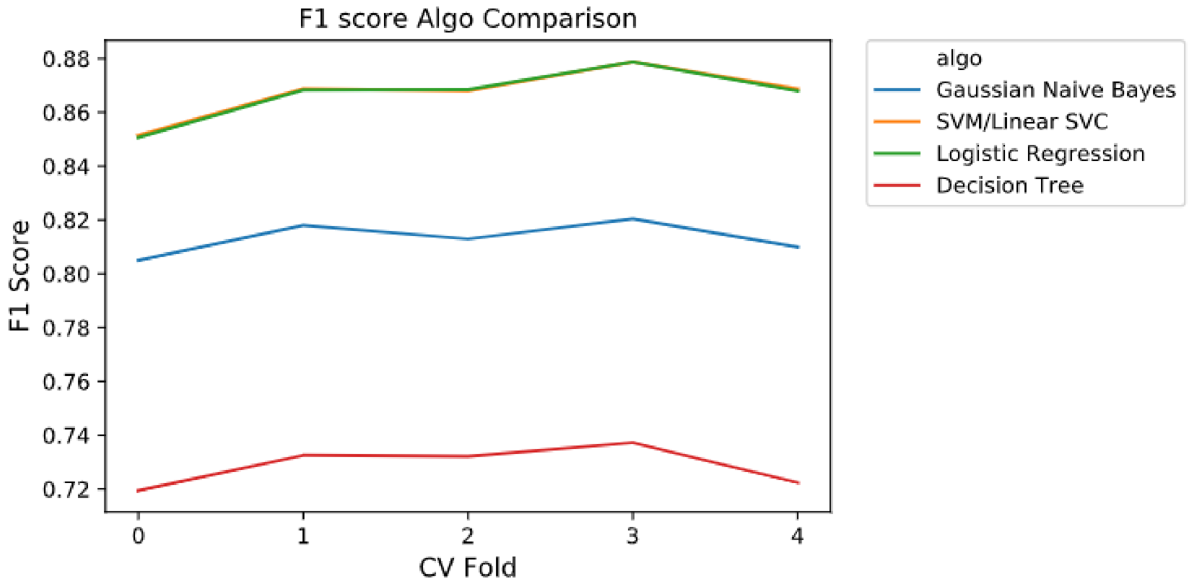
- 120,000 instances
- Min word count = 5
- No.of Dimensions = 300

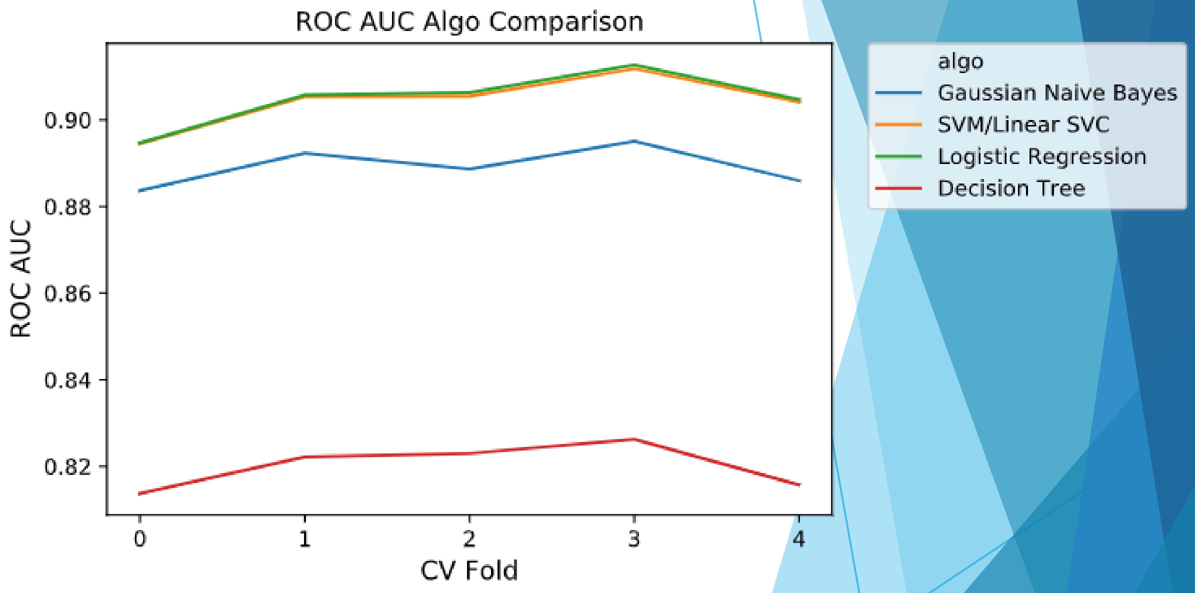
CROSS VALIDATION

- We ran a 5 fold cross validation on the training dataset and re-ran all the models
- The models on the "solo" runs were trained on the 120,000 instance training set and tested on a separate 7600 instance test set
- The results of cross validation closely follow the previous "solo" runs
- Logistic Regression & SVM compare very closely in terms of the metrics. However, SVM was very resource intensive

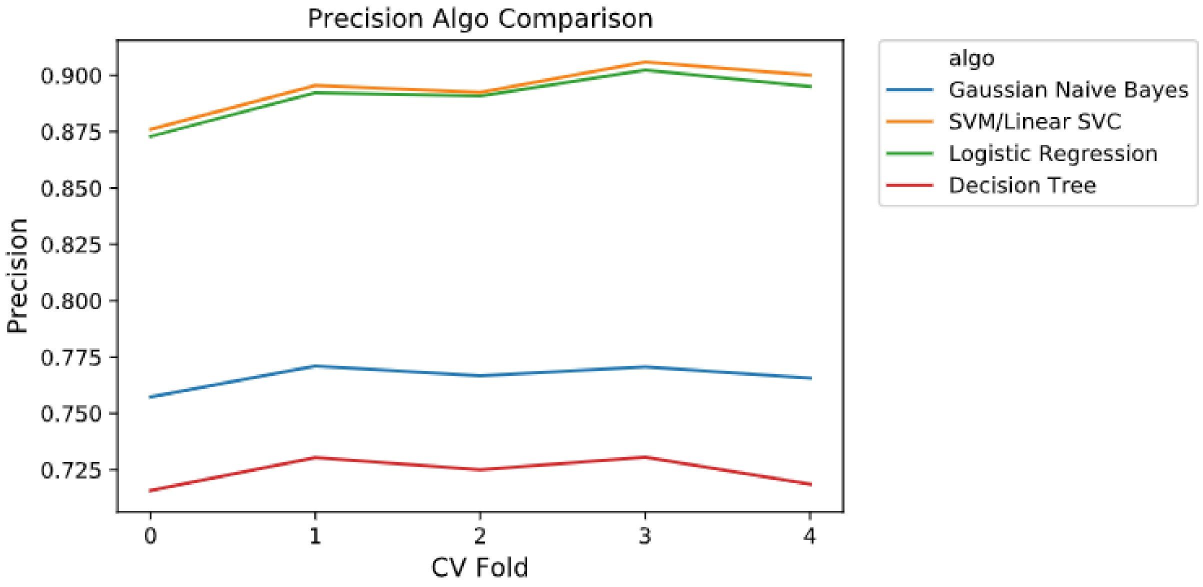
YORK UNIVERSITÉ UNIVERSITY

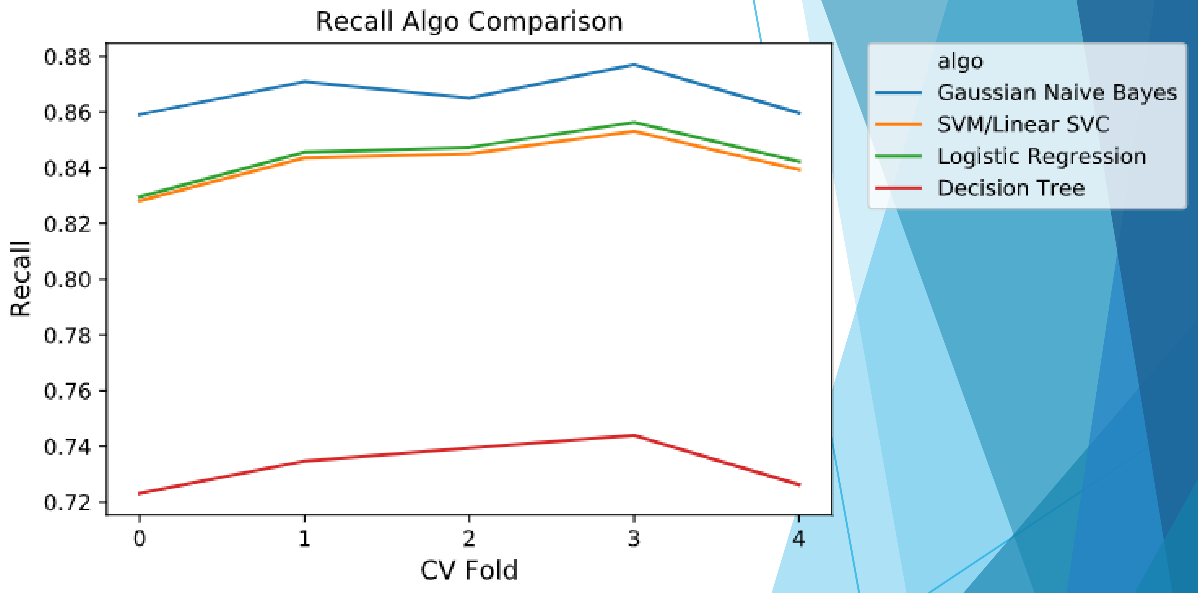
CROSS VALIDATION – COMPARISON OF RESULTS



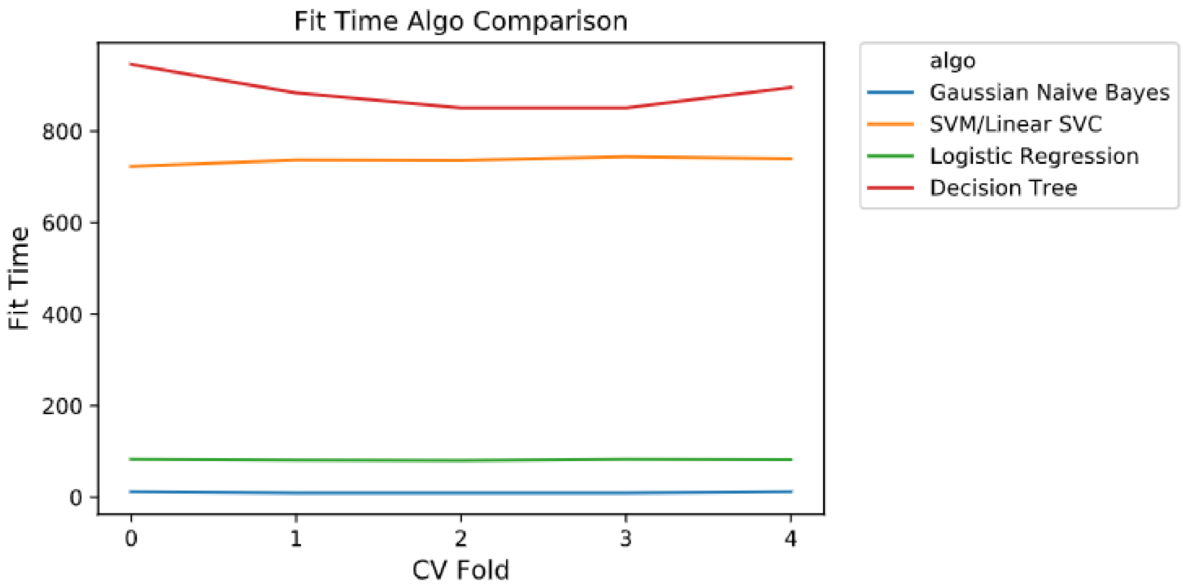


CROSS VALIDATION – COMPARISON OF RESULTS





CROSS VALIDATION – COMPARISON OF RESULTS

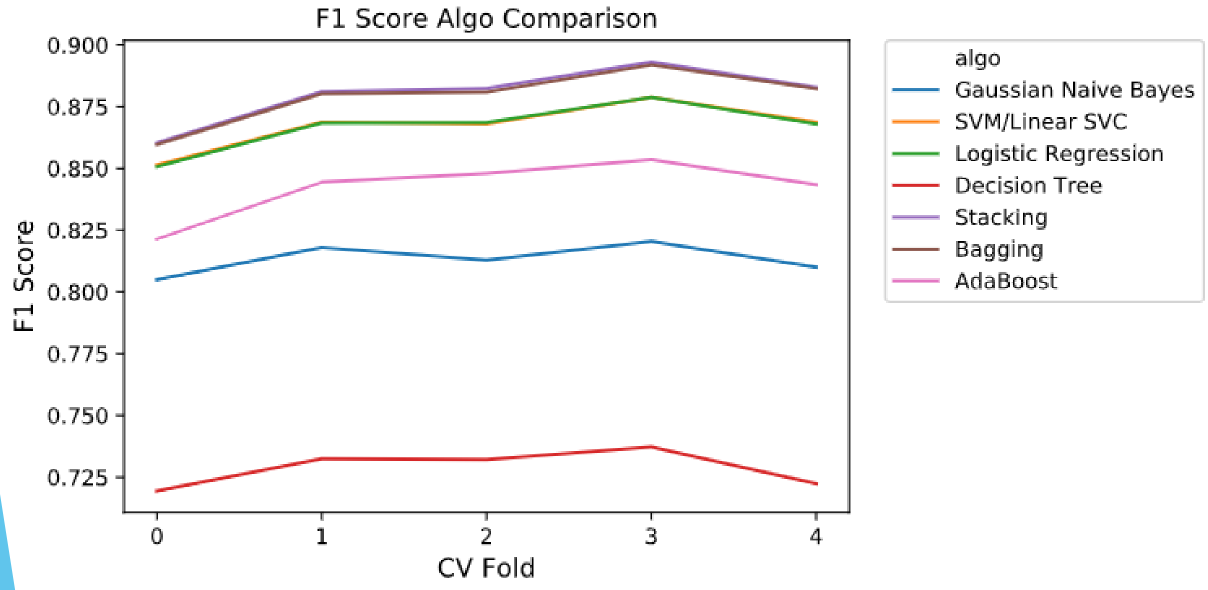


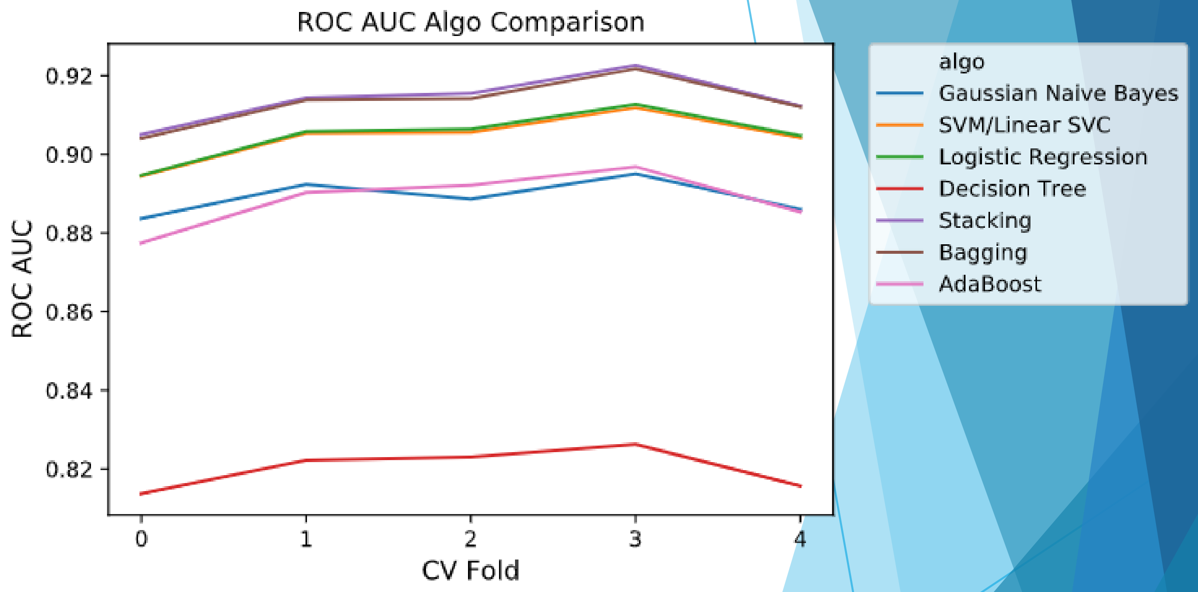
- Based on the previous metrics and comparison, Logistic Regression emerges as the model of choice

ENSEMBLE METHODS

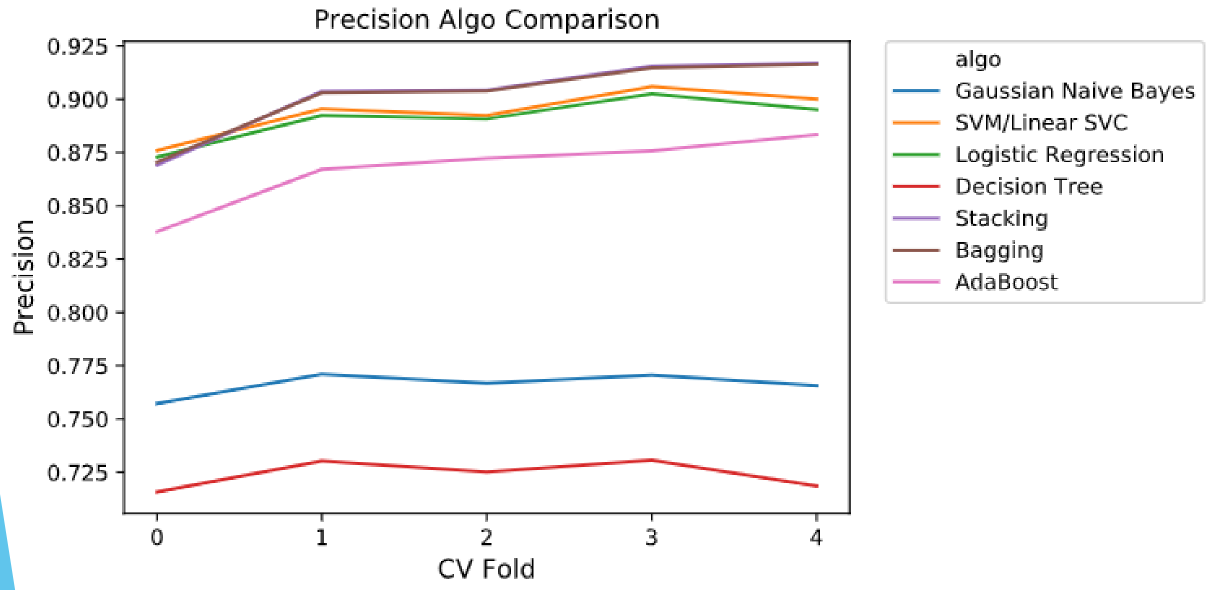
- The following ensemble methods were used
  - Bagging
    - The base estimator was chosen to be Logistic Regression
  - Stacking
    - The initial estimators in stacking were chosen to be Naïve bayes and SVM. The meta learner was Logistic Regression
  - Boosting
    - An Adaboost classifier was used for boosting in this iteration.

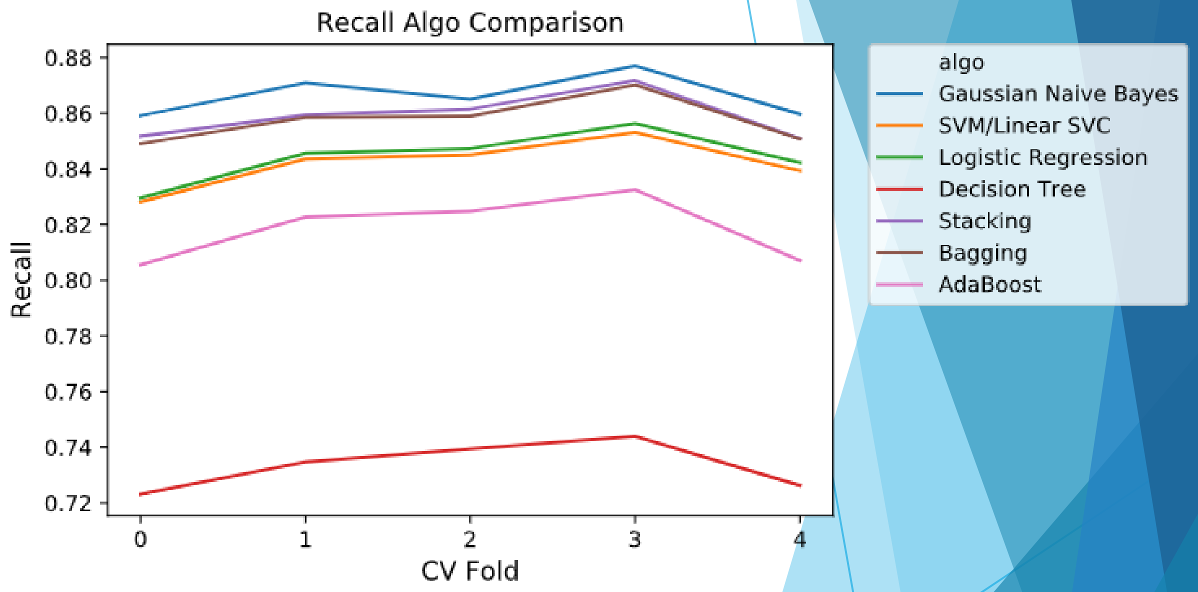
ENSEMBLE METHODS vs OTHERS



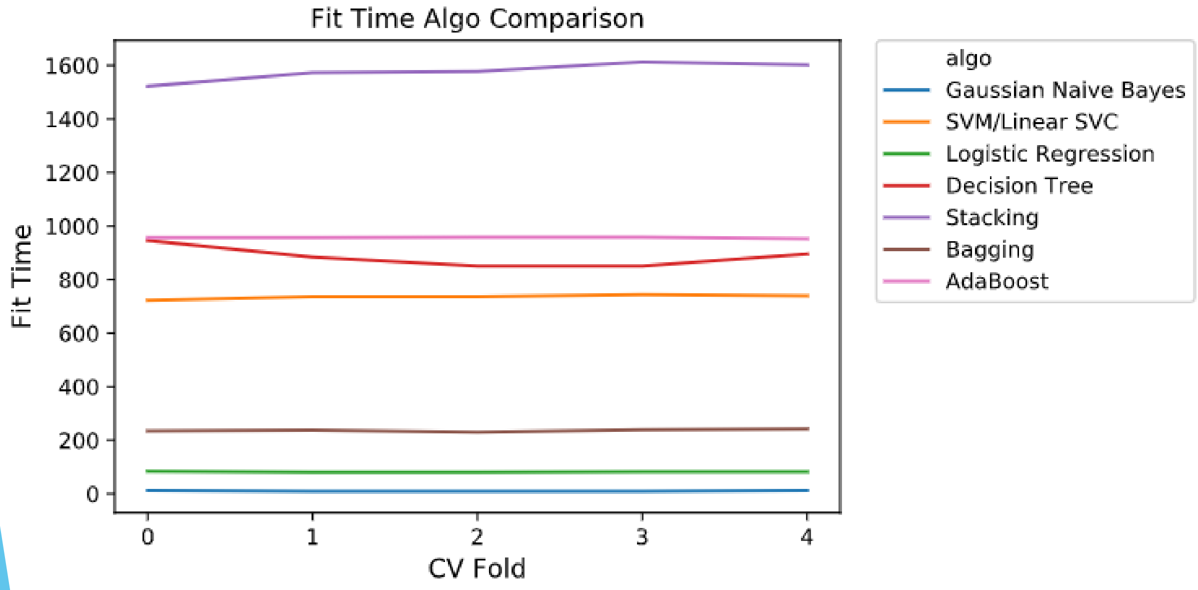


ENSEMBLE METHODS vs OTHERS





ENSEMBLE METHODS vs OTHERS



- Based on the metrics seen so far, it is clear that “Bagging” ensemble method is the algorithm of choice
- Stacking and Bagging perform almost similarly in terms of F1 Score, ROC AUC, Precision & Recall
- However, in terms of Fit time, stacking took almost 25 minutes to fit, while bagging took just over 3.5 mins