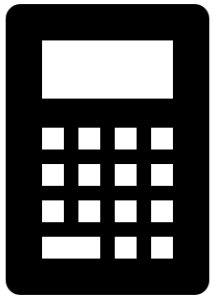
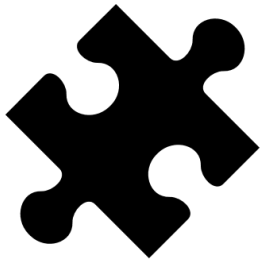
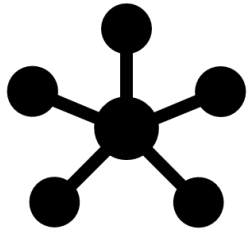
Predicting the Scientific Domain of a Research Paper in arXiv

Final Project – IS 590 ML Fall 2018

Team - arXiver Text Miners

- Nikolaus Parulian
- Janina Sarol
- Aarushi Mishra
- Saumil Shah

Contents

Introduction

What is arXiv?

- A huge electronic archive and distribution server for scientific research papers(only)
- Maintained by Cornell University
- Launched 27 years ago, in August, 1991
- Acts as a platform where people can read and share research articles before publishing them
- A team of moderators reviews the submissions
- Any viewer has open access to 1, 471, 151 e-prints, as of today

What is arXiv?

Covers e-prints from the following scientific domains :

- Physics
- Mathematics
- Computer Science
- Nonlinear sciences
- Quantitative biology
- Quantitative Finance
- Statistics
- Electrical Engineering and systems science
- Economics

Why are we here after all?

- For making a successful submission to arXiv, it is required that the domain(s) of the article is specified
- We are here to present the machine learning model, which we built, that can predict the domain(s) of the article
- This model can also be used for
  - ✓ Building a domain suggestion tool for new research papers
  - ✓ Designing a similar suggestion tool for other publications
  - ✓ Analysing the evolution of scientific domains (e.g. through understanding the changes in model performance when using features from different time slices)
- At the end of this presentation, we will provide a demo of our working model

Dataset Analysis

- Training Dataset – consists of complete data from year 2016 and a part from year 2017, accounting to 200,092 rows, in total
- Test Dataset – consists of data from 2017, accounting to 37, 125 rows, in total
- Test dataset is independent of/ different from training dataset

Prediction Classes

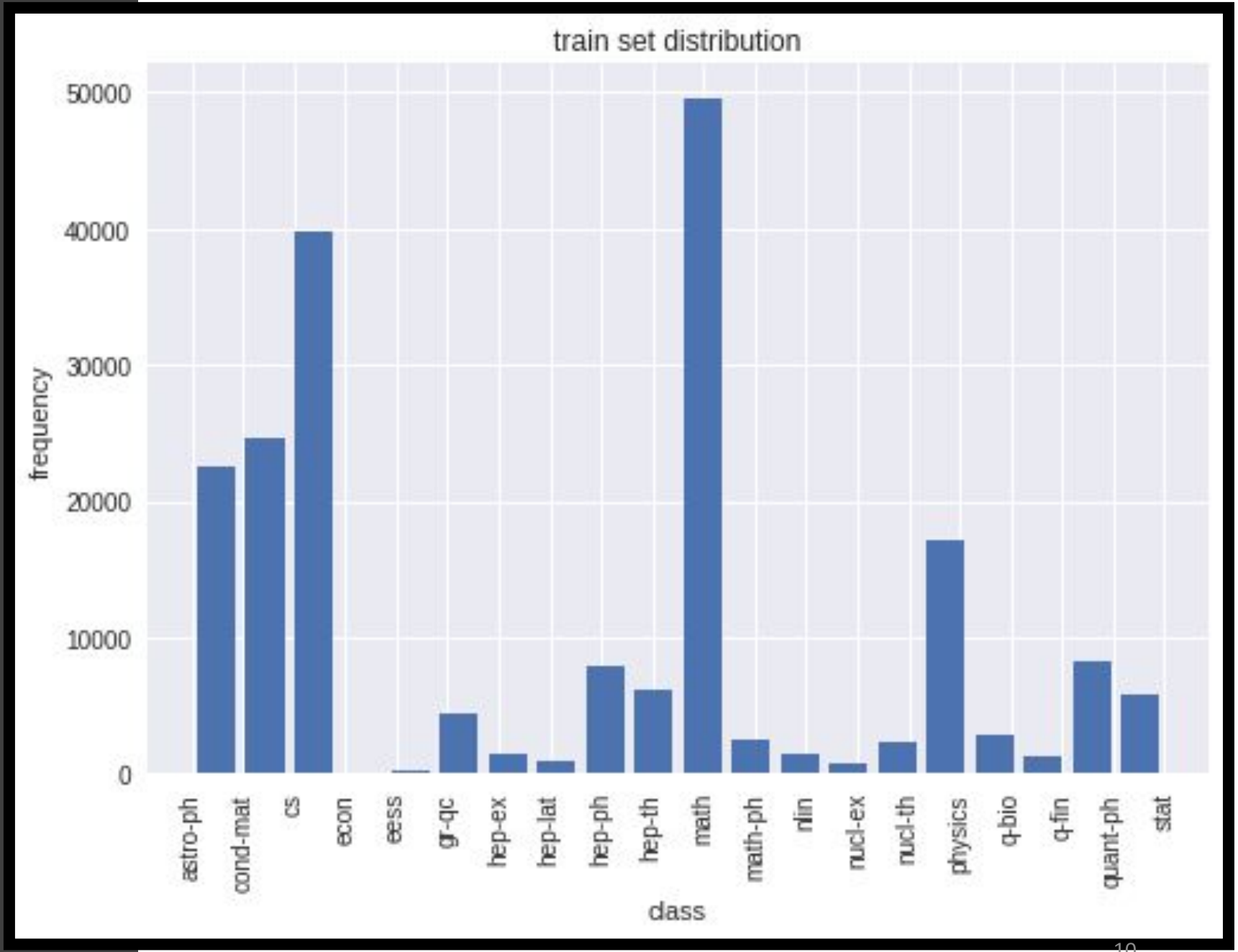
There are 20 different prediction classes

Class_Num	Class	Description
0	math	Mathematics
1	physics	Physics
2	astro-ph	Astrophysics
3	gr-qc	General Relativity and Quantum Cosmology
4	hep-th	High Energy Physics - Theory
5	nucl-th	Nuclear Theory
6	stat	Statistics
7	cond-mat	Condensed Matter
8	cs	Computer Science
9	hep-ph	High Energy Physics - Phenomenology
10	nucl-ex	Nuclear Experiment
11	q-fin	Quantitative Finance
12	quant-ph	Quantum Physics
13	math-ph	Mathematical Physics
14	nlin	Nonlinear Sciences
15	q-bio	Quantitative Biology
16	hep-ex	High Energy Physics - Experiment
17	hep-lat	High Energy Physics - Lattice
18	econ	Economics
19	eess	Electrical Engineering and System Science

Training Dataset

Top 3 prediction classes:

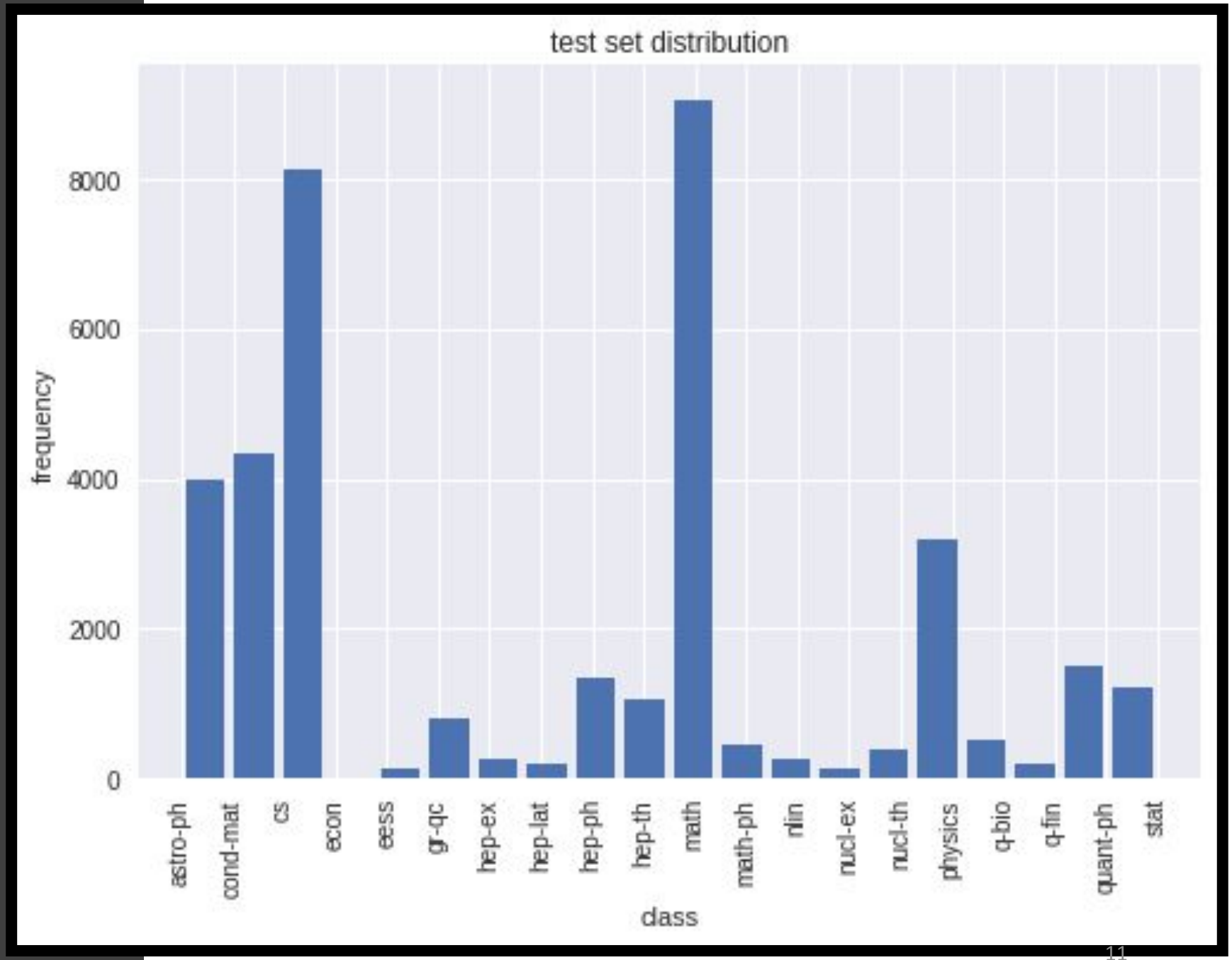
1. Mathematics
2. Computer Science
3. Condensed Matter



Test Dataset

Top 3 prediction classes:

1. Mathematics
2. Computer Science
3. Condensed Matter



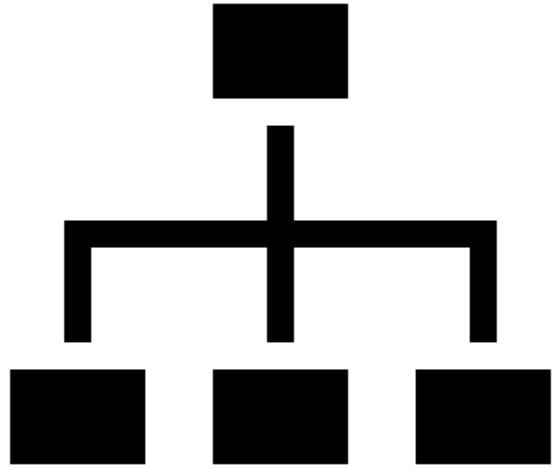
Data Preprocessing

1. Extracted noun words from abstract using NLTK library. Additionally also cleaned up stop words and non-alphabetical characters.

2. Generated a TF-IDF vector using scikit learn TFIDFVectorizer to get 44,819 features for the model.

3. Reduced the TF-IDF features from 44,819 to top 10,000 features.

Models and Results



Logistic Regression

1. Built a multinomial logistic regression model using scikit learn package.
2. Trained it using top 10,000 features.
3. Accuracy: 80.28 %
4. Kappa score: 0.766
5. Precision: 0.79
6. Recall: 0.80
7. F1 score: 0.79

Logistic Regression (Cross Validation)

1. Built a multinomial logistic regression model using 10-fold cross validation.
2. Trained it using top 10,000 features.
3. Accuracy: 80.33 %
4. Kappa score: 0.767
5. Precision: 0.79
6. Recall: 0.80
7. F1 score: 0.79
8. Cross validation does not bring about a significant improvement in the results obtained.

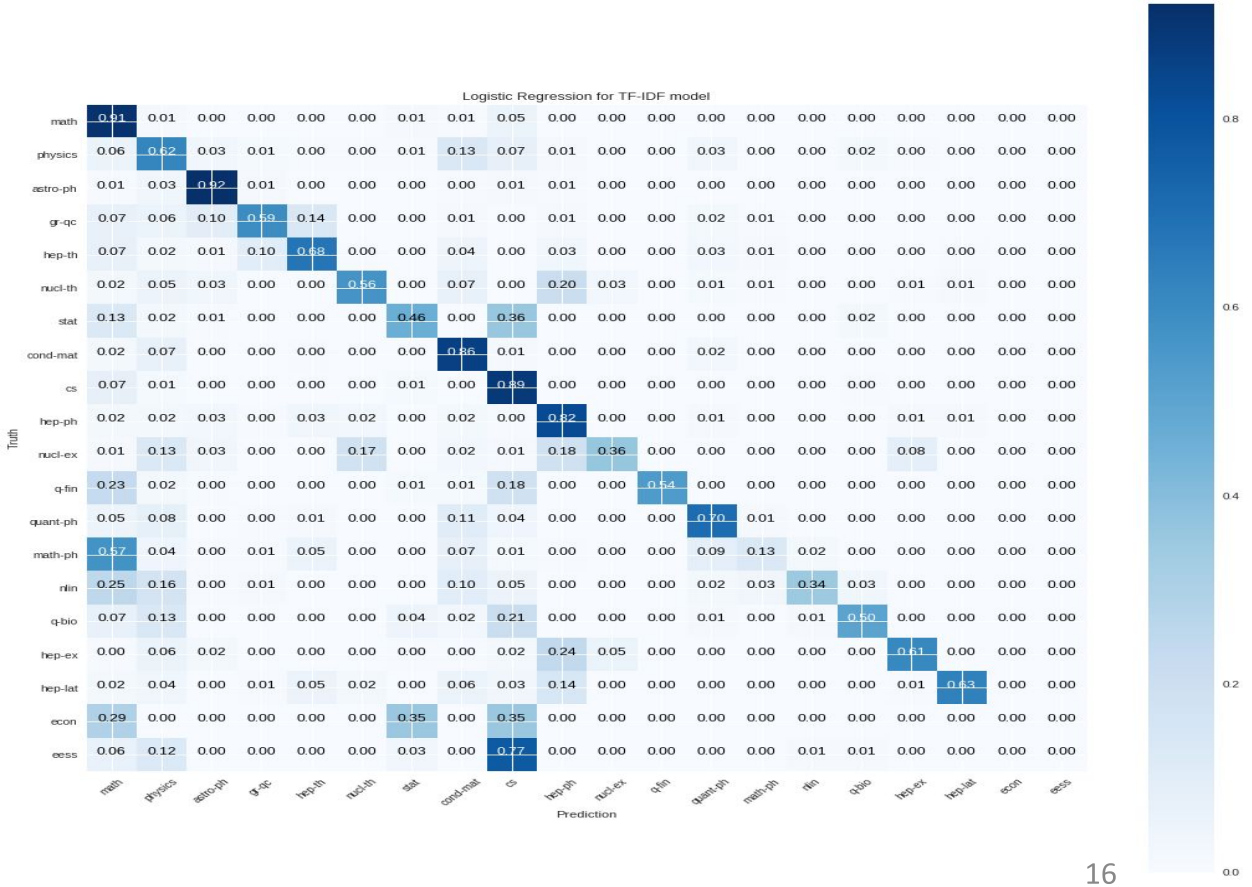
Logistic Regression (Cross Validation)

```
y_test_pred = sorted(lr_cv,key=lambda x:x["kappa"],reverse=True)[0]["model"].predict(X_t
```

```
from sklearn.metrics import classification_report,accuracy_score,cohen_kappa_score
acc = accuracy_score(y_test_class,y_test_pred)
kap = cohen_kappa_score(y_test_class,y_test_pred)
print("accuracy:",acc)
print("kappa:",kap)
print(classification_report(y_test_class,y_test_pred,target_names=y_class_label))
```

```
accuracy: 0.8033939393939394
kappa: 0.7673778218632191
```

	precision	recall	f1-score	support
math	0.83	0.91	0.87	9054
physics	0.65	0.62	0.63	3187
astro-ph	0.93	0.92	0.93	3980
gr-qc	0.69	0.59	0.64	798
hep-th	0.74	0.68	0.70	1065
nucl-th	0.69	0.56	0.62	379
stat	0.68	0.46	0.55	1204
cond-mat	0.81	0.86	0.84	4324
cs	0.83	0.89	0.86	8125
hep-ph	0.79	0.82	0.81	1354
nucl-ex	0.59	0.36	0.45	132
q-fin	0.78	0.54	0.64	214
quant-ph	0.76	0.70	0.73	1498
math-ph	0.45	0.13	0.20	442
nlin	0.58	0.34	0.43	261
q-bio	0.63	0.50	0.56	504
hep-ex	0.81	0.61	0.69	263
hep-lat	0.85	0.63	0.72	195
econ	0.00	0.00	0.00	17
eess	0.00	0.00	0.00	129
avg / total	0.79	0.80	0.79	37125



Multinomial Naïve Bayes

1. Built a multinomial naive bayes model using scikit learn package.
2. Trained it using top 10,000 features.
3. Accuracy: 77.01 %
4. Kappa score: 0.725
5. Precision: 0.76
6. Recall: 0.77
7. F1 score: 0.75
8. Does not perform well as compared to the logistic regression model.

Multinomial Naïve Bayes (Cross Validation)

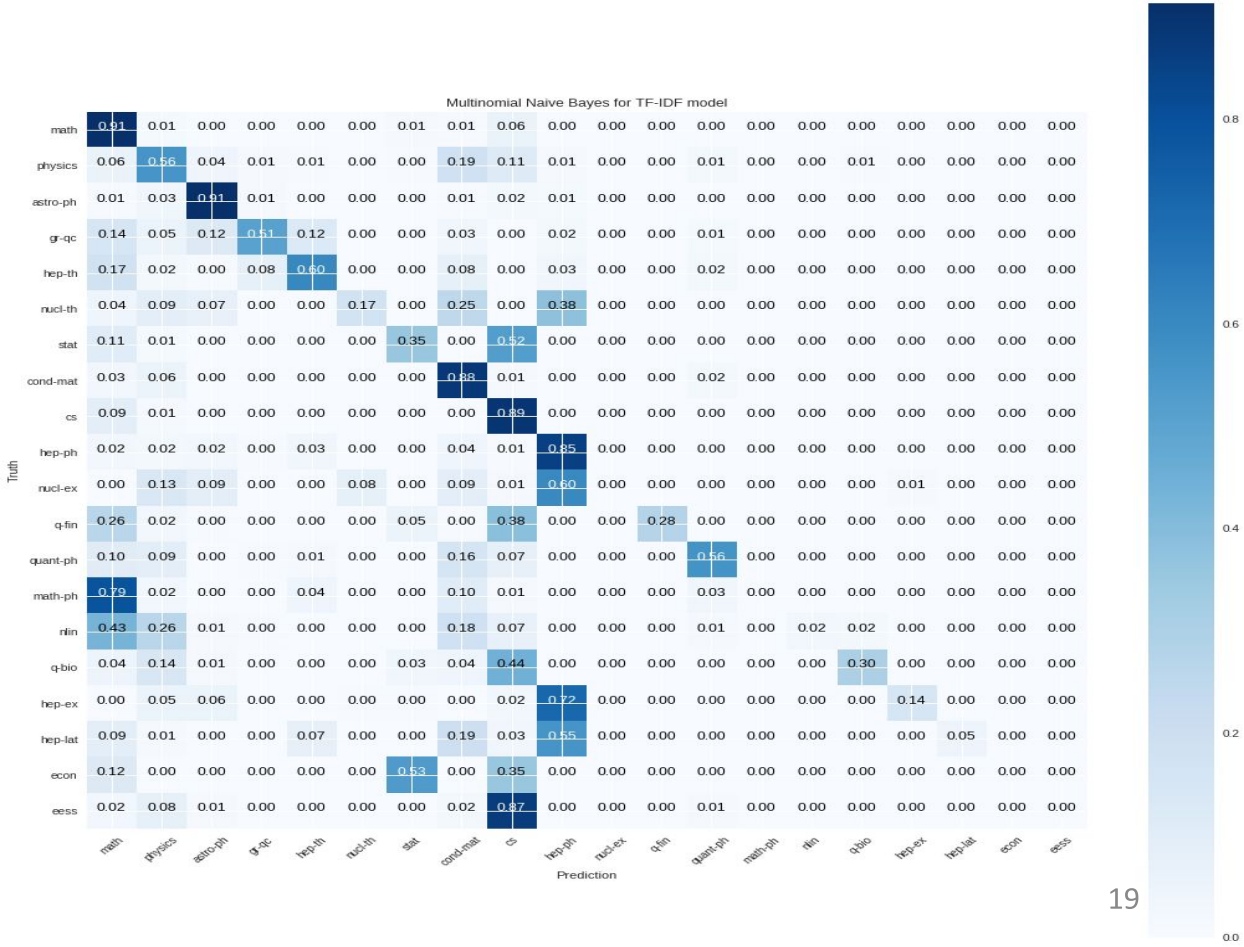
1. Built a multinomial logistic regression model using 10-fold cross validation.
2. Trained it using top 10,000 features.
3. Accuracy: 76.84 %
4. Kappa score: 0.723
5. Precision: 0.76
6. Recall: 0.77
7. F1 score: 0.74
8. Cross validation does not bring about a significant improvement in the results obtained.

Multinomial Naïve Bayes (Cross Validation)

```
y_test_pred = sorted(mnb_cv,key=lambda x:x["kappa"],reverse=True)[0]["model"].predict(X_test)
acc = accuracy_score(y_test_class,y_test_pred)
kap = cohen_kappa_score(y_test_class,y_test_pred)
print("accuracy:",acc)
print("kappa:",kap)
print(classification_report(y_test_class,y_test_pred,target_names=y_class_label))
```

```
accuracy: 0.7684579124579125
kappa: 0.7232815482739645
```

	precision	recall	f1-score	support
math	0.79	0.91	0.84	9054
physics	0.64	0.56	0.59	3187
astro-ph	0.91	0.91	0.91	3980
gr-qc	0.71	0.51	0.59	798
hep-th	0.74	0.60	0.66	1065
nucl-th	0.78	0.17	0.28	379
stat	0.72	0.35	0.47	1204
cond-mat	0.74	0.88	0.80	4324
cs	0.77	0.89	0.83	8125
hep-ph	0.64	0.85	0.73	1354
nucl-ex	0.00	0.00	0.00	132
q-fin	0.86	0.28	0.42	214
quant-ph	0.83	0.56	0.67	1498
math-ph	0.00	0.00	0.00	442
nlin	0.86	0.02	0.04	261
q-bio	0.73	0.30	0.42	504
hep-ex	0.97	0.14	0.25	263
hep-lat	1.00	0.05	0.09	195
econ	0.00	0.00	0.00	17
eess	0.00	0.00	0.00	129
avg / total	0.76	0.77	0.74	37125



Support Vector Machine

1. Built a support vector machine model (linearSVC kernel) using scikit learn.
2. Trained it using top 10,000 features.
3. Accuracy: 80.77 %
4. Kappa score: 0.772
5. Precision: 0.80
6. Recall: 0.81
7. F1 score: 0.80
8. Shows a significant improvement over the logistic regression model.

Support Vector Machine (Cross Validation)

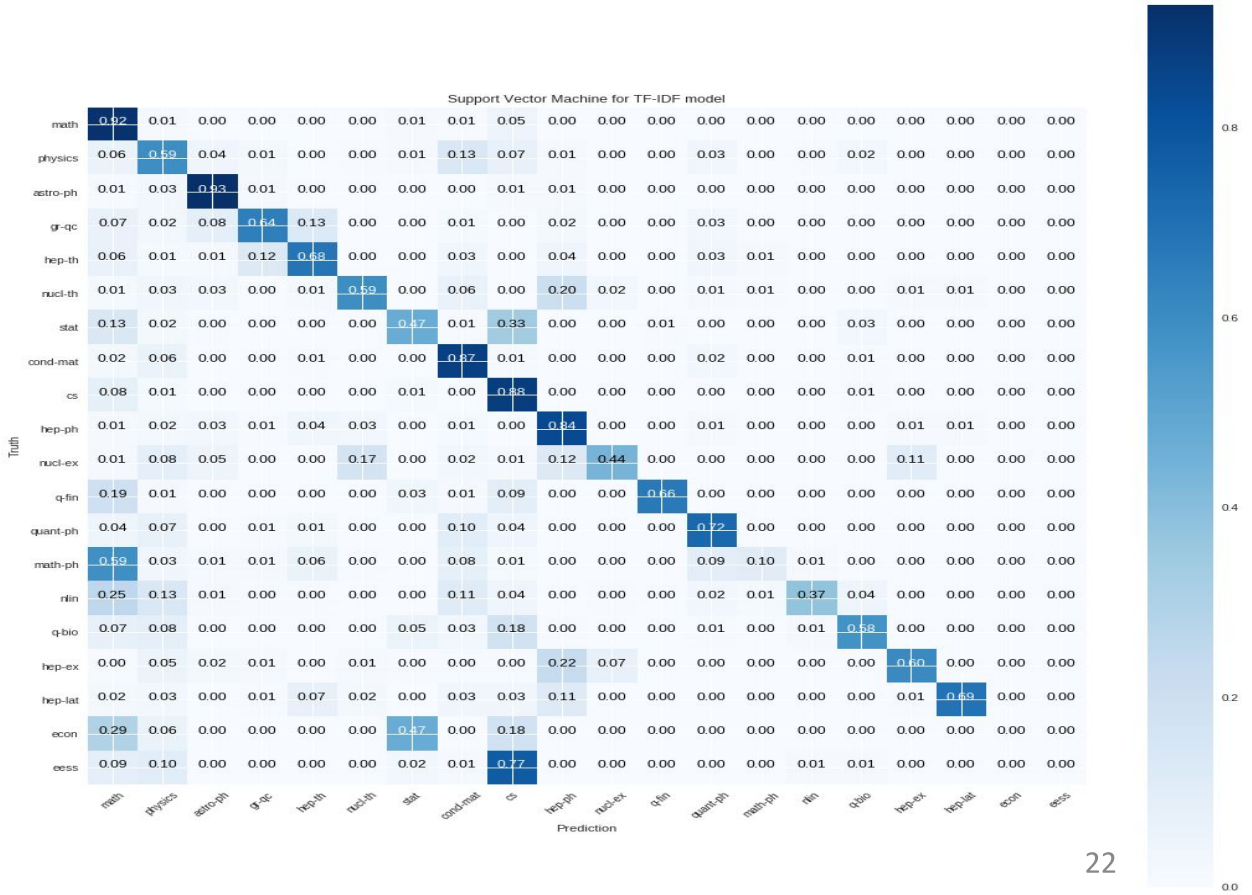
1. Built a support vector machine model using 10-fold cross validation.
2. Trained it using top 10,000 features.
3. Accuracy: 80.72 %
4. Kappa score: 0.772
5. Precision: 0.80
6. Recall: 0.81
7. F1 score: 0.80
8. Cross validation does not bring about a significant improvement in the results obtained.

Support Vector Machine (Cross Validation)

```
y_test_pred = sorted(svm_cv,key=lambda x:x["kappa"],reverse=True)[0]["model"].predict(X_test)
acc = accuracy_score(y_test_class,y_test_pred)
kap = cohen_kappa_score(y_test_class,y_test_pred)
print("accuracy:",acc)
print("kappa:",kap)
print(classification_report(y_test_class,y_test_pred,target_names=y_class_label))
```

```
accuracy: 0.8072727272727273
kappa: 0.7722563365965263
```

	precision	recall	f1-score	support
math	0.83	0.92	0.87	9054
physics	0.68	0.59	0.63	3187
astro-ph	0.93	0.93	0.93	3980
gr-qc	0.67	0.64	0.66	798
hep-th	0.73	0.68	0.70	1065
nucl-th	0.71	0.59	0.65	379
stat	0.67	0.47	0.55	1204
cond-mat	0.82	0.87	0.84	4324
cs	0.84	0.88	0.86	8125
hep-ph	0.79	0.84	0.81	1354
nucl-ex	0.57	0.44	0.50	132
q-fin	0.77	0.66	0.71	214
quant-ph	0.75	0.72	0.74	1498
math-ph	0.51	0.10	0.17	442
nlin	0.62	0.37	0.46	261
q-bio	0.57	0.58	0.57	504
hep-ex	0.79	0.60	0.68	263
hep-lat	0.86	0.69	0.76	195
econ	0.00	0.00	0.00	17
eess	0.00	0.00	0.00	129
avg / total	0.80	0.81	0.80	37125



Neural Network over TF-IDF Vector

1. Built using Keras library in a Sequential method in which we add one layer after another in sequence.
2. First layer: Dense Layer with 5000 neurons connected to the input layer which is a TF-IDF vector representations of the abstracts. Used Rectified Linear Unit (relu) as activation function for this layer.
3. Stacked 3 hidden layers on top of the first layer to make each layer decrease in steps. Thus, hidden layers have 1000, 500 and 100 neurons respectively.
4. Last layer is output layer which is a dense layer. Used softmax as the activation for this layer which means that the output will be normalized as probability.

Neural Network over TF-IDF Vector

5. Model compiled using categorical cross entropy which predicts only one class for a training instance.
6. Training Accuracy: 99%
 Test Accuracy: 80.31%
7. Kappa score: 0.768
8. Precision: 0.80
9. Recall: 0.80
10. F1 score: 0.80
11. Overall performance not better than SVM and Logistic regression

Neural Network over TF-IDF Vector

```
from keras.layers import Input
from keras.models import Model


model = Sequential()
# input layer
model.add(keras.layers.Dense(5000,input_shape=(tfidf.shape[1],),activation='relu'))

# hidden layer
model.add(keras.layers.Dense(1000,activation='relu'))
model.add(keras.layers.Dense(500,activation='relu'))
model.add(keras.layers.Dense(100,activation='relu'))

# output layer
model.add(keras.layers.Dense(len(y_train_label_1),activation='softmax'))
model.compile(loss='categorical_crossentropy',optimizer='adam',metrics=['accuracy'])
```

```
model.summary()
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_1 (Dense)              (None, 5000)              224100000
_________________________________________________________________
dense_2 (Dense)              (None, 1000)              5001000
_________________________________________________________________
dense_3 (Dense)              (None, 500)               500500
_________________________________________________________________
dense_4 (Dense)              (None, 100)               50100
_________________________________________________________________
dense_5 (Dense)              (None, 20)                2020
=================================================================
Total params: 229,653,620
Trainable params: 229,653,620
Non-trainable params: 0
_________________________________________________________________
```

Neural Network over TF-IDF Vector with Dropout Regularization

1. The neural network seemed to overfit on the training data because it achieved an accuracy of 99% over the training data and does not perform well on the test data.
2. Thus, we reconstructed the neural network model by adding a Dropout layer as regularization to avoid overfitting.
3. The input layer has the maximum number of neurons and thus a dropout layer with a regularization parameter of 0.4 was used which means that it will leave 40% of the neurons at random un-updated after each iteration.
4. Next dropout layer was used after the first dense layer with hyperparameter 0.3 which means that it will leave 30% of 5000 neurons un-updated.

Neural Network over TF-IDF Vector with Dropout Regularization

5. Last dropout layer was used after the first hidden layer with hyperparameter 0.2 which means that 80% neurons will be updated

```
from keras.layers import Input,Dropout
from keras.models import Model


model = Sequential()
# input layer
model.add(Dropout(0.4,input_shape=(tfidf.shape[1],)))
model.add(keras.layers.Dense(5000,activation='relu'))
model.add(Dropout(0.3))

# hidden layer
model.add(keras.layers.Dense(1000,activation='relu'))
model.add(Dropout(0.2))
model.add(keras.layers.Dense(500,activation='relu'))
model.add(keras.layers.Dense(100,activation='relu'))

# output layer
model.add(keras.layers.Dense(len(y_train_label_l),activation='softmax'))
model.compile(loss='categorical_crossentropy',optimizer='adam',metrics=['accuracy'])
```

```
model.summary()
```

Layer (type)	Output Shape	Param #
dropout_1 (Dropout)	(None, 44819)	0
dense_1 (Dense)	(None, 5000)	224100000
dropout_2 (Dropout)	(None, 5000)	0
dense_2 (Dense)	(None, 1000)	5001000
dropout_3 (Dropout)	(None, 1000)	0
dense_3 (Dense)	(None, 500)	500500
dense_4 (Dense)	(None, 100)	50100
dense_5 (Dense)	(None, 20)	2020

```
Total params: 229,653,620
Trainable params: 229,653,620
Non-trainable params: 0
```

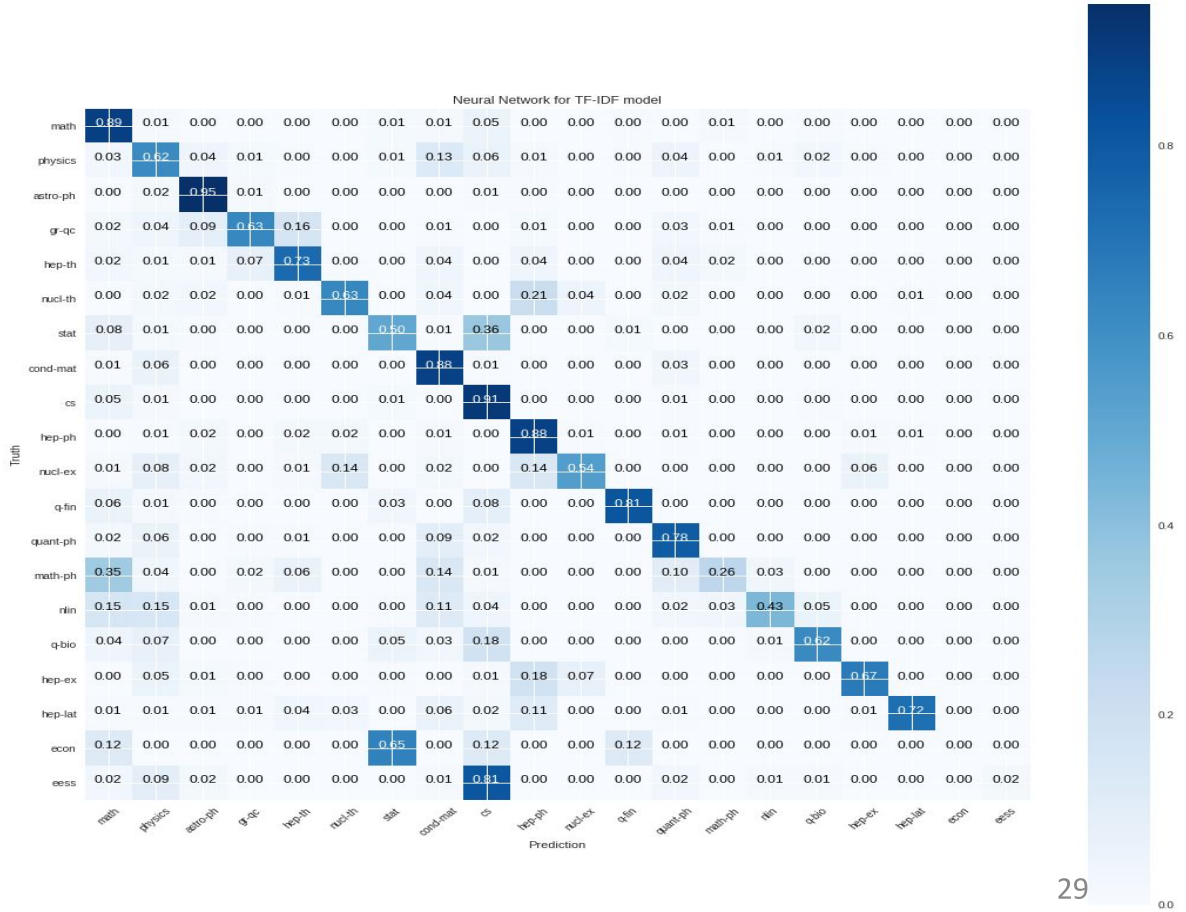
Neural Network over TF-IDF Vector with Dropout Regularization

1. Training accuracy: 95% over 13 epochs.
2. Testing accuracy: 82.41%
3. Kappa score: 0.7932
4. Precision: 0.82
5. Recall: 0.82
6. F1-score: 0.82
7. Best performance obtained till now.
8. Dropout layer proved to be very effective to reduce overfitting due to the training set.

Neural Network over TF-IDF Vector with Dropout Regularization

accuracy: 0.8241077441077441
kappa: 0.7932320593261615

	precision	recall	f1-score	support
math	0.90	0.89	0.89	9054
physics	0.71	0.62	0.66	3187
astro-ph	0.93	0.95	0.94	3980
gr-qc	0.73	0.63	0.68	798
hep-th	0.75	0.73	0.74	1065
nucl-th	0.76	0.63	0.69	379
stat	0.66	0.50	0.57	1204
cond-mat	0.82	0.88	0.85	4324
cs	0.84	0.91	0.87	8125
hep-ph	0.81	0.88	0.85	1354
nucl-ex	0.56	0.54	0.55	132
q-fin	0.67	0.81	0.73	214
quant-ph	0.71	0.78	0.74	1498
math-ph	0.43	0.26	0.32	442
nlin	0.59	0.43	0.50	261
q-bio	0.60	0.62	0.61	504
hep-ex	0.81	0.67	0.73	263
hep-lat	0.90	0.72	0.80	195
econ	0.00	0.00	0.00	17
eess	0.50	0.02	0.03	129
avg / total	0.82	0.82	0.82	37125



Convolutional Neural Network

Additional Preprocessing:

- Use word sequence as the representation of full text for the input layer
- Embedding Layer to transform values from categorical (number) to vector
- Use of Sparse Matrix for memory efficiency

```
from scipy.sparse import lil_matrix
my_l_matrix = lil_matrix((pd_train.shape[0],150),dtype=int)
```

```
max = 150
for i,x in enumerate(abs_noun_title.values):
  if i % 10000 == 0:
    print(i,end=" ")
  x = x.split(" ")
  len_x = len(x) if len(x) < 150 else 150
  #print(len_x)
  feat_x = []
  for my_x in x[:len_x]:
    if len(my_x) > 0:
      try:
        feat_x.append(tf_feature_dict[my_x])
      except:
        pass
  if len(feat_x) > 0:
    my_l_matrix[i,-len(feat_x):] = feat_x
```

Convolutional Neural Network

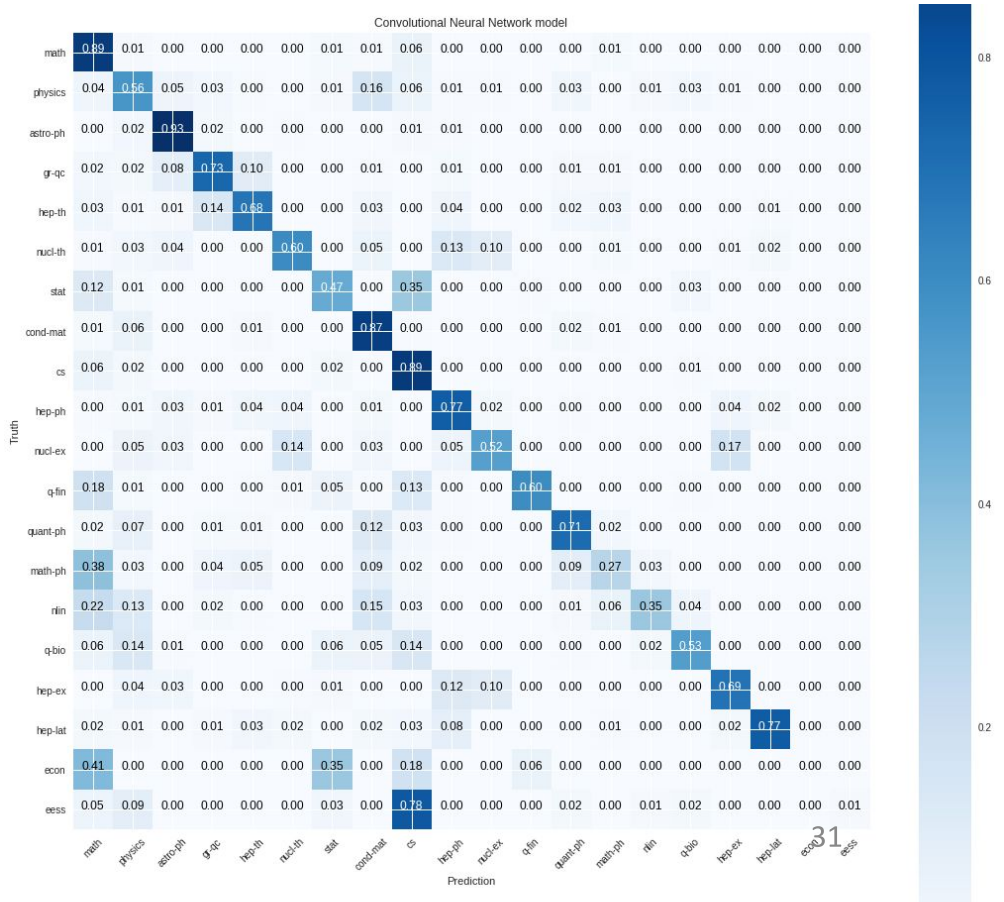
Network Structure:

Layer (type)	Output Shape	Param #
input (InputLayer)	(None, 150)	0
embedding (Embedding)	(None, 150, 300)	3968100
dropout1 (Dropout)	(None, 150, 300)	0
conv1 (Conv1D)	(None, 146, 256)	384256
pool1 (MaxPooling1D)	(None, 48, 256)	0
dropout2 (Dropout)	(None, 48, 256)	0
cov2 (Conv1D)	(None, 46, 256)	196864
pool2 (MaxPooling1D)	(None, 23, 256)	0
dropout4 (Dropout)	(None, 23, 256)	0
flatten (Flatten)	(None, 5888)	0
dense1 (Dense)	(None, 512)	3015168
dropout5 (Dropout)	(None, 512)	0
dense2 (Dense)	(None, 256)	131328
output (Dense)	(None, 20)	5140

Total params: 7,700,856
Trainable params: 7,700,856
Non-trainable params: 0

accuracy: 0.8007272727272727
kappa: 0.7655796878120434

	precision	recall	f1-score	support
math	0.87	0.89	0.88	9054
physics	0.66	0.56	0.61	3187
astro-ph	0.92	0.93	0.92	3980
gr-qc	0.60	0.73	0.66	798
hep-th	0.74	0.68	0.71	1065
nucl-th	0.66	0.60	0.63	379
stat	0.62	0.47	0.54	1204
cond-mat	0.80	0.87	0.84	4324
cs	0.83	0.89	0.86	8125
hep-ph	0.83	0.77	0.80	1354
nucl-ex	0.38	0.52	0.44	132
q-fin	0.72	0.60	0.65	214
quant-ph	0.77	0.71	0.74	1498
math-ph	0.31	0.27	0.29	442
nlin	0.52	0.35	0.42	261
q-bio	0.54	0.53	0.54	504
hep-ex	0.62	0.69	0.65	263
hep-lat	0.78	0.77	0.77	195
econ	0.00	0.00	0.00	17
eess	0.08	0.01	0.01	129
avg / total	0.79	0.80	0.80	37125



Convolutional Neural Network with Pre-trained Embedding

- Use Fasttext pretrained 300 dimension embedding
- Embedding vectors are selected from the features we used in the training

```
#get only feature from the tf_feature_dict

# index words from word2vec
word_vector = []
tfidf_counter = 0
with open("dataset/arxiv/cc.en.300.vec","r") as file:
  i = 0
  for vec in file:
    i+=1
    if i%100000==0:
      print(i)
    #word_index.append((vec.split(" ")[0],i))
    vec = vec.replace("\n","")
    word_idx = vec.split(" ")[0]
    word_coef = np.asarray(vec.split(" ")[1:], dtype='float32')
    try:
      tf_embedding_dict[word_idx]["v"] = word_coef
    except:
      continue
```

```
embedding_temp = []
for x in tf_embedding_dict.items():
  if len(x[1]["v"])>0:
    embedding_temp.append((x[0],x[1]["v"]))
```

```
tf_feature_dict = {}
for i,x in enumerate(embedding_temp):
  tf_feature_dict[x[0]] = i+1
```

```
embedding_weights = np.zeros((len(tf_feature_dict)+1,300))
for i,x in enumerate(embedding_temp):
  embedding_weights[i+1,:] = x[1]
```

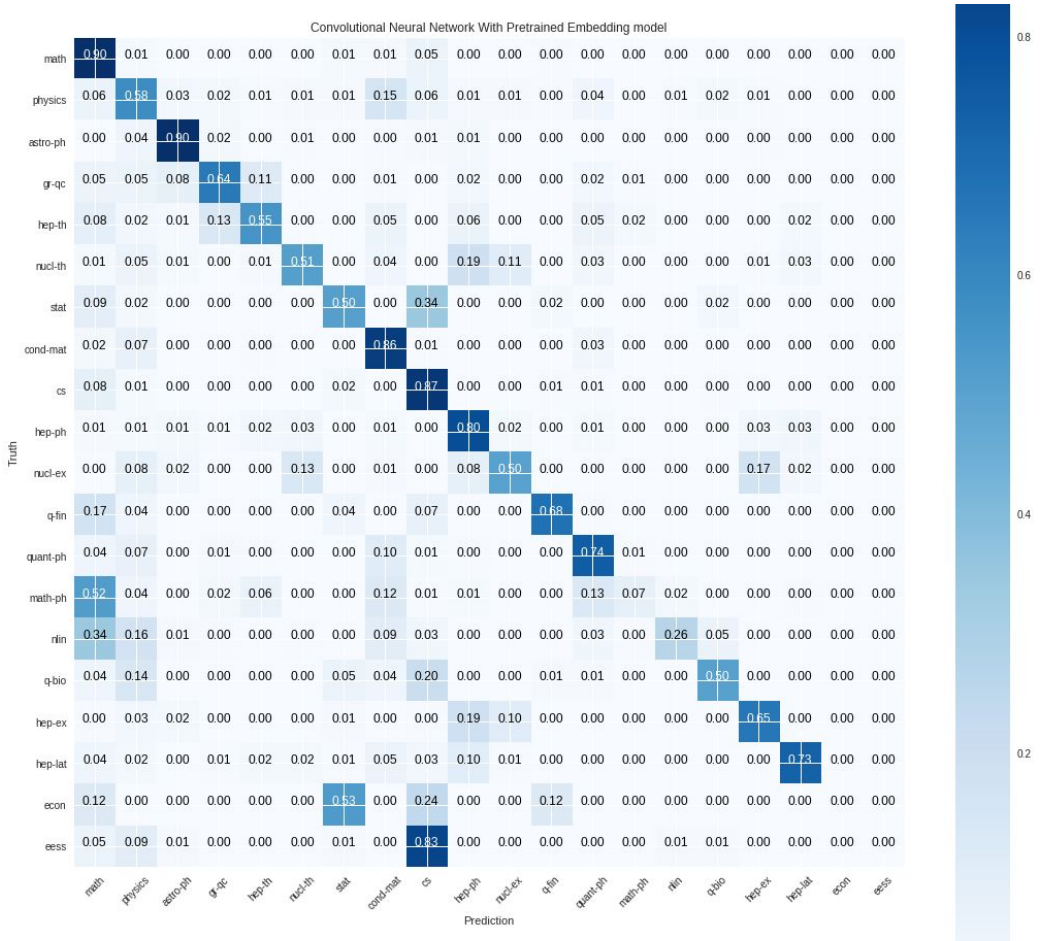
Additional Configuration in the Embedding layer

```
from keras.layers import Input
from keras.layers import Dropout
sequence_input = Input(shape=(max,), dtype='int32',name="input")
embedded_sequences = Embedding(input_dim=len(tf_feature_dict)+1,output_dim=300,
                               weights=[embedding_weights],name="embedding",
                               trainable=False)(sequence_input)
```

Convolutional Neural Network with Pre-trained Embedding

accuracy: 0.7916228956228957
kappa: 0.7543214643472373

	precision	recall	f1-score	support
math	0.84	0.90	0.87	9054
physics	0.64	0.58	0.61	3187
astro-ph	0.94	0.90	0.92	3980
gr-qc	0.63	0.64	0.64	798
hep-th	0.72	0.55	0.62	1065
nucl-th	0.60	0.51	0.55	379
stat	0.63	0.50	0.56	1204
cond-mat	0.80	0.86	0.83	4324
cs	0.84	0.87	0.86	8125
hep-ph	0.76	0.80	0.78	1354
nucl-ex	0.34	0.50	0.40	132
q-fin	0.54	0.68	0.60	214
quant-ph	0.69	0.74	0.72	1498
math-ph	0.34	0.07	0.12	442
nlin	0.53	0.26	0.35	261
q-bio	0.59	0.50	0.54	504
hep-ex	0.64	0.65	0.65	263
hep-lat	0.62	0.73	0.67	195
econ	0.00	0.00	0.00	17
eess	0.00	0.00	0.00	129
avg / total	0.78	0.79	0.78	37125



Used of Pre-trained embedding surprisingly is not improving the model if we are using the same network structure.

Convolutional + LSTM Neural Network

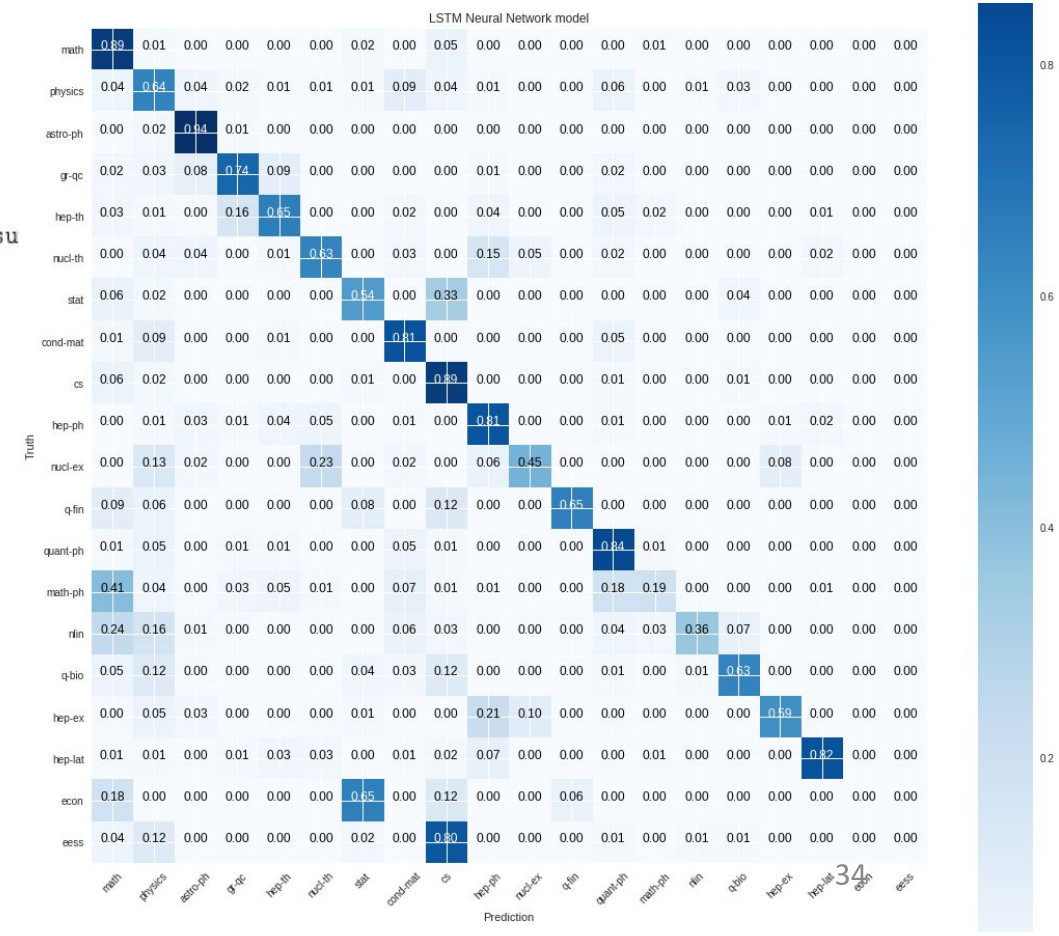
Network Structure

Layer (type)	Output Shape	Param #
input (InputLayer)	(None, 150)	0
embedding (Embedding)	(None, 150, 300)	3658200
cov1 (Conv1D)	(None, 148, 512)	461312
pool1 (MaxPooling1D)	(None, 74, 512)	0
dropout1 (Dropout)	(None, 74, 512)	0
lstm1 (LSTM)	(None, 150)	397800
dropout2 (Dropout)	(None, 150)	0
dense1 (Dense)	(None, 256)	38656
dropout3 (Dropout)	(None, 256)	0
dense2 (Dense)	(None, 128)	32896
output (Dense)	(None, 20)	2580

Total params: 4,591,444
Trainable params: 933,244
Non-trainable params: 3,658,200

accuracy: 0.8125252525252525
kappa: 0.779957696796953

	precision	recall	f1-score	su
math	0.88	0.89	0.89	
physics	0.65	0.64	0.64	
astro-ph	0.92	0.94	0.93	
gr-qc	0.62	0.74	0.68	
hep-th	0.73	0.65	0.69	
nucl-th	0.62	0.63	0.63	
stat	0.65	0.54	0.59	
cond-mat	0.87	0.81	0.84	
cs	0.85	0.89	0.87	
hep-ph	0.81	0.81	0.81	
nucl-ex	0.48	0.45	0.46	
q-fin	0.76	0.65	0.70	
quant-ph	0.65	0.84	0.73	
math-ph	0.38	0.19	0.25	
nlin	0.64	0.36	0.46	
q-bio	0.55	0.63	0.59	
hep-ex	0.82	0.59	0.69	
hep-lat	0.74	0.82	0.78	
econ	0.00	0.00	0.00	
eess	0.00	0.00	0.00	
avg / total	0.81	0.81	0.81	



Neural Network with Multilabel Classifier - Final Model

- Multilabel can produce probability for all classes simultaneously during the training process
- With the same network structure, we use Sigmoid as the activation function on the output layer, and binary_crossentropy for the loss function

```
l_dense2 = Dense(256, activation='relu',name="dense2")(dropout4)
preds = Dense(len(y_train_label_l), activation='sigmoid',name="output")(l_dense2)
```

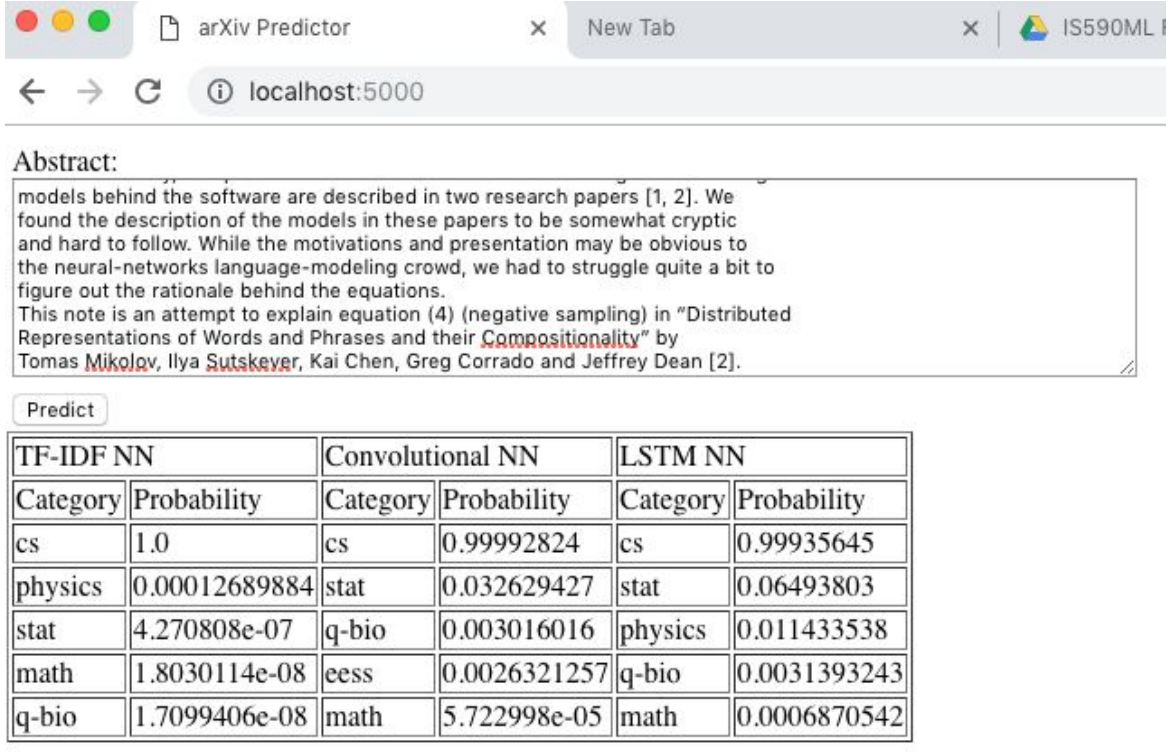
```
model = Model(input=sequence_input, output=preds)

model.compile(loss='binary_crossentropy', optimizer='adam',
              metrics=['categorical_accuracy'])
```

Demo

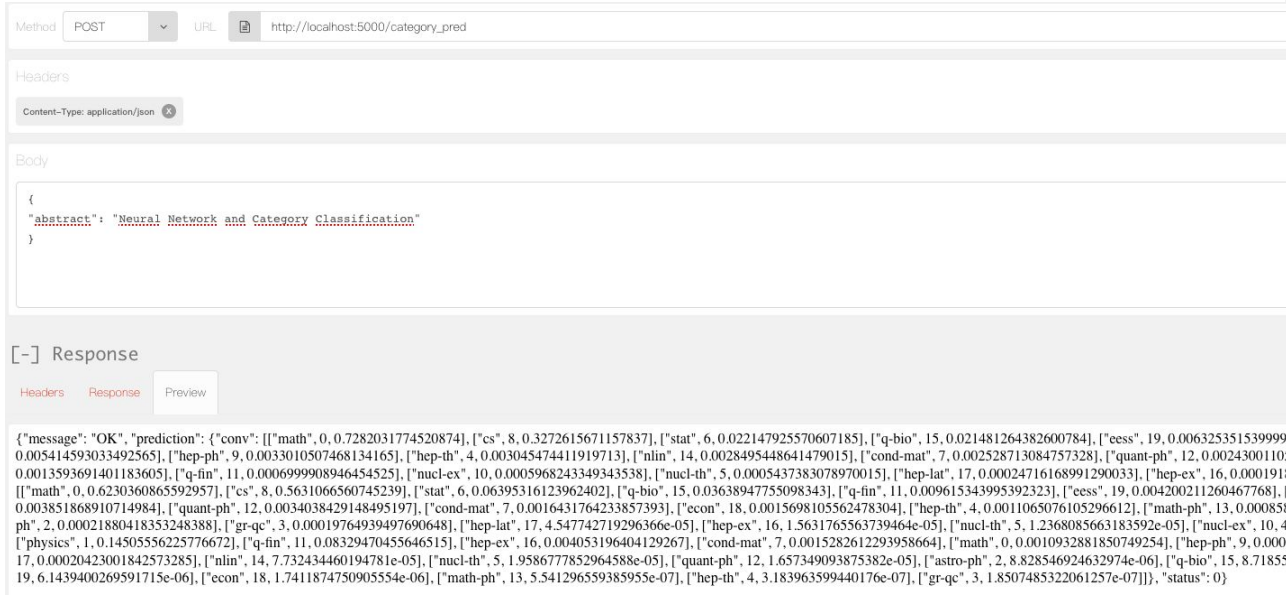
AWS public instance: http://3.16.52.228

Tools



TF-IDF NN		Convolutional NN		LSTM NN	
Category	Probability	Category	Probability	Category	Probability
cs	1.0	cs	0.99992824	cs	0.99935645
physics	0.00012689884	stat	0.032629427	stat	0.06493803
stat	4.270808e-07	q-bio	0.003016016	physics	0.011433538
math	1.8030114e-08	eess	0.0026321257	q-bio	0.0031393243
q-bio	1.7099406e-08	math	5.722998e-05	math	0.0006870542

API



Model Comparison

Model Comparison	Accuracy	Kappa Score	Precision	Recall	F1 Score
Logistic Regression	80.33%	0.767	0.79	0.80	0.79
Multinomial Naive Bayes	76.84%	0.723	0.76	0.77	0.74
Support Vector Machines	80.72%	0.772	0.80	0.81	0.80
Neural Network over TFIDF	82.41%	0.793	0.82	0.82	0.82
Convolutional Neural Network	80.07%	0.765	0.79	0.80	0.80
Convolutional Neural Network with Pre-trained Embedding	79.16%	0.754	0.78	0.79	0.78
Convolutional + LSTM Neural Network	81.25	0.779	0.81	0.81	0.81

Conclusion

- We were able to come up with different models that could predict the category of an article.
- Neural Network models were the models with better figures as compared to other baseline models.
- Although our final model (Convolutional + LSTM model) has the same results as far as all performance parameters are concerned when compared to other Neural network models, it comparatively uses fewer parameters thereby making the model more efficient and robust.

Code Available on GitHub

https://github.com/nikolausn/IS598ML-FINAL

.idea	add pickle model and website sources	8 days ago
data-by-year	Added data files	22 hours ago
feature-extraction	Added noun feature files	20 hours ago
features/nouns	Added noun feature files	20 hours ago
multi-label-models	Implemented logistic regression and SVM	6 hours ago
website	add categories definition	16 hours ago
.DS_Store	Added noun feature files	20 hours ago
.gitignore	Added data files	22 hours ago
4th_model_nn_conv_strata_drop_s...	add pickle model and website sources	8 days ago
5th_model_nn_lstm_strata_drop_s...	add pickle model and website sources	8 days ago
README.md	Create README.md	a month ago
abstract_noun.pickle	add pickle model and website sources	8 days ago
abstract_noun_clean.pickle	add pickle model and website sources	8 days ago
arxiv-dataset.ipynb	add csv generator	22 days ago
arxiv_tar_json_csv.ipynb	add csv generator	22 days ago
data-exploration.ipynb	Added noun feature files	20 hours ago

Ongoing and Future Work

- Multi-label classification
  - Baseline models finished: Naive Bayes, Logistic Regression, and SVM
  - To do: neural network models
- Try different features
  - Phrases
  - Whole sentences

References

arXiv - https://arxiv.org/

fastText - https://fasttext.cc/

scikit-learn documentation - https://scikit-learn.org/stable/supervised_learning.html

keras documentation - https://keras.io/