Predicting and Evaluating the Popularity of Online News

Final Presentation

Team 35
Dipankar Niranjan
Sriharsh Bhyravajjula
Susobhan Ghosh

ABSTRACT

- The dataset consists of various features extracted from Mashable’s Online News Articles (provided by UCI ML repo, originally acquired and preprocessed by K.Fernandes et al.)
- The number of shares under a News Article indicates how popular it is.
- Our ‘classification’ task is then to use the extracted features and predict whether a news article is popular or not by predicting the number of shares.
- Please note that actually predicting the number of shares is a ‘regression’ task, which yields sub par results. So what we’ve done is to categorize the news articles into two classes - ‘Popular’ (≥ 1400 shares) and ‘Not popular’ (< 1400 shares) and hence it becomes a binary ‘classification’ task.
- Using this categorization, the online news companies can predict the news popularity before publication.

DATASET

- The dataset consists of 39797 instances with 58 predictive features (and 1 goal field - num of shares) with no missing values for all the features.
- There is a split of 46%-54% of the instances amongst the two classes - which is a pretty even split - data is not skewed towards one class and this helps in some of the classifiers.
- The features consist of both discrete and continuous features which have to be handled differently for some classifiers.
- Cleary, using all the features for all the samples is intractable. We have to rely on one of PCA/LDA/Feature selection using Fisher Criterion.
- Another point to be noted is that there was no separate training and test data - all the samples are in a single file. So we have to split the data into train and test and employ cross validation to test our classifiers.

FEATURE SELECTION/DIMENSIONALITY REDUCTION

- The first thing we did was to employ scikit-learn’s PCA to reduce the dimensionality from k = 58 to k = 10, 20. Then we implemented Gaussian Naive Bayes and linear regression to check the performance of classification. The error rate was high and the results were unsatisfactory for both the new values of k (as high as 90 % in the case of linear regression - classification). A reason for this could be that the feature set is pretty much non-correlated.
- The next thing in mind is the LDA. Since the number of samples is not so large that we need to reduce the feature set from 58 dimensions to the 1 dimensional LDA space, we concluded that this method of dimension reduction is an overkill.
- An improvisation over the LDA is to select a subset of the k = 58 features using Fischer scores. We tested the performance of the linear regression classifier for k = 10, 20 and 30 and found that k = 20 was yielding the best results amongst the three.

FEATURE SELECTION USING FISCHER SCORES

The Fischer Score of the jth feature (for a binary classification problem) is given by:

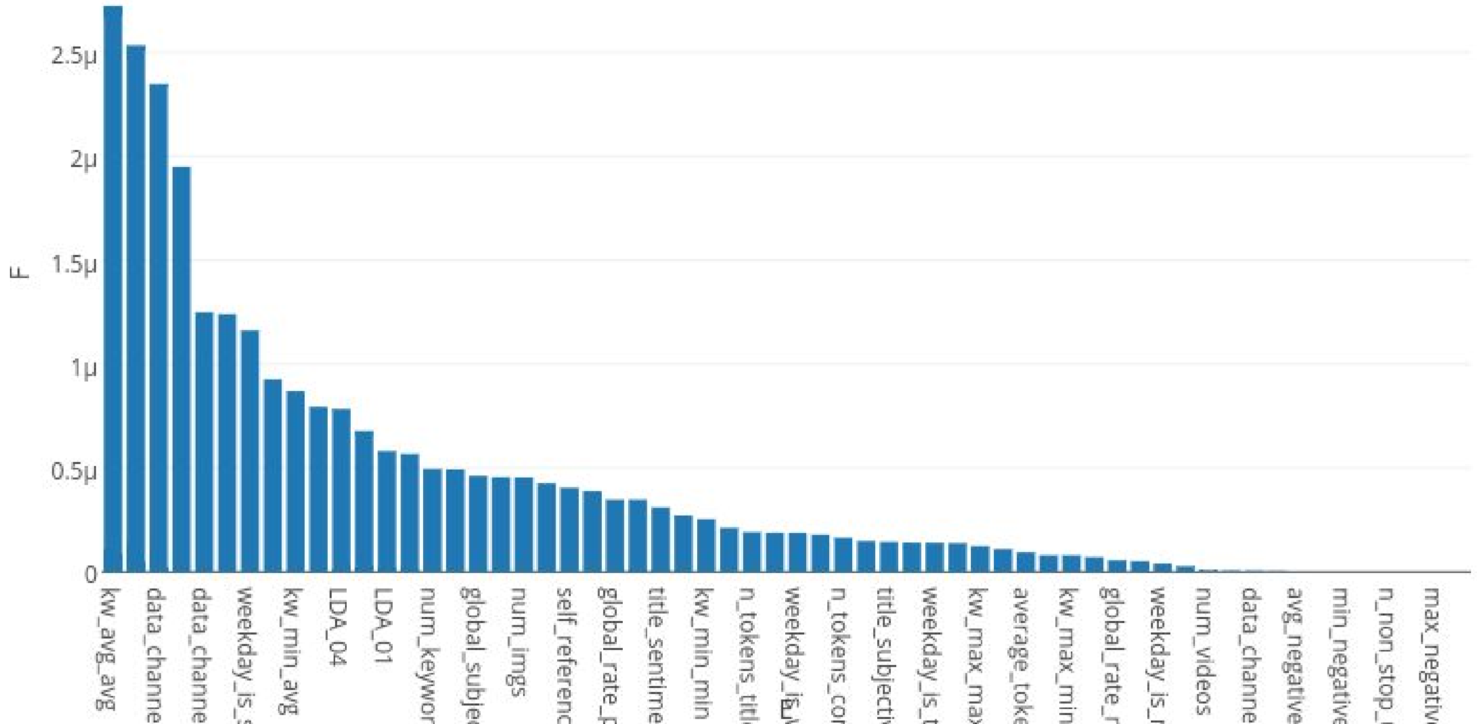
$$F(j) = \frac{(\bar{x}_j^1 - \bar{x}_j^2)^2}{(s_j^1)^2 + (s_j^2)^2}$$

where

$$(s_j^k)^2 = \sum_{x \in X^k} (x_j - \bar{x}_j^k)^2$$

We aim to maximize the between class scatter - the numerator, and to minimize the within class scatter - the denominator.
Thus, higher the F Score, the more likely it is that this feature is more discriminative.
We selected 20 features with the highest F scores.

FISHER SCORES



SELECTED FEATURES (both continuous and discrete)

'Kw_avg_avg',
'LDA_02',
'Data_channel_is_world',
'is_weekend',
'data_channel_is_socmed',
'weekday_is_saturday',
'LDA_04',
'data_channel_is_entertainment',
'data_channel_is_tech',
'kw_max_avg',
'weekday_is_sunday',
'LDA_00',
'num_hrefs',
'global_subjectivity',
'kw_min_avg',
'global_sentiment_polarity',
'rate_negative_words',
'num_keywords',
'num_imgs',
'LDA_01'

LMS WIDROW HOFF PROCEDURE

This was the first algorithm that we implemented - a regression method. The Widrow-Hoff algorithm is a Least Mean Square (LMS) algorithm. It is similar to the relaxation rule, but the key difference is that the relaxation rule is a correction rule, so $a^T y_k$!=b, and the corrections never cease. In case of Widrow-Hoff, the learning rate η, is annealed i.e. decreased wrt k (the number of iterations) for convergence to occur. Effectively, $\eta(k) = \eta(1)/k$ is the learning rate. It terminates whenever an iteration k is reached such that :

$$\eta(k)(b_k - a^T y_k)y_k < \theta$$

LMS WIDROW HOFF PROCEDURE

With cross validation, the output of the LMS procedure is the actual number of shares. But since the target variable (the actual number of shares) exhibits a high variance, this output on test data is not acceptable.

Hence we discretize the target variable to two classes: < 1400 shares (label -1) and ≥ 1400 shares (label +1) and we consider the prediction to be correct if the value and the actual result have the same sign (both + or -)

The prediction accuracy was around 53.3%

NAIVE BAYES CLASSIFIER

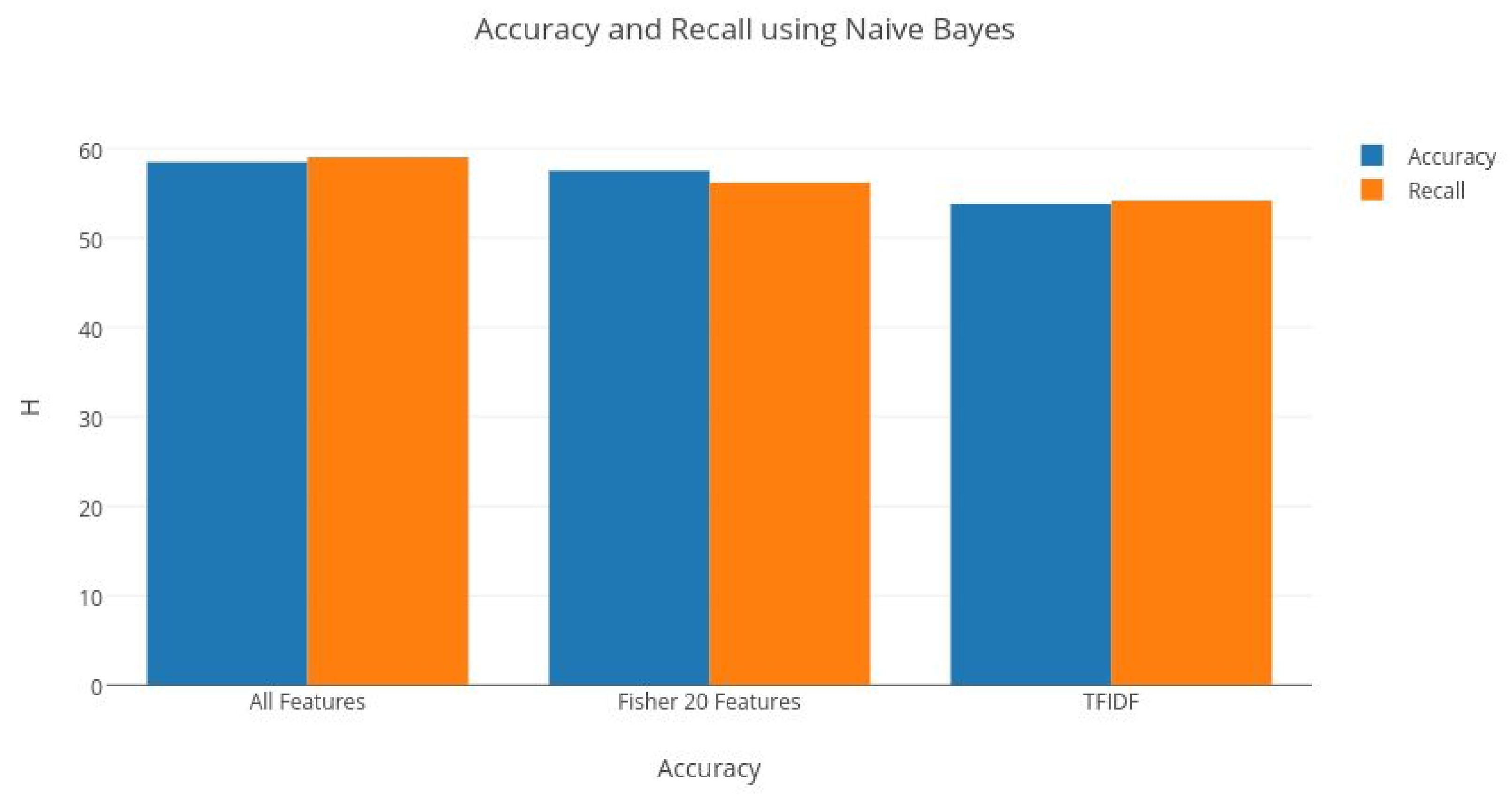
We implemented the Naive Bayes classifier for this two class problem. We used only the discrete features as the approach involving discrete features combined with fitting a Gaussian to the continuous features didn't yield satisfactory results. Even binning for the continuous features didn't yield good enough results. We used log probabilities to avoid numerical errors. We also implemented laplace smoothing to take care of samples not seen during training.

The Naive Bayes probability model along with the Maximum A Posteriori decision rule gives rise to the Naive Bayes classifier. This assigns a class label $\hat{y} = C_k$ for some k as follows:

$$\hat{Y} = \operatorname{argmax}_{k \in \{1, 2, \dots K\}} p(C_k) \prod_{i=1}^{n} p(x_i \mid C_k)$$

Since the Naive Bayes classifier is not compute intensive, we could get results on using the entire feature set (of only discrete valued features) - 58.4% accuracy and using a subset of the features(again only discrete valued features) - 57.5%

NAIVE BAYES RESULTS



RANDOM FORESTS

Random Forests use multiple decision trees which are built on separate sets of samples drawn from the dataset (with replacement). In each tree, we use a subset of all the features we have. By using more decision trees and averaging the result, the variance of the model can be greatly lowered.

The randomization comes from two sources -

i) Bootstrapping ii) Selecting a subset of the features at each node of an individual tree

The accuracy for Random Forests with 500 trees and 5 features for classification out of 20 at each node, was around 66%

RANDOM FORESTS

```
1. For $b = 1$ to $B$:
    (a) Draw a bootstrap sample $Z^*$ of size $N$ from the training data.
    (b) Grow a random-forest tree $T_b$ to the bootstrapped data, by recursively repeating the following steps
    for each terminal node of the tree, until the minimum node size $n_{min}$ is reached.
        i. Select $m$ variables at random from the $p$ variables.
        ii. Pick the best variable/split-point among the $m$.
        iii. Split the node into two daughter nodes.
2. Output the ensemble of trees ${}_1\{T_b\}^B$
```

To make a prediction at a new point x: Classification: Let $\hat{C}_b(x)$ be the probability prediction of the bth random-forest tree. Then $\hat{C}_{rf}^B(x)$ = average or majority vote ${}_1\{\hat{C}_b(x)\}^B$

RANDOM FORESTS

For Random Forests, there are three main things to be considered:

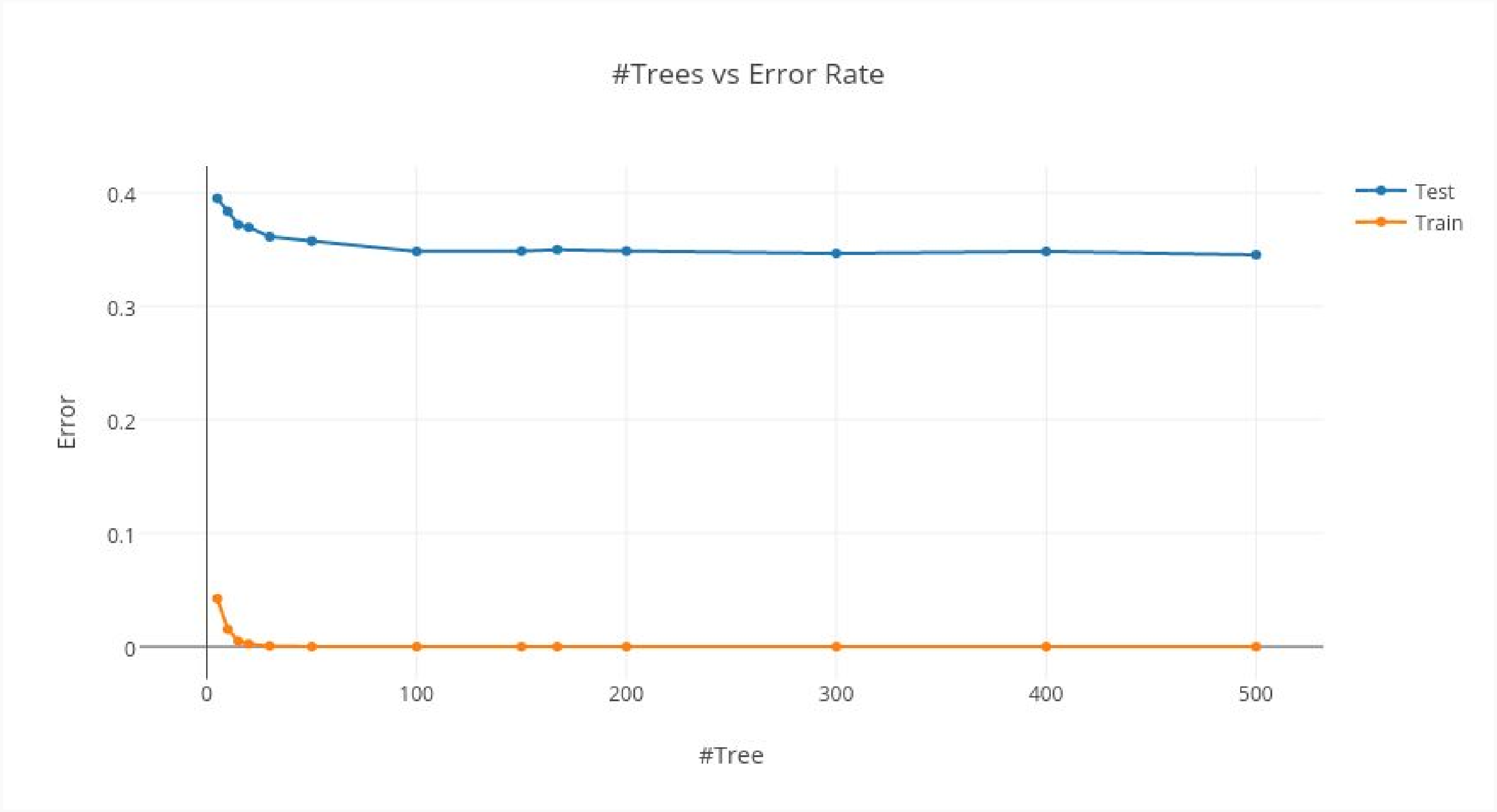
1) The number of trees in the forest
2) The number of features selected at each node - the optimal value is around √p where p is the number of features
3) The minimum number of samples at the leaf node (when to stop growing the tree) (the optimal seems to be 2 samples)

We used Gini scores to calculate the Information Gain at each node for each possible split point for each feature. The point of the feature which yielded the maximum IG is selected for splitting.

$$IG(D_p, a) = I(D_p) - \frac{N_{left}}{N} I(D_{left}) - \frac{N_{right}}{N} I(D_{right}).$$

$$I_G(t) = \sum_{i=1}^{c} p(i|t)(1 - p(i|t)) = 1 - \sum_{i=1}^{c} p(i|t)^2.$$

RANDOM FORESTS



REPTree

Horizon Effect: It is hard to tell when a tree algorithm should stop because it is impossible to tell if the addition of a single extra node will dramatically decrease error. A common strategy is to grow the tree until each node contains a small number of instances then use pruning to remove nodes that do not provide additional information.

One of the simplest forms of pruning is Reduced Error Pruning.

- Consider each node for pruning
- Pruning = removing the subtree at that node, make it a leaf and assign the most common class at that node
- A node is removed if the resulting tree performs no worse than the original on the validation set - removes coincidences and errors
- Nodes are removed iteratively choosing the node whose removal most increases the decision tree accuracy on the graph
- Pruning continues until further pruning is harmful

We used algorithmia's weka library implementation of REPTree on our dataset, and achieved an accuracy of 63.6338 %

LOGISTIC REGRESSION

We used scikit-learn's logistic regression algorithm.

The target value is expected to be a linear combination of the input variables. In mathematical notion, if Ŷ is the predicted value, $\hat{y}(w, x) = w_0 + w_1 x_1 + ... + w_p x_p$ In this model, the probabilities describing the possible outcomes of a single trial are modeled using a logistic func.

The hypothesis is:

$$h_\theta(x) = g(\theta^T x) = 1/(1 + e-^{\theta Tx})$$

and parameters are chosen so as to maximize their likelihood

$$_{i=1}\Pi^{m}\ p(y^{(i)}|x^{(i)}\ ;\ \theta)$$

The accuracy was around 64%

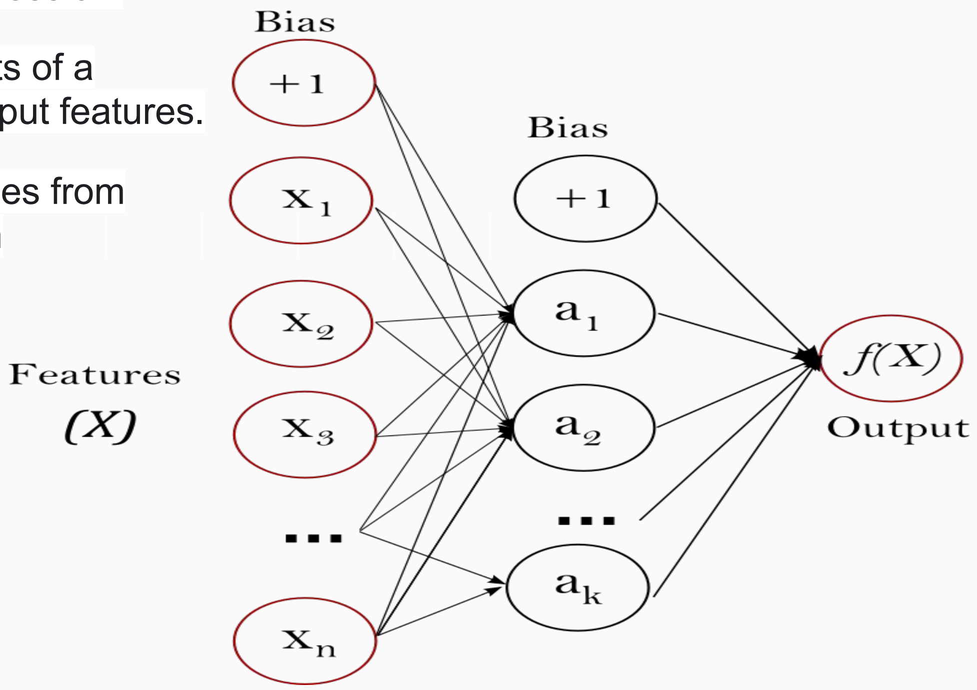
NEURAL NETWORKS

A **Multi-layer Perceptron (MLP)** is a supervised learning algorithm that learns a function $f(\cdot) : R^m \rightarrow R^o$ by training on a dataset, where m is the number of input dimensions and o is the number of output dimensions. Given a set of features $X = x_1, x_2, ..., x_m$ and a target variable y, it can learn a non-linear function approximator for either classification or regression.

The leftmost layer, known as the input layer, consists of a set of neurons $\{x_i | x_1, x_2, ..., x_m\}$representing the input features.

Each neuron in the hidden layer transforms the values from the previous layer with a weighted linear summation $w_1x_1 + w_2x_2 + ... + w_mx_m$ followed by a non-linear activation function $g(\cdot) : R \rightarrow R$

The output layer receives the values from the last hidden layer and transforms them into output values.



NEURAL NETWORKS

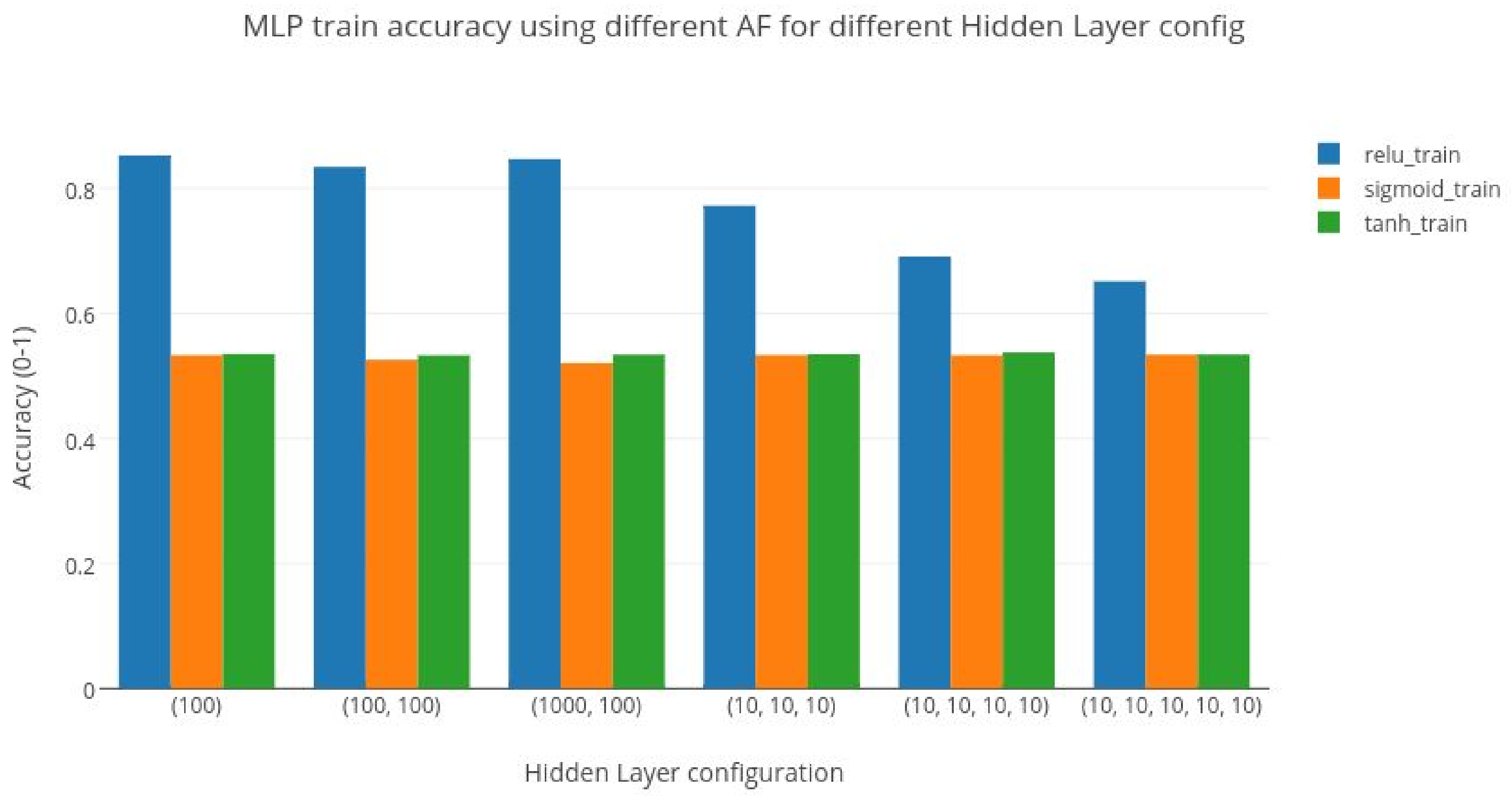
We’ve used scikit-learn for neural nets, also known as the Multi Layer Perceptron Classifier - MLPClassifier which uses a ‘relu’ (rectified linear unit function f(x) = max(0, x) activation function by default, and we’ve tried sigmoid and tanh activation function

If we used 20 features which were taken at the start, with 10 epochs and cv of 70:30, 4 hidden layers of size 10 each we got an average accuracy of about 55-57%, with across all activation functions.

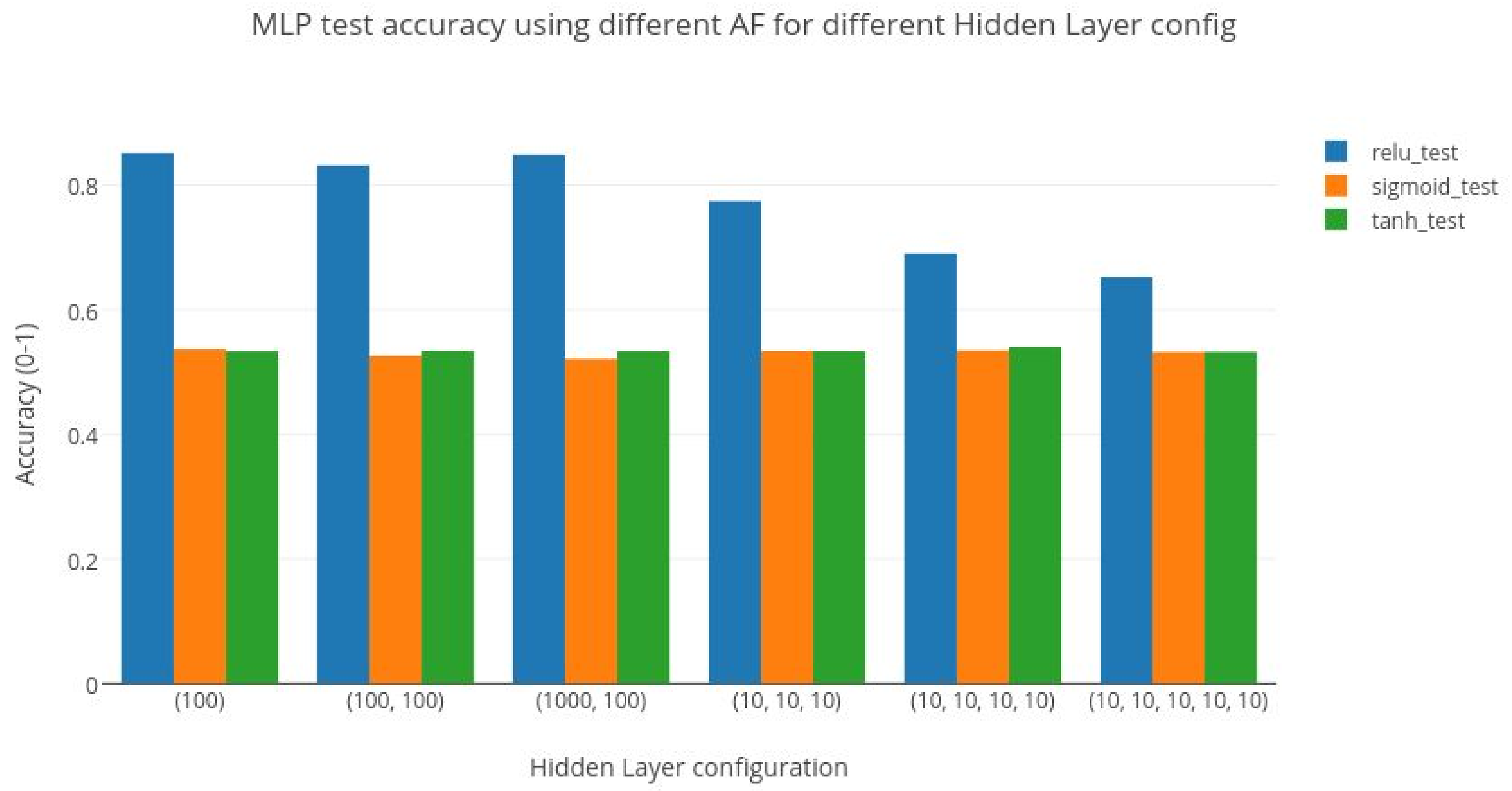
We decided to run it on **ALL** the features, and with 10 epochs and cross-validation of 70-30, and running a NN with 4 hidden layers and 10 layer size each. Using tanh and sigmoid, we achieved accuracy of ~56%, but surprisingly, for ReLU activation function, average accuracy came out to be 72%. On closer examination, we found out that either the accuracy came out to be ~53-56%, or it was ~79-89% (more than half the time), and very rarely in between. This was spread out across multiple configurations of hidden layer sizes and numbers (data on the next slides).

After varying the hidden layers and sizes, maximum accuracy in one epoch was once reported to be 92.6% at 1 hidden layer of 100 size, and it also had the maximum average accuracy. The model was re-run for 10 epochs again and the accuracies were reported to be similar.

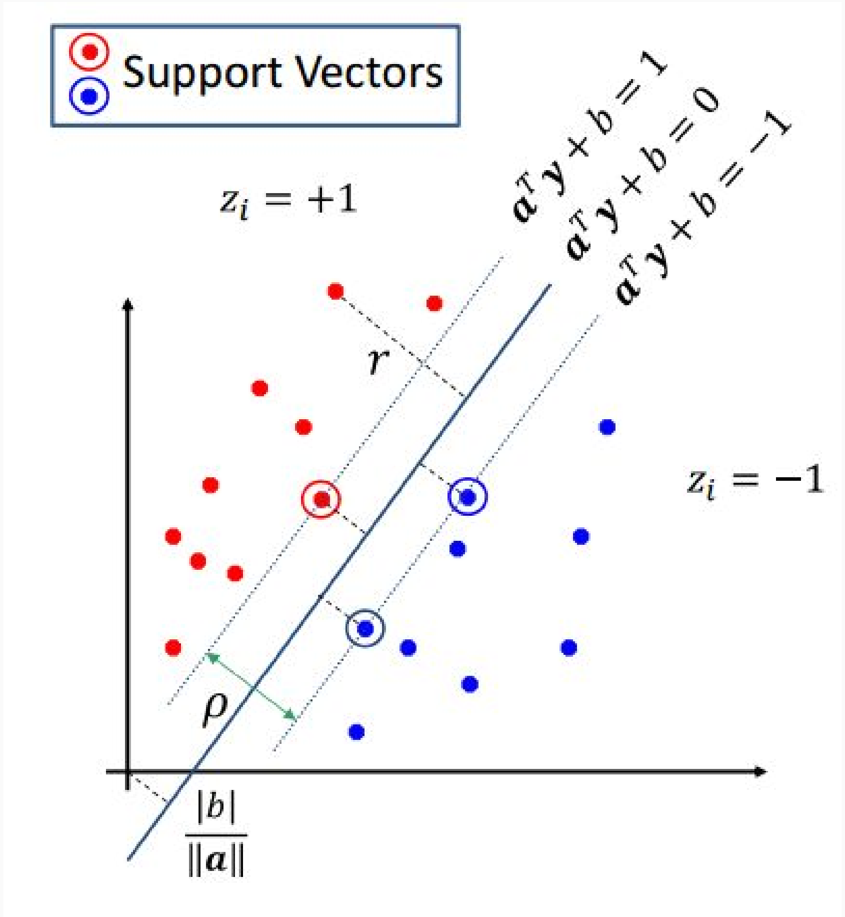
NEURAL NETWORK ACROSS CONFIGURATIONS - TRAIN



NEURAL NETWORK ACROSS CONFIGURATIONS - TEST



SVM


Support Vectors
$z_i = +1$
$z_i = -1$
$\boldsymbol{a}^T\boldsymbol{y} + b = 1$
$\boldsymbol{a}^T\boldsymbol{y} + b = 0$
$\boldsymbol{a}^T\boldsymbol{y} + b = -1$
r
ρ
$\frac{|b|}{\|\boldsymbol{a}\|}$

SVM

We used scikit-learn’s svm method with three kernels (using feature set of 20 features for each sample):

1) Linear kernel - 65.4% $\langle x, x' \rangle$
2) RBF kernel - 53.6% $\exp(-\gamma |x - x'|^2)$
3) Polynomial kernel (degree 3) - not terminating (due to large sample size) Even on taking sample sizes as small as 100, there was no convergence $(\gamma \langle x, x' \rangle + r)^d$

THE TF-IDF APPROACH

1. Extract the content of all the articles in unicode format from the given urls of each article using Python-Goose article extractor (took around 12 hours).
2. Assign labels to the articles +1 and -1 according to the number of shares.
3. Extract features from the text files - X(i, j) - Bag of Words representation - Each entry denotes the number of times the j^{th} unique word occurs in the i^{th} document. Since many of the entries are 0's, use sparse matrix notation. i ~ 37000, j ~ 123454
4. Though the number of occurrences of a word in a document is a good start, longer documents will have higher average count values than shorter documents, even though they might talk about the same topics. Correct this using Term Frequencies - tf. Another refinement on top of tf is to downscale weights for words that occur in many documents in the corpus and are therefore less informative than those that occur only in a smaller portion of the corpus. This downscaling is called tf–idf for Term Frequency times Inverse Document Frequency.
5. Use tf-idf to train a classifier and use it for predicting the labels of test data.

TABLE OF ACCURACIES

Classifier/Technique	Accuracy
Linear Regression LMS	53% (All)
Logistic Regression (Library)	64% (All)
Naive Bayes	58.4% (All), 57.5% (20 features), 53.84% (TFIDF)
Random Forest	66% (500 Trees) TFIDF - 54.58% (500 Trees)
REPTree (Library)	63.6338%
SVM (Library)	20 Features - 65.4% (Linear), 53.6% (RBF) TFIDF - 53.8% (Linear), 53.52 % (RBF)
Neural Net (MLP) (Library)	85% (All Features, ReLU), 57% (20 features, others)

CONCLUSION

We’ve reached a maximum accuracy (avg.) of 85% with MLP classifier but with some variance as well, and all others vary from 53% to 66%. Most of these accuracies are similar to what has been reported in the paper, and some haven’t been implemented in the paper (Neural Nets being one).

THANK YOU