Achieving Fine Grained Control over Incidence of False-positives and False-negatives in NLP Classification Tasks

Akshay Channesh
University of Illinois Chicago
achann4@uic.edu

Abstract

In this work we propose a classifier agnostic meta-algorithm called FGC-Classify that wraps around any existing classifier and allows you to control the incidence of False Positives and False Negatives. We evaluate this using Naive Bayes classifier and a Spam text message dataset.

1 Introduction

Many tasks in NLP are classification tasks. These include spam detection, document classification, sentiment analysis and others. Achieving a 100% accuracy on these tasks isn't always possible and moreover not necessary in most cases. In instances where the classifier wrongly classifies something, two kinds of errors occur—the False Positive and the False Negative. These kinds of misclassifications are common in any statistical task. However in the real world we encounter many instances where one of these errors may turn out to be grievous. For example many personal assistants use Trigger Word Detection to switch themselves on and record a command. If the device falsely starts recording even when a trigger word hasn't been uttered, that would be detrimental to both, the privacy of the users, and the reputation of the company producing the device. The error in this case would be a False Positive error. Detecting a trigger word when there wasn't any in reality.

In such cases no matter how good the accuracy is, the classifier isn't all that impressive if there are even a few false-positives. Therefore it becomes essential to strive towards reducing FPs even at the cost of others attributes like accuracy. Similarly in many other settings, FPs and FNs become more important than accuracy.

Even though significant advances have happened towards increasing the accuracy of these classifiers, there aren't many efforts towards addressing the problem of False Negatives (FN) and False Positives (FP). Existing classifiers are somewhat rigid and do not allow for dynamic control of FPs and FNs.

This work proposes a classifier agnostic meta-algorithm called FGC-Classify that wraps around any existing classifier and gives fine grained control over the occurrences of FPs and FNs. It has been inspired by the Game Theoretic methods of Equilibrium Semantics found in Parikh (2010) and Parikh (2019).

2 Related Work and Preliminary Concepts

Equilibrium Semantics was introduced and refined by Prashant Parikh (Parikh, 2010). It uses Game Theoretic methods to define frameworks to capture the semantic relationship of languages and the world in which that language is used. In Parikh (2019) the author mentions in passing that classification can be seen as a game, and gives the example of spam classification. However the author does not describe in detail as to how this can be done. We have taken this idea as our inspiration and based our work on it.

Although our work has got little to do with Semantics, we have been inspired by the methods employed in it. Our work focus on classification tasks in general and employs a few rudimentary concepts of game theory to achieve its goals.

2.1 Game Theory

Game theory is the study of rational and intelligent agents that interact with each other. The setting they interact in is called a Game. Narahari (2014) gives the following definitions which are relevant to us.

Definition 1 (Information Set) *An information set of a player is a set of that player's decision nodes that are indistinguishable to her.*

Definition 2 (Extensive Form Game) *An extensive form game* Γ *consists of a tuple* $\Gamma = \langle N, (A)_{i \in N}, H, P, (I_i)_{i \in N}, (u_i)_{i \in N} \rangle$ *where*

- $N = \{1, 2, \ldots, n\}$ *is a finite set of players*
- $(A)_{i \in N}$ *is the set of actions available to player* i
- *H is the set of all terminal histories where a terminal history is a path of actions from the root to a terminal node such that it is not a proper subhistory of any other terminal history. Denoted by* S_H *the set of all proper subhistories (including the empty history* ϵ*) of all terminal histories.*
- $P : S_H \rightarrow N$ *is player function that associates each proper subhistory to a certain player*
- $(I_i)_{i \in N}$ *is the set of all information sets of player* i
- $u_i : H \rightarrow R \; \forall i$ *gives the utility of player* i *corresponding to each terminal history.*

2.2 Utility Theory

Utility Theory was introduced by von Neumann et al. (1944) and describes in extensive details how real valued utilities can be used to capture the ordering of the preferences of each agent over the outcomes of the game. Every player tries to maximize her payoffs. Payoffs are calculated as a weighted sum of utilities weighted with the probabilities of occurrence of each outcome. Although in games utilities are somewhat predefined, in our case they act as tunable hyper-parameters.

2.3 The Classification Problem

The classification problem is foundational to Machine Learning and keeps coming up every so often. Many tasks in NLP are classification tasks. Blum et al. (2020) say that a core problem underlying many machine learning applications is learning a good classification rule from labeled data. They define the problem as one that consists of a domain of interest X called the instance space, such as a set of SMS texts, and a classification task, such as classifying texts into spam versus non-spam.

We will typically assume our instance space $X = \{0, 1\}^d$ or $X = R^d$, corresponding to data that is described by d Boolean or real-valued features. Features for SMS texts could be the presence or absence of various types of word or phrases.

To perform the learning task, our learning algorithm is given a set S of labeled training examples, which are points in X along with their correct classification.

The algorithm then aims to use the training examples to produce a classification rule that will perform well over new data, i.e., new points in X. A key feature of machine learning, which distinguishes it from other algorithmic tasks, is that our goal is generalization: to use one set of data in order to perform well on new data we have not seen yet.

3 Proposed Framework

Our main contribution is the introduction of the meta algorithm FGC-Classify. This is shown in Algorithm 1. This runs a game over any existing classifier and uses a user defined utility vector to weight the final classified labels. This allows for a fine grained control over final labels. By selecting the utility vector correctly one may control the incidence of False Positive and False Negatives.

We do this by running a game over the underlying classifier. The classifier and the wrapper form the two players of the game. The wrapper FGC-Classify has a different set of preferences than the classifier underneath. While the classifier is trying to maximize accuracy, FGC-Classify is doing something else depending on the utilities specified by the user. Typically these utilities are specified with the goal of either reducing FPs or FNs. FGC-Classify takes the output of the classifier—the probabilities of labels—and then picks the final label by weighting the probabilities with utilities.

Algorithm 1: FGC-Classify

Data: $\mathcal{D}$ (test Data), $[u_i]$ (utility vector), $classify(\cdot)$ (classification algorithm)
Result: Classified lables
for *every item* x *in test data* $\mathcal{D}$ **do**
  $p_i \leftarrow classify(x)$ `// generate label probabilities`
  final label $= \text{argmax}\, u_i \cdot p_i$ `// pick the lable with the highest payoff`
end

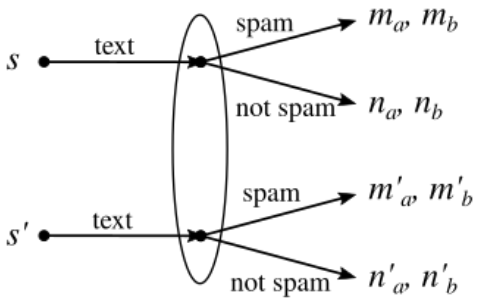

Figure 1: The Classification Game

Figure 1 shows the extensive form representation of the game being run. Here an underlying classifier returns the probabilities of each of the labels. FGC-Classify then weights the utilities on these probabilities and picks the best final label. The utilities (m_a, n_a, m'_a, n'_a) form the hyper parameter vector that is of interest. The vector (m_b, n_b, m'_b, n'_b) is ignored for now since it forms the utilities of the other player—the underlying classifier in our case. This is of use only in the case where we're using interactive classifiers. For our setting we will stick with the simpler case where we work with only one vector.

3.1 Hyper-parameters

Since utilities are user defined they form the hyper-parameters of our algorithm. These need to be tuned by the user. Since these are von Neumann-Morgenstern utilities, one may use all the results found in Utility theory in tuning these. Utilities in this case are very forgiving to small perturbations and more often than not one finds that they've picked the right numbers by following their intuition.

4 Experiments and Discussion

We've chosen a simple classification task to evaluate this framework. We stick with the well known Spam Classification problem. Since the classification itself is of little interest to us and since our focus lies in the control of false-positives(FP) and false-negatives(FN), a simple classification task which provides enough FPs and FNs is suitable for our evaluation. Our work is available as a Jupyter Notebook on Github.[1]

[1] https://github.com/AkshayChn/cs521/blob/main/Classifier.ipynb

4.1 Dataset

Almeida et al. (2011) have produced an excellent dataset for this task, and over the years this dataset has become quite famous and is the go-to dataset for demonstrating many new ideas in classification.

The dataset is of SMS text messages which are labeled as either 'spam' or 'ham' (not spam). This dataset contains 5572 text messages of which about 86.6% are ham texts and 16.4% are spam texts. It is interesting to note here that even a really unintelligent classifier which always classifies all texts as 'ham' will achieve 86.6% accuracy. And thus anything less than this is of no interest to us.

4.2 Baselines

Our meta algorithm uses an existing classifier. And thus we need a simple classifier example to illustrate our algorithm. We've chosen the tutorial by KDnuggets[2] titled *Spam Filter in Python: Naive Bayes from Scratch* as our example.

Since the given classification task is quite simple even the simple Naive Bayes Classifier does a remarkably good job. It achieves over 99% accuracy even with small amount of data. Thus to artificially get lower accuracy so as to get enough FPs and FNs for analysis we use just 3% of the entire dataset for training, and the rest for testing. Thus in the dataset we have 167 rows for training and 5405 rows for testing. Even so the classifier achieves an accuracy of 95% with 5170 correct labels and 235 errors. Of these errors we get 207 FNs (when a true spam is falsly allowed as ham) and 27 FPs (when a true ham is falsly flagged as spam).

4.3 Implementation Detail

We first randomize the data and split it into training and testing sets. We the clean the data by removing all punctuation and changing all capital letters to small letters. We count all the unique words in the training set and create a vocabulary set. We train the Naive Bayes classifier using this vocabulary and use Laplace smoothing with $\alpha = 1$.

Later we use this trained classifier inside the FGC-Classify algorithm to control the occurrence of FPs and FNs. The utility vector

[2] https://www.kdnuggets.com/2020/07/spam-filter-python-naive-bayes-scratch.html

(m_a, n_a, m'_a, n'_a) acts as hyper-parameter vector, varying which we can finely control FPs and FNs.

4.4 Main Results

Our main result is split into two cases. One where we reduce False Positives in small steps, and the other were we reduce False Negatives similarly.

4.4.1 Controlling False Positives

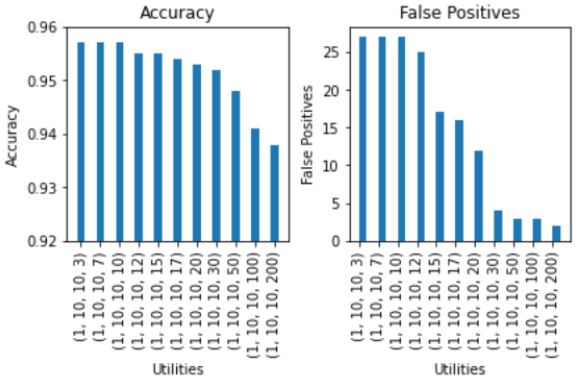

Figure 2: Reducing False Positives

Figure 2 shows a graph where as we go on increasing n'_a with respect to other utilities, we observe that FPs go on reducing. Accuracy also falls. This shows that we can trade accuracy for a lowered incidence of FPs. We start with the utility vector (m_a, n_a, m'_a, n'_a) set as $(1, 10, 10, 3)$ and sequentially increase n'_a until $(1, 10, 10, 200)$. We see a dramatic reduction of FPs. This is because the branch associated with n'_a is getting progressively weighted higher and is being preferred by the algorithm to pick the final label.

4.4.2 Controlling False Negatives

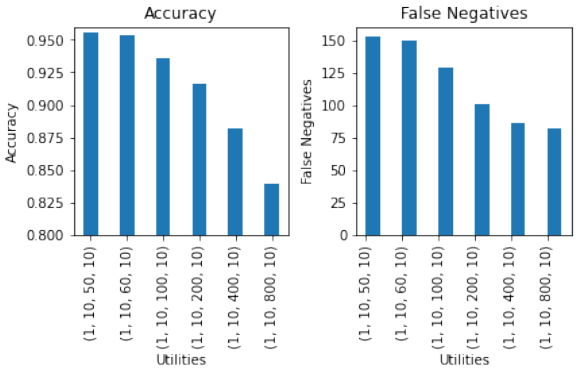

Figure 3: Reducing False Negatives

Just as we did with FPs, we now set out to reduce FNs. Observe Figure 3, here we increase m'_a with respect to other utilities. We start with a utility vector of $(1, 10, 50, 10)$ and then go until $(1, 10, 800, 10)$. We see that FNs reduce by about half. However in this case the accuracy falls by quite a lot.

4.4.3 Discussion of Results

These results demonstrate the efficacy of the framework. We can indeed control FPs and FNs as desired while trading off on accuracy. This framework also allows us to do so in a very fine grained manner. Utilities are real numbers and can be defined to arbitrary precision. And since our control relies on setting these utilities, we get fine control over FNs and FPs.

One may also notice that in our case since there are an unequal number of Spam and Ham messages to begin with, it is harder to control FNs than FPs.

5 Conclusion

In this work we demonstrated a method to achieve fine grained control over the incidence of False Positives and False Negatives. We proposed a classifier agnostic meta-algorithm called FGC-Classify that wraps around any existing classifier. We showed that by tuning the hyper-parameters one can reduce FPs and FNs. We evaluated this method on an SMS text dataset using Naive Bayes classifier and showed that we can reduce of FPs and FNs with fine grained control. We also discussed how the hyper-parameters can be tuned and pointed out that the extensive and beautiful theory of utility functions can be used for this tuning.

In this work we have looked at binomial classification and in the future one may look at multi-way classifications. There is also scope to look at interactive classifiers whereby you can take full advantage framework by defining the whole game. One can also extend this to classification settings outside of NLP such as image processing and data analysis.

References

Tiago A. Almeida, José María Gómez Hidalgo, and Akebo Yamakami. 2011. Contributions to the study of sms spam filtering: new collection and results. In *DocEng '11*.

Avrim Blum, John Hopcroft, and Ravindran Kannan. 2020. *Foundations of Data Science*. Cambridge University Press, Cambridge.

Y. Narahari. 2014. *Game Theory and Mechanism Design*. World Scientific Publishing Company Pte. Limited.

Prashant Parikh. 2010. *Language and Equilibrium*. MIT Press.

Prashant Parikh. 2019. *Communication and content*. Number 4 in Topics at the Grammar-Discourse Interface. Language Science Press, Berlin.

John von Neumann, Oskar Morgenstern, and Ariel Rubinstein. 1944. *Theory of Games and Economic Behavior (60th Anniversary Commemorative Edition)*. Princeton University Press.