# Observe AI Task

**Sairam bade**

**Approach::**

Data given is transcribed by a speech recognizer so it is very noisy. So the proposed approach does preprocessing and applies different machine learning algorithms. Data is very imbalanced with very few training examples for some classes so that is also taken into consideration while training a classifier.

**Preprocessing::**

The data given doesn't have any non alphabetic characters other than [‘] which are decontracted to normal words
E.g. i’m → i am and doesn’t → does not

**Stopping::**

1. This involves removing all the category neutral words like (is, the,a etc) which doesn’t have much effect on classification.
2. Output text from a speech recognizer has a lot of article insertions and deletions , so it is better to remove these stopwords.

**Stemming::**

1. Stemming is the conflation of the morphological variants of the same word(e.g. Connection,connecting ,connected into common stem(connect).
2. Different inflected forms of a certain word do not carry category dependent information .
3. This also reduces the number of words.
4. But root word is not generally an actual word (else → els) it is preffered to do lemmatization.

The experiments prove these assumptions.

**Feature Selection:**:

**BOW features(counts) ::**

1. Just use the counts of each word , non relevant words which are present more are taken as an important feature.
2. These are sparse.

**BOW features(TF-IDF) ::**

1. TF-IDF score relatives the relative importance of a term in the text and the entire training set
2. Here we can use bigram as tokens along with unigrams but one quick look at the data suggests this is not a better idea as outcome of the classification doesn't depend on combination of words.

**Models::**

There is a lot of correlation of most prominent words in training and dev dataset. Top 5 words for each class in train and dev sets.

**Training set**

[('okay', 19861), ('one', 9021), ('yeah', 8138), ('thank', 8000), ('alright', 7778), ('like', 6228), ('order', 5582), ('know', 5255), ('right', 5108), ('see', 4467)]
[('okay', 6197), ('thank', 3329), ('one', 3007), ('alright', 2452), ('yeah', 2320), ('account', 1782), ('like', 1611), ('right', 1555), ('know', 1544), ('phone', 1297)]
[('okay', 1624), ('one', 757), ('yeah', 669), ('know', 650), ('thank', 602), ('like', 590), ('alright', 544), ('right', 464), ('order', 400), ('na', 374)]
[('okay', 316), ('yeah', 155), ('thank', 145), ('one', 140), ('alright', 132), ('like', 117), ('go', 100), ('know', 99), ('na', 88), ('right', 82)]
[('okay', 24), ('thank', 14), ('phone', 11), ('number', 10), ('yeah', 10), ('alright', 9), ('one', 8), ('yes', 7), ('na', 7), ('email', 7)]

**dev set**

[('okay', 5826), ('one', 2672), ('thank', 2608), ('yeah', 2475), ('alright', 2472), ('order', 1773), ('like', 1655), ('right', 1550), ('know', 1510), ('see', 1359)]
[('okay', 1973), ('thank', 1104), ('one', 905), ('yeah', 785), ('alright', 782), ('account', 591), ('know', 574), ('like', 516), ('right', 485), ('na', 470)]
[('okay', 414), ('thank', 214), ('alright', 182), ('one', 167), ('yeah', 163), ('right', 95), ('order', 95), ('like', 91), ('know', 78), ('na', 77)]
[('okay', 94), ('thank', 58), ('one', 46), ('know', 41), ('alright', 39), ('yeah', 38), ('right', 25), ('name', 23), ('na', 22), ('yes', 22)]
[('thank', 12), ('okay', 12), ('alright', 10), ('name', 7), ('one', 7), ('number', 7), ('yes', 6), ('right', 5), ('yeah', 5), ('three', 5)]

So first I applied Naive Bayes approach with tf-idf features

**NB approach(alpha =1)**

Accuracy on dev set is .7064935064935065 which is good for a naive count based approach.

**Logistic regression without regularization:**

**Tf-idf features::**

Accuracy :: 0.8389610389610389

As the number of training features are large I applied L1 regularization and also performed Grid Search to obtain the regularization parameters.

**Logistic regression with regularization:**

Here I did many experiments to check my prior assumptions.
without stopword removal :: 0.8346320346320346
without stemming :: 0.8380952380952381
Only Unigram :: **0.8432900432900433**
only Bigram :: 0.7515151515151515
Uni + Bigram Accuracy :: 0.8251082251082251

Using stemmed unigram has the better accuracy.

These show that our assumption of not using bigrams and using stem words is a better idea.

**XGB:**

Gradient boosting is the goto classifier for many machine learning tasks.
Accuracy obtained after fine tuning is 0.8476190476190476

**SVM:**

This is the best model that is obtained using tf-idf counts . After fine tuning
Using linear kernel gives 0.8493506493506493
And non linear kernel gave **0.8554112554112554**

**This is the best accuracy I achieved using tf-idf counts .**

**Word2Vec ::**

Bag of words has the disadvantage of data being sparse and unknown words are all assumed to be the same.

So I used google pre trained word2vec which contains 3 Billion words to obtain word embedding of size 300.

Loading word2vec for all the words takes a lot of time and space(3.5 GB which will be loaded in RAM).

**Doc2vec ::**

I used the sum of all word vectors as a document vector.There is proof that this naive doc2vec is good at conveying the information present in the sentence.
To achieve this i precomputed the doc2vecs and pickled them to be used later.

Here I used both actual words and lemmatized words but using lemmatized words seems to be helping.(maybe different forms of the stem word has not that much impact on classification).

Accuracies obtained are for

Lemmatized words and SVM :: 0.8268398268398268
Full words and SVM :: 0.8233766233766234
Full words and logistic regression with L1 regularization ::0.8216

**Doc2vec + tf-idf params + XGB(Majority Voting) ::**

Using an ensemble is used to reduce the error done by each individual model.
Accuracy :: 0.8554112554112554

**Imbalanced Data ::**

There is a big problem that is not looked upon while calculating only accuracy until now . Standard classifier algorithms like Decision Tree and Logistic Regression have a bias towards classes which have number of instances.

We are stuck in the **"accuracy paradox".** Where in accuracy seems to be high but f1 score would be lower as classes with smaller training data were not classified correctly.

Train :: [('order', 2293), ('customer', 914), ('shopper', 201), ('applicant', 45), ('misc', 7)]
Dev :: [('order', 765), ('customer', 305), ('shopper', 67), ('applicant', 15), ('misc', 3)]

As can be seen here shopper,applicant and misc classes are underrepresented and all our classifiers are not able to predict these classes.

**Bagging::**

Used Random forests for bagging. Here samples are chosen from the population at random with replacement. This overcomes overfitting and is better when dealing with noisy data. Penalizing with weights is also not helping

Accuracy :: 0.767965367965368

**Penalized-SVM::**

To avoid imbalance problem I used penalized-svm with different weights given to different classes. These weights are used in misclassified errors.
Accuracy dropped to 0.7948051948051948 but class wise accuracies improved using doc2vec and svm.

Using tf-idf frequencies and svm there is no improvement with penalized-svm

Results are provided with this on eval set.

**Conclusion & Improvements that can be pursued ::**

At the end of the day, some machine learning projects succeed and some fail. What makes the difference? Easily the most important factor is the features used…This is typically where most of the effort in a machine learning project goes (Pedros Domingos “A few useful things to know about machine learning”)

Using both wordvec and tf-idf as features might provide a better result.
Synthetic minority oversampling technique(SMOTE) can be used to create more training samples for classes with less representation in training set.