Fake News Identification: Logistic Regression and Support-Vector Machine

Machine Learning
Nishant Date and Nick Tran
April 28, 2021

Introduction and Hypothesis

Fake news has become a major issue today, but there is an issue identifying what is true and what is not. This report details how Support-vector-machines (SVM) and Logistic Regression can be used to identify the truth to try to classify statements, made by different politicians, as either fake or true. The dataset used is 'Liar, Liar pants on fire' dataset, provided by William Wang, william@cs.ucsb.edu. There are 14 fields and 10,240 examples in the training and test sets.

We hypothesize that Support-Vector Machines and Logistic Regression will be able to classify fake and real news with at least 70% accuracy. The business applications of identifying fake news can be applied to social media companies that need to monitor the content that is shared on their sites.

Description of Dataset

The dataset used is Liar: A Benchmark Dataset for Fake News Detection from William Yang Wang. There are 14 features:

Column 1: the ID of the statement ([ID].json).
Column 2: the label
Column 3: the statement.
Column 4: the subject(s).
Column 5: the speaker.
Column 6: the speaker's job title.
Column 7: the state info.
Column 8: the party affiliation.
Column 9-13: the total credit history count, including the current statement.
9: barely true counts.
10: false counts.
11: half true counts.
12: mostly true counts.

13: pants on fire counts.
Column 14: the context (venue / location of the speech or statement).

Example of the training dataset

	ID	label	statement	subject(s)	speaker	speaker title	state	party	barely true	false	half-true	mostly-true	pants on fire	context
0	2635.json	false	Says the Annies List political group supports ...	abortion	dwayne-bohac	State representative	Texas	republican	0.0	1.0	0.0	0.0	0.0	a mailer
1	10540.json	half-true	When did the decline of coal start? It started...	energy,history,job-accomplishments	scott-surovell	State delegate	Virginia	democrat	0.0	0.0	1.0	1.0	0.0	a floor speech.
2	324.json	mostly-true	Hillary Clinton agrees with John McCain "by vo...	foreign-policy	barack-obama	President	Illinois	democrat	70.0	71.0	160.0	163.0	9.0	Denver
3	1123.json	false	Health care reform legislation is likely to ma...	health-care	blog-posting	NaN	NaN	none	7.0	19.0	3.0	5.0	44.0	a news release
4	9028.json	half-true	The economic turnaround started at the end of ...	economy,jobs	charlie-crist	NaN	Florida	democrat	15.0	9.0	20.0	19.0	2.0	an interview on CNN

Logistic Regression

Below is a table that lists the feature encodings that were done to the dataset. The columns: ID, statement, subject(s), speaker, speaker-title, and context were excluded from the feature vector. Instead of having to predict 6 labels (pants-on-fire, false, barely-true, half-true, mostly-true, and true), each label is classified to be true (1) or false (0), this is referred to as Binary Classification throughout this report. The labels: ‘Half-true’, ‘barely-true’, and ‘true’ are changed to 1 and the labels ‘false’ and ‘pants-on-fire’ are changed to 1. We also created a new feature called the truth-score. It is the probability that the speaker will tell the truth based on previous statements made.

Feature Encoding Table 1

1	2	3	4	5	6	7	8	9
Label	State	Party	Barely True	False	half-true	mostly-true	Pants-on fire	Truth-score
Binary and Label Encoder	LabelEncoder	LabelEncoder	-	-	-	-	-	See below

- Label = {'pants-fire':0, 'false':0,'barely-true':1, 'half-true':1, 'mostly-true':1, 'true':1}
- Truth-score = (barely true + half-true + mostly-true) / (total number of statements)

	ID	label	statement	subject(s)	speaker	speaker title	state	party	barely true	false	half-true	mostly-true	pants on fire	context	truth_score
0	2635.json	0	Says the Annies List political group supports ...	abortion	dwayne-bohac	State representative	Texas	republican	0.0	1.0	0.0	0.0	0.0	a mailer	0.000000
1	10540.json	1	When did the decline of coal start? It started...	energy,history,job-accomplishments	scott-surovell	State delegate	Virginia	democrat	0.0	0.0	1.0	1.0	0.0	a floor speech.	1.000000
2	324.json	1	Hillary Clinton agrees with John McCain "by vo...	foreign-policy	barack-obama	President	Illinois	democrat	70.0	71.0	160.0	163.0	9.0	Denver	0.830867
3	1123.json	0	Health care reform legislation is likely to ma...	health-care	blog-posting	NaN	NaN	none	7.0	19.0	3.0	5.0	44.0	a news release	0.192308
4	9028.json	1	The economic turnaround started at the end of ...	economy,jobs	charlie-crist	NaN	Florida	democrat	15.0	9.0	20.0	19.0	2.0	an interview on CNN	0.830769

Figure 1: Dataset after adding truth_score and binary classification on Label column

Table 2: Results for Binary-Classified Labels

Test	Regularization	Feature Transformation	Features Passed In	Training Accuracy	Testing Accuracy
1	L1	None	Features[2, 3, 4-8]	72.32%	73.1%
2	L2	None	Features[2, 3, 4-8]	73.80%	72.45%
3	L1	None	Features[9]	80.30%	80.43%
4	L2	None	Features[9]	80.30%	80.43%
5	L1	None	Features[2, 3, 9]	79.96%	80.82%
6	L2	None	Features[2, 3, 9]	80.24%	80.66%
7	L1	Polynomial Degree 3	Features[2, 3, 9]	80.00%	81.16%
8*	**L2**	**Polynomial Degree 3**	**Features[2, 3, 9]**	**79.96%**	**81.77%**
9	L1	Polynomial Degree 3	Features[2,3,4-9]	73.46%	74.66%
10	L2	Polynomial Degree 3	Features[2,3,4-9]	74.05%	76.00%

***Test 8 yielded the best testing accuracy**

Test 8 yielded the best testing accuracy with 81.77%. For reference, we have also included the results of running logistic regression on the dataset without using binary classification on the labels (Figure 2 and Table 3). The labels column was encoded using the sklearn package: LabelEncoder. Refer to Table 5 for the graphs associated with these tests.

	ID	label
0	2635.json	1
1	10540.json	2
2	324.json	3
3	1123.json	1
4	9028.json	2

Figure 2: LabelEncoder on Label column

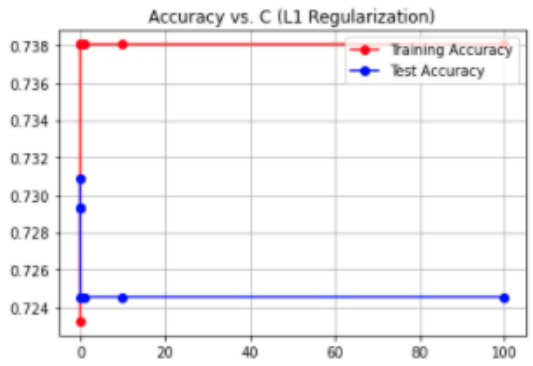
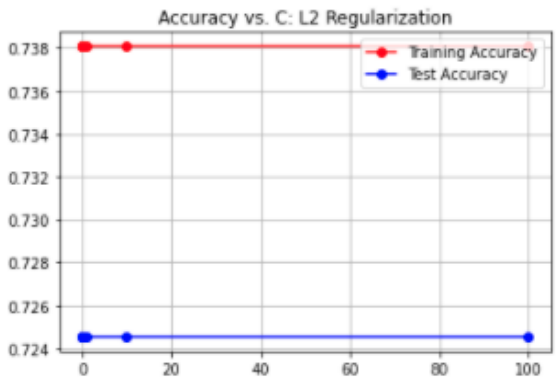
Table of Results 3: Without Binary Classification of Labels

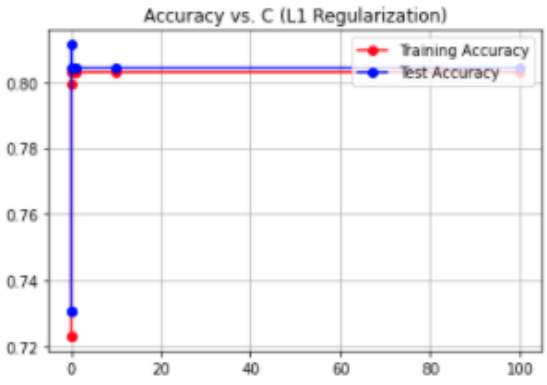
Test	Regularization	Feature Transformation	Features Passed In	Training Accuracy	Testing Accuracy
A	L1	None	Features[4-8]	32.67%	31.57%
B	L2	None	Features[4-8]	28.05%	27.70%
C	**L1**	**Polynomial Degree 3**	**Features[2,3,4-9]**	**34.46%**	**36.23%**
D*	L2	Polynomial Degree 3	Features[2,3,4-9]	31.96%	32.99%

***Test C yielded the best results**

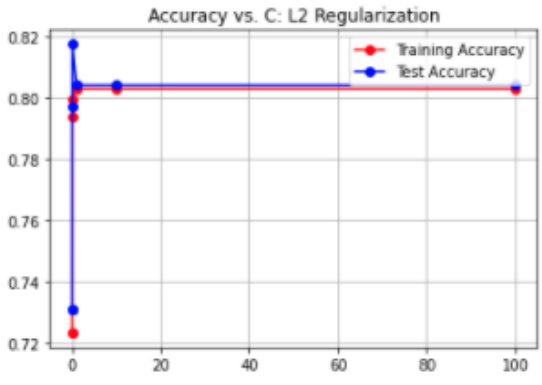
```
cVals = [0.0001, 0.001, 0.01, 0.1, 1, 10, 100]
```

Table 4: Test Graphs for Binary Classified Labels

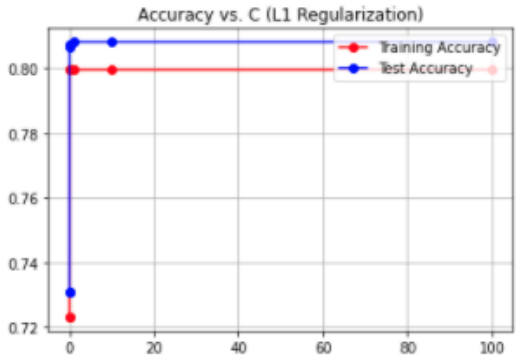
Test 1	Test 2
	
Test 3	Test 4

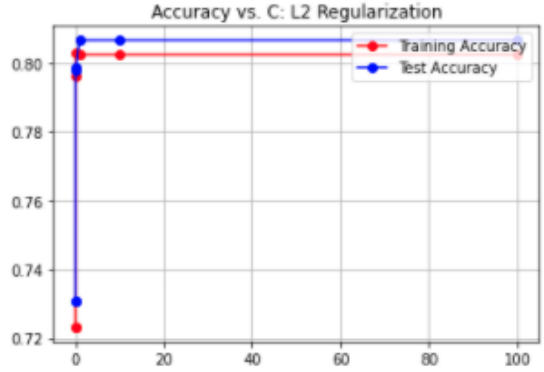


Test 5



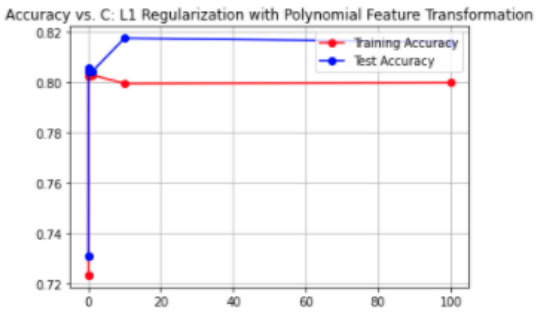
Test 6



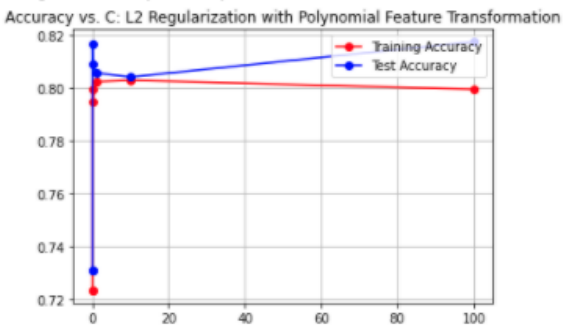


.

Test 7



Test 8



Test 9

Test 10

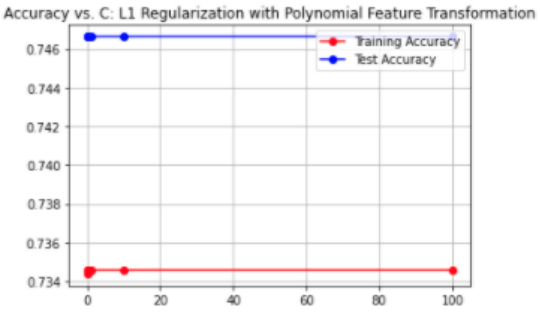
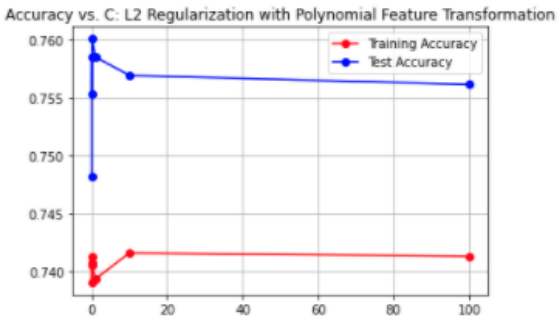
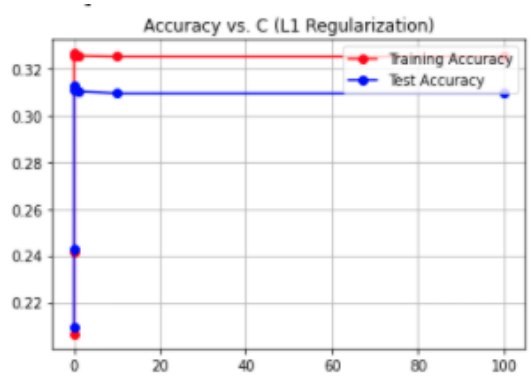
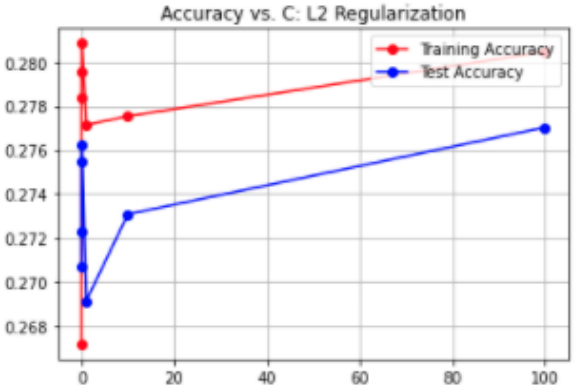
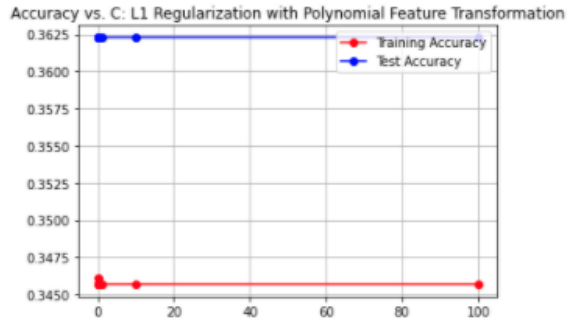
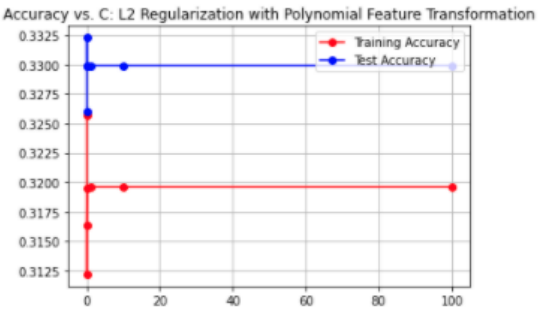
	

Table 5: Test Graphs for Non-Binary Classified Labels

Test A	Test B
	
Test C	**Test D**
	

Baseline Function

Support Vector Machines (SVM)

We decided to use the binary-classified labels (Figure 1) data to train the SVM to predict. Here are our results:

Kernel	Training Accuracy	Training Accuracy
Linear	79.76%	81.37%
RBF	72.32%	73.08%
Polynomial	72.32%	73.08%

The Linear Kernel yielded the best training accuracy.

Linear Kernel

```
c_svm_linear = []
acc_test_svm_linear = []
acc_train_svm_linear = []
from sklearn import svm
def svm_linear(c):
    svc_linear = svm.SVC(probability = False, kernel = 'linear', C = c)
    svc_linear.fit(X_train, Y_train)
    Yhat_svc_linear_train = svc_linear.predict(X_train)
    acc_train_linear = np.mean(Yhat_svc_linear_train == Y_train)
    acc_train_svm_linear.append(acc_train_linear)
    print('Train Accuracy = {0:f}'.format(acc_train_linear))
    Yhat_svc_linear_test = svc_linear.predict(X_test)
    acc_test = np.mean(Yhat_svc_linear_test == Y_test)
    acc_test_svm_linear.append(acc_test)
    print('Test Accuracy = {0:f}'.format(acc_test))
    c_svm_linear.append(c)

for c in cVals:
    svm_linear(c)
```

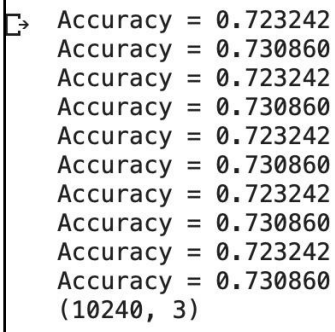
```
Train Accuracy = 0.723242
Test Accuracy = 0.730860
Train Accuracy = 0.723242
Test Accuracy = 0.730860
Train Accuracy = 0.794922
Test Accuracy = 0.797948
Train Accuracy = 0.797559
Test Accuracy = 0.813733
Train Accuracy = 0.797559
Test Accuracy = 0.812944
```

RBF Kernel

```
acc_train_svm_rbf = []
acc_test_svm_rbf = []
c_svm_rbf = []

from sklearn import svm
def svm_rbf(c):
    svc_rbf = svm.SVC(probability = False, kernel = 'rbf', C = c)
    svc_rbf.fit(X_train, Y_train)
    Yhat_svc_linear_train = svc_rbf.predict(X_train)
    acc_train = np.mean(Yhat_svc_linear_train == Y_train)
    acc_train_svm_rbf.append(acc_train)
    print('Accuracy = {0:f}'.format(acc_train))
    Yhat_svc_rbf_test = None
    acc_test = None
    Yhat_svc_linear_test = svc_rbf.predict(X_test)
    acc_test = np.mean(Yhat_svc_linear_test == Y_test)
    acc_test_svm_rbf.append(acc_test)
    print('Accuracy = {0:f}'.format(acc_test))
    c_svm_rbf.append(c)
for c in cVals:
    svm_rbf(c)
X_train.shape
```

```
Train Accuracy = 0.723242
Test Accuracy = 0.730860
Train Accuracy = 0.723242
Test Accuracy = 0.730860
Train Accuracy = 0.723242
Test Accuracy = 0.730860
Train Accuracy = 0.723242
Test Accuracy = 0.730860
Train Accuracy = 0.723242
Test Accuracy = 0.730860
```

Polynomial Kernel		

```
acc_train_svm_poly = []
acc_test_svm_poly = []
c_svm_poly = []
def svm_polynomial(c):
    svc_polynomial = svm.SVC(probability = False, kernel = 'poly', C = c)
    A = X_train
    B = Y_train
    C = X_test
    D = Y_test
    svc_polynomial.fit(A, B)
    Yhat_svc_polynomial_train = svc_polynomial.predict(A)
    acc_train = np.mean(Yhat_svc_polynomial_train == B)
    acc_train_svm_poly.append(acc_train)
    print('Train Accuracy = {0:f}'.format(acc_train))
    Yhat_svc_polynomial_test = svc_polynomial.predict(C)
    acc_test = np.mean(Yhat_svc_polynomial_test == D)
    acc_test_svm_poly.append(acc_test)
    print('Test Accuracy = {0:f}'.format(acc_test))
    c_svm_poly.append(c)

cVals = np.geomspace(0.000001, 0.001, 5)
for c in cVals:
    svm_polynomial(c)
```

```
Accuracy = 0.723242
Accuracy = 0.730860
Accuracy = 0.723242
Accuracy = 0.730860
Accuracy = 0.723242
Accuracy = 0.730860
Accuracy = 0.723242
Accuracy = 0.730860
Accuracy = 0.723242
Accuracy = 0.730860
(10240, 3)
```

Neural Networks

Neural Network

Hidden Layers - 5

Input Layers - 3

Output layers - 2

3000 iterations

```
from sklearn.preprocessing import StandardScaler  # It is important in neural networks to scale the date
from sklearn.model_selection import train_test_split  # The standard - train/test to prevent overfitting and choose hyperparamet
from sklearn.metrics import accuracy_score #
import numpy as np
import numpy.random as r # We will randomly initialize our weights
import matplotlib.pyplot as plt

def f(z):
    return 1 / (1 + np.exp(-z))
def f_deriv(z):
    return f(z) * (1 - f(z))

def setup_and_init_weights(nn_structure):
    W = {} #creating a dictionary i.e. a set of key: value pairs
    b = {}
    for l in range(1, len(nn_structure)):
        W[l] = r.random_sample((nn_structure[l], nn_structure[l-1])) #Return "continuous uniform" random floats in the half-open
        b[l] = r.random_sample((nn_structure[l],))
    return W, b
def init_tri_values(nn_structure):
    tri_W = {}
    tri_b = {}
    for l in range(1, len(nn_structure)):
        tri_W[l] = np.zeros((nn_structure[l], nn_structure[l-1]))
        tri_b[l] = np.zeros((nn_structure[l],))
    return tri_W, tri_b
def feed_forward(x, W, b):
    a = {1: x} # create a dictionary for holding the a values for all levels
    z = { } # create a dictionary for holding the z values for all the layers
    for l in range(1, len(W) + 1): # for each layer
        node_in = a[l]
        z[l+1] = W[l].dot(node_in) + b[l]  # z^(l+1) = W^(l)*a^(l) + b^(l)
        a[l+1] = f(z[l+1]) # a^(l+1) = f(z^(l+1))
    return a, z
```

```
def calculate_out_layer_delta(y, a_out, z_out):
    # delta^(nl) = -(y_i - a_i^(nl)) * f'(z_i^(nl))
    return -(y-a_out) * f_deriv(z_out)
def calculate_hidden_delta(delta_plus_1, w_l, z_l):
    # delta^(l) = (transpose(W^(l)) * delta^(l+1)) * f'(z^(l))
    return np.dot(np.transpose(w_l), delta_plus_1) * f_deriv(z_l)

def train_nn(nn_structure, X, y, iter_num=3000, alpha=0.25):
    W, b = setup_and_init_weights(nn_structure)
    cnt = 0
    lmbda = 0.15
    N = len(y)
    row_len = X.shape[1]
    avg_cost_func = []
    print('Starting gradient descent for {} iterations'.format(iter_num))
    while cnt < iter_num:
        if cnt%100 == 0:
            print('Iteration {} of {}'.format(cnt, iter_num))
        tri_W, tri_b = init_tri_values(nn_structure)
        avg_cost = 0
        for i in range(N):
            delta = {}
            a, z = feed_forward(X.iloc[i,0:row_len], W, b)
            for l in range(len(nn_structure), 0, -1):
                if l == len(nn_structure):
                    delta[l] = calculate_out_layer_delta(y[i], a[l], z[l])
                    avg_cost += np.linalg.norm((y[i]-a[l]))
                    #print("lin alg done")
                else:
                    if l > 1:
                        delta[l] = calculate_hidden_delta(delta[l+1], W[l], z[l])
                        #print("hidden delta done")
                    # triW^(l) = triW^(l) + delta^(l+1) * transpose(a^(l))
                    tri_W[l] += np.dot(delta[l+1][:,np.newaxis], np.transpose(a[l][:,np.newaxis]))
                    tri_b[l] += delta[l+1]
```

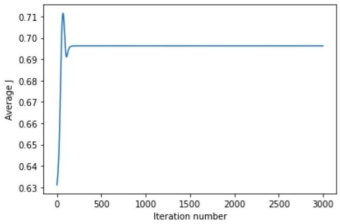
```
        for l in range(len(nn_structure) - 1, 0, -1):
            W[l] += -alpha * (1.0/N * tri_W[l])
            b[l] += -alpha * (1.0/N * tri_b[l])
        avg_cost = 1.0/N * avg_cost
        avg_cost_func.append(avg_cost)
        cnt += 1
    return W, b, avg_cost_func

def predict_y(W, b, X, n_layers):
    N = X.shape[0]
    row_len = X.shape[1]
    y = np.zeros((N,))
    for i in range(N):
        a, z = feed_forward(X.iloc[i,0:row_len], W, b)
        y[i] = np.argmax(a[n_layers])
    return y

nn_structure = [3, 4, 2]


# train the NN
print(Y_train.shape)
W, b, avg_cost_func = train_nn(nn_structure, X_train, Y_train, 3000)
```

```
# plot the avg_cost_func
plt.plot(avg_cost_func)
plt.ylabel('Average J')
plt.xlabel('Iteration number')
plt.show()
# get the prediction accuracy and print
y_pred = predict_y(W, b, X_test, 3)
print('Prediction accuracy is {}%'.format(accuracy_score(Y_test, y_pred) * 100))
```

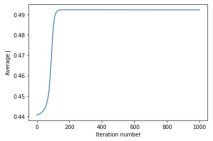



```
Prediction accuracy is 56.27466456195738%
```

56% accuracy

Neural Network
1000 iterations
1 Output Layer
6 Hidden Layers

```
# plot the avg_cost_func
plt.plot(avg_cost_func)
plt.ylabel('Average J')
plt.xlabel('Iteration number')
plt.show()
# get the prediction accuracy and print
y_pred = predict_y(W, b, X_test, 3)
print('Prediction accuracy is {}%'.format(accuracy_score(Y_test, y_pred) * 100))
```



43.6% accuracy

Concluding Remarks

In a world with increasingly unreliable media and harder to qualify statements -- we think that fake news is a growing plague upon the internet and world in general. It allows actual truths to seep through the cracks while fake ideas and information are used as talking points for dangerous precedents. In this paper we wanted to create a tool to filter through these ideas such that that truth gets recognized and what is fake can be knowingly dismissed.

The dataset we had statements of various political and public figures in the political space. The features of the dataset included the statements they made and the level of truth assigned to them by PolitiFact. We wanted to use this to ascertain new statements made by these individuals and categorize how true or false they may be based on these features.

To process the data we first normalized the labels and made it a binary classification of True/Untrue. We then normalized the features giving each data entry a “Truth Score” based on their previous record of telling the truth. Using these features we created various Machine Learning models on the training data.
For a dry test of our hypothesis -- we created a baseline that simply returned the median of the dataset labels. This allows us to see exactly how much better our predictions are doing on the dataset.

We used various forms of logistic regression with l1 regularization - l2 regularization and polynomial regression. We used support vector machines with Linear, RBF , and polynomial features kernels. Finally, we ran a neural network with varying hidden layers and iterations. The logistic regression and support vector machines gave a consistent average of roughly 70 percent on the data used while the Neural Network peaked at the same with fewer iterations and did worse as the iterations increased. The loss of accuracy with increasing iterations could be attributed to overfitting the data and losing some general trends seen in the training data.

We were able to derive an average of 70 percent accuracy through the supervised learning mechanisms stated above, beating the median score by roughly 30 percent. Showing that tools like this can be reliable in filtering fake news can have a profound impact on politics. Disallowing and automatically filtering malicious bots on the internet that attempt to inject misinformation in the public eye.

start with an overview of the problem they are solving
then go on to discuss any preprocessing steps
then discuss individually each of the algorithms (+ regularization/hyper parameter tuning, feature transformations)
and finally discuss what conclusions can be drawn about what works best and why.

Sources

LIAR: A BENCHMARK DATASET FOR FAKE NEWS DETECTION

William Yang Wang, "Liar, Liar Pants on Fire": A New Benchmark Dataset for Fake News Detection, to appear in Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (ACL 2017), short paper, Vancouver, BC, Canada, July 30-August 4, ACL.