Predicting Depression from Social Media Updates

Sharon Babu

June 2018

Advisor: Professor Michael J. Carey

Donald Bren School of Information and Computer Sciences

Abstract

Depression is a serious problem around the world, and while there has been much research on the topic, the stigma surrounding mental illness still causes missed diagnoses. This study seeks to predict if a user is at risk for depression using their combined Facebook status updates as the predictors; all data is taken from myPersonality. I used the Big Data Management System AsterixDB to store and query my data, and ran algorithms such as Logistic Regression, Support Vector Machines, and Random Forests to solve this classification problem. While one major finding is that more data is needed, we are able to see an improvement from our baseline accuracy using a model that employs both Regression and a text analysis feature representation called Term Frequency-Inverse Document Frequency (TF-IDF).

Introduction

According to the National Institute of Mental Health, in America alone, approximately 16.2 million adults suffered from at least one major depressive episode in 2012. This represents 6.7% of all US adults. 37% of these did not receive any treatment (NIMH). One reason for this is that factors such as stigma surrounding mental health can lead to those struggling with depression not being diagnosed. As social media plays a huge role in society today, we begin to wonder how we can apply this field to the problem of underdiagnosis in depression.

To solve this problem, text analysis and supervised machine learning can be applied in an attempt to predict if a Facebook user is at risk for depression based off their Facebook status updates. Supervised machine learning is a type of machine learning that predicts a known target variable from a set of features. In this case, the target variable is if the user is depressed or not, and the features are the individual words that make up the status posts made by all users (a common technique in text analysis).

Models are then trained, and the accuracy of the model is determined by seeing how often the model can accurately classify users as at risk for depression or not. As the target variable is a discrete class, this is a classification problem (as opposed to a regression problem where the

predicted variable is a real value on a continuous scale). Machine learning models that will be used in this problem are Logistic Regression, Random Forests, and Support Vector Machines.

Related Work

While using social media to study psychological phenomena is relatively new, there has been much work done in this field over the past few years, as "Natural Language Processing in Mental Health Applications using Non-Clinical Texts" describes (Calvo, et al.). This review focuses on data sources and techniques used in labelling and diagnosing individuals, as well as ways to use these results to personalize mental health interventions. Major social media data sources include Twitter and Facebook, and studies have included manual analysis and the tagging of posts that indicate depression in addition to automated labeling. One successful attempt used Naïve Bayes to detect emotions in Facebook posts.

More specifically, using the myPersonality Project databases, which is also used and discussed later on for this project, many researchers were able to do studies on topics such as predicting subjective well-being using Facebook status updates (Liu, et al.), predicting personality using various features of Facebook data (Markovikj, et al.), and even using Facebook information to help identify users with self-destructive behaviors such as suicidal ideation, self-harm, and extreme drug abuse (Inkster, et al.).

Data

Data Source and Description

All data used to solve this problem come from an organization called myPersonality that was founded by Michal Kosinski and David Stillwell. From 2009-2012, the organization used a Facebook application that received voluntary access to Facebook data, such as status updates from users. In addition, users could use the app to take psychological tests, such as the BIG5

Personality test, and release those results as well. myPersonality curated these results into many datasets; each user is given an anonymized unique user ID, so that user test scores and social media information can be correlated.

The two datasets used in this project include a Facebook status updates dataset, as well as a CES-D Depression Inventory dataset.

The CES-D Depression Inventory is a standardized tool used to diagnose and rate depression. Figure 1 shows a list of the twenty questions asked in this inventory. For example, the first question states "I was bothered by things that usually don't bother me." The user would respond with a score between 0 and 3, with 0 being rarely or none of the time, and 3 being all of the time. The scores will be added (some questions are reverse scored) for a total score ranging from 0 to 60, where 60 is the most depressed. The National Institute of Mental Health uses a cutoff score of 16 to signify that an individual is at risk for depression. myPersonality's data does not include the total score, but does have all the itemized responses for users who have taken this test. Using this data, a total score can be calculated, and each user can be given a rating of 0 for "not depressed," or a 1 for "is depressed"—a 0 if the total depression score is less than 16, and 1 otherwise.

Center for Epidemiologic Studies Depression Scale (CES-D), NIMH

Below is a list of the ways you might have felt or behaved. Please tell me how often you have felt this way during the **past week**. Circle **one** number on each line.

	During the Past Week			
	Rarely or none of the time (less than 1 day)	Some or a little of the time (1–2 days)	Occasionally or a moderate amount of time (3–4 days)	All of the time (5–7 days)
1. I was bothered by things that usually don't bother me	0	1	2	3
2. I did not feel like eating; my appetite was poor	0	1	2	3
3. I felt that I could not shake off the blues even with help from my family or friends	0	1	2	3
4. I felt I was just as good as other people	0	1	2	3
5. I had trouble keeping my mind on what I was doing	0	1	2	3
6. I felt depressed	0	1	2	3
7. I felt that everything I did was an effort	0	1	2	3
8. I felt hopeful about the future	0	1	2	3
9. I thought my life had been a failure	0	1	2	3
10. I felt fearful	0	1	2	3
11. My sleep was restless	0	1	2	3
12. I was happy	0	1	2	3
13. I talked less than usual	0	1	2	3
14. I felt lonely	0	1	2	3
15. People were unfriendly	0	1	2	3
16. I enjoyed life	0	1	2	3
17. I had crying spells	0	1	2	3
18. I felt sad	0	1	2	3
19. I felt that people dislike me	0	1	2	3
20. I could not get "going"	0	1	2	3

Figure 1: CES-D Depression Inventory

The Facebook Status Updates dataset includes status posts made by users, where each record represents an individual status post by a user, similar to what is shown in Figure 2—so a user will have multiple records depending on the number of Facebook posts they have made.

UserId	Status Message	Date Posted
User1	Today was a good day.	5/10/2010
User1	Got a dog today!	6/2/2010
User2	First day of school…	9/2/2010
…	…	…

Figure 2: Facebook Status Updates Dataset in myPersonality

In this supervised machine learning problem, the depression classification is the target variable we are trying to predict, and the combined status messages for each user make up the features we will use.

Data Summary Statistics

As mentioned, the two datasets must be joined using the unique user ID provided by myPersonality. This means that only data where both information for users' Facebook Status Updates and their CES-D Depression Inventory results are present can be used. Figure 3 shows the number of unique users and status updates in each individual dataset and for the intersection of the two datasets. That intersection is the total number of users and status updates that we can use to train and test our models.

	# Unique Users	# Status Updates
statusUpdates	153727	22043394
depressionUsers	5946	--
statusUpdates ∩ depressionUsers	1055	197230

Figure 3: Summary Statistics of Data

myPersonality has curated over 22 million status updates for over 150 thousand users. Unfortunately, their dataset with the depression inventory results is much smaller—less than 6000 users are represented. When the intersection of the two datasets is taken, approximately 200 thousand status updates from about 1000 users are left. This means there is only about 1000 samples for training and testing purposes, which is an extremely small amount for machine learning; we will see later how this affects the findings and models.

Figure 4 shows a histogram of the distribution of the total depression scores among the 1000 users. The distribution is a standard Gaussian, but what is important to note here is the red line at x=16, which signifies the cutoff for being at risk for depression. Since the cutoff is half of the average, it makes sense that most of the users represented are at risk for depression – about 75% of the sample. This leads to a large imbalance between non-depressed users and depressed users, which will also affect the results.

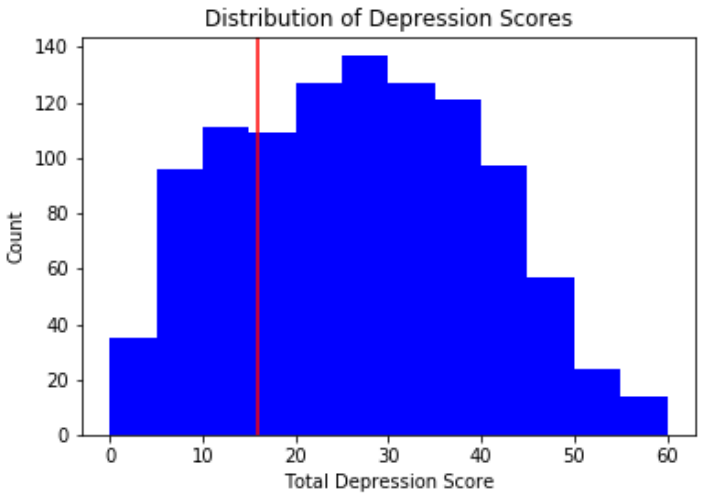


Figure 4: Distribution of Depression Scores

Technical Approach

A flowchart for the technical approach to this problem is shown in Figure 5. To begin, the two datasets, the Facebook Status Updates Dataset and the CES-D Depression Inventory Dataset, will be imported into AsterixDB, a new and upcoming Big Data Management System. Using

AsterixDB, two datasets can be created that have only the necessary information from each of the myPersonality files (both datasets contains information irrelevant to this problem). The depression dataset will have the attributes of the unique user ID and the twenty question responses to the CES-D. The Facebook Status Updates dataset will have the unique user ID and the text of a status update that the user has posted. Statuses posted by users who have not taken the CES-D Depression Inventory will be filtered out.

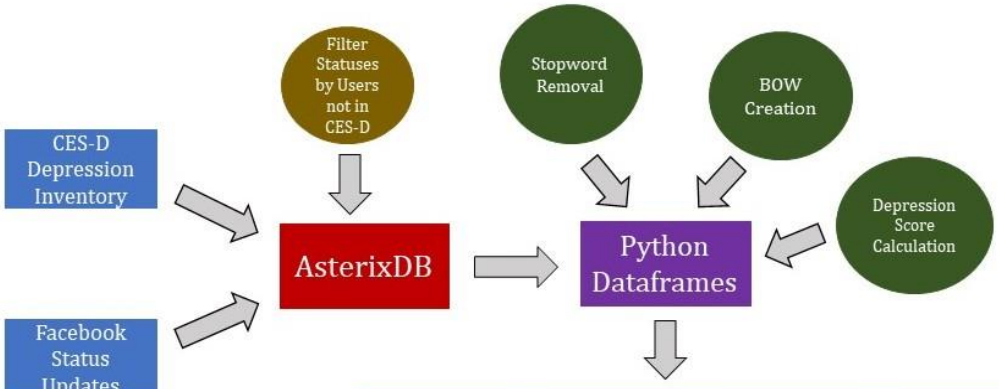


Figure 5: Technical Approach Diagram

From here, the datasets can be moved into the Python Dataframes, PANDAS. While doing this, the Facebook Status Updates dataset will be normalized in a way so that each user is represented only once, no matter the number of status updates they have. Each user will now be associated with a combination of all their status updates, simply separated by a space; the depression dataset will remain the same. The data is moved into PANDAS so that Python libraries can be used to preprocess the data and run machine learning algorithms on them.

The first step in PANDAS is to calculate a total depression score for each user. All scores for the twenty questions are added up, reverse scoring four of them. This provides the target variable for each user. Next, the status updates need to be prepared into a feature set that can be

used for training. A common feature representation in text analysis is a Bag-of-Words representation. Bag-of-Words is a representation that uses all possible words as individual features, and their values are the counts of that word in the "document" (in this case, the document is the combination of all the status updates for one user). For example, in Figure 5, User1 has used the word "happy" twice, and "movie" once across all their posts – these will be their feature values for those two words. These values will be normalized so that the sum of each user's feature values is 1 – this is to account for the fact that users have very different frequencies in creating status updates, so some users have many more words in their "document". Stopword removal will also be accounted for so that common words such as "the," "a," "I," and other words that we believe to have no impact on the problem will be discarded.

Now that the feature values and target values are prepared, the machine learning models can be trained and tested. As previously mentioned, this study will primarily focus on Logistic Regression, Support Vector Machines, and Random Forests. The models will be evaluated using the percentage accuracy of the classifiers. To get this accuracy, a train-test split where we train the model on 90% of the data and test on the remaining 10% will be used. Since the sample size is so small, a method called cross-validation will also be used. Cross-validation is a method that takes different folds of train and test data, obtains accuracies on all the folds, and then averages them together—this way, all data is being utilized for training purposes, instead of leaving 10% out.

In addition to testing the classifiers with the standard method of accuracy, tools such as precision-recall and the probability of classifications among extremely depressed individuals can be used—both these methods can show how the classifiers can be used in slightly different manners and will be discussed more in the next section.

Experimentation and Findings

Initial Results

The initial results using Logistic Regression, Support Vector Machines, and Random Forests can be seen in Figure 6. These models were created with the default settings and nothing done to the data other than normalization of counts and removing common English stopwords (stopwords taken from one of the Python libraries, NLTK).

	Training Accuracy	Test Accuracy (10%)	Cross-Validation (k=3)
Baseline	74.4%	72.6%	74.2%
Logistic Regression	74.5%	72.6%	74.2%
Support Vector Machines	74.4%	72.6%	74.2%
Random Forests	97.5%	74.5%	71.6%

Figure 6: Initial Results

To determine the baseline accuracy, a common classification trick where we predict the most common class for every case is used. In this problem, depressed is the most common class, so everyone is predicted to be depressed. Since about 75% of the sample is classified as at risk for depression, the baseline accuracy is about 75%.

Looking at the results, none of the models are performing very well. They are all performing at about the baseline accuracy of 75% with no improvement, and the Random Forests model has a severe overfitting problem of over 20% (this can be seen since the training accuracy is so much higher than the test accuracy; overfitting is the phenomenon in which the model is focusing too much on the training samples and is unable to generalize to new data not included in the training phase).

To get a better look at how the models are classifying the samples, a confusion matrix can be viewed. In binary classification problems, a confusion matrix shows the rates for True

Positives, True Negatives, False Positives, and False Negatives. Figure 7 shows the confusion matrix for the test data of our Random Forests model.

The top left quadrant shows the True Negative rate and the bottom right quadrant shows the True Positive rate. In a perfect model, both these quadrants would have ratios of 1—correctly classifying all depressed individuals as depressed and correctly classifying all non-depressed individuals as not depressed—while the other quadrants would both be 0s. In this case, it is seen that the model is simply predicting the majority of users as depressed; this leads to a high True Positive rate, but also a high False Positive rate as well. Logistic Regression and Support Vector Machines are also predicting virtually all users as depressed as well, so these models are useless.

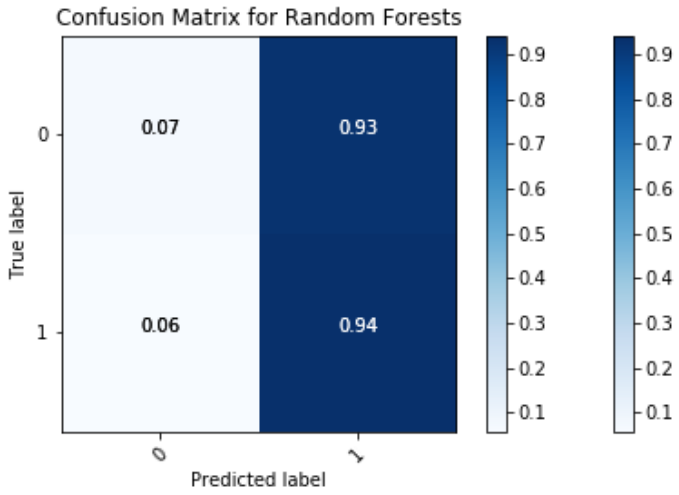


Figure 7: Initial Confusion Matrix for Random Forests

Increasing Sample Size

One of the reasons that the models are performing so poorly has to do with the skew between the amount of non-depressed users and depressed users. Not only is the skew large, but the sample size is small as well. So, before continuing to train new models, it needs to be seen how to increase the sample size.

A possible solution is to simulate new depression data for users who have status updates but did not take the depression inventory. myPersonality has many other datasets for psychological tests, including a Satisfaction with Life Survey (SWL). Intuitively, it can be

believed that there must be a negative correlation between depression and Satisfaction with Life; if this is found to be true, a model that predicts depression scores based on Satisfaction with Life scores can be built, and then some of the missing depression scores for users in the Facebook Status Updates dataset can be filled in. Unfortunately, as can see in Figure 8, there is a slight negative correlation between Satisfaction with Life and depression, but there is so much noise and it is not nearly strong enough to build an accurate model (in fact, it was only able to correctly predict the depression classifications about 75% of the time).

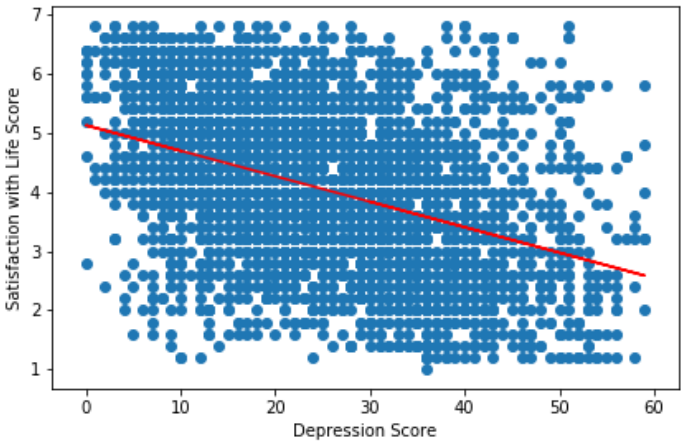


Figure 8: Correlation between Depression and Satisfaction with Life

As the first attempt was not a viable way to increase the sample size, a simpler solution is to duplicate the non-depressed data—this increases sample size while decreasing the skew between the two classes as well. Figure 9 shows the results of training the same models on this duplicated dataset.

	Training Accuracy	Test Accuracy (10%)	Cross-Validation (k=3)
Baseline	59.0%	59.4%	59.0%
Logistic Regression	84.3%	67.7%	70.0%
Support Vector Machines	59.0%	59.4%	59.0%
Random Forests	98.2%	87.2%	85.1%

Figure 9: Results using Duplicated Dataset

The baseline accuracy has gone down to 59%. Support Vector Machines are still mainly classifying everyone as depressed, but there is an improvement in Logistic Regression and Random Forests. While there is still a problem with overfitting, this is a good starting point to seeing how different models are working, so that when new and increased data is available in the

future, these models can be tested first. Once again, a Confusion Matrix for Random Forests can be viewed, as shown in Figure 10.

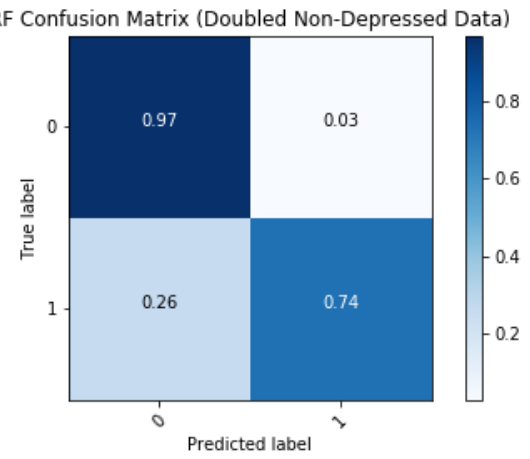


Figure 10: Random Forests Confusion Matrix (Duplicated Data)

The True Positive and True Negative rates are much more balanced now, which is a good sign, showing that the model is not just predicting depressed for every individual now. Something worth noting is that the False Negative rate has increased (the model classifying depressed individuals as not depressed). This leads us to question whether it is better for the model to have more False Positives or False Negatives. Ideally, there would be neither, but it is often the case that different models have tradeoffs such as this. Looking at this problem from the viewpoint of the disease prevention model, we prefer False Positives over False Negatives; this way, individuals who are not truly depressed can be ruled out later, but no individuals who really are depressed will be "missed" by classifying them in the negative class.

Regression

While this is a classification problem, regression techniques can be applied here. This can be done by using the total depression score as the target variable instead of the simple "is depressed" or "is not depressed" classification (as mentioned earlier, total depression score ranges from 0 to 60 on a continuous scale). Using linear regression and random forest regressors, r-squared scores that show the models are performing quite poorly are calculated – 0.02 for

linear regression and 0.09 for random forest regressors (r-squared is a metric for evaluating regression; it is on a scale of 0 to 1, with 1 being the best).

However, these predicted scores from the regression models can be converted back to "is depressed" or "is not depressed" based on the threshold of 16. By doing this and then determining what the accuracy is, the results shown in Figure 11 are found.

	Training Accuracy	Test Accuracy (10%)	Cross-Validation (k=3)
Baseline	60.0%	51.9%	59.0%
Linear Regression	97.7%	86.5%	80.4%
Random Forests	92.8%	70.0%	73.8%

Figure 11: Regression Results

Purely through numbers, it is seen that Linear Regression is performing better than Random Forests by about 6%. However, something else interesting appears when the confusion matrices for these two models are viewed (shown in Figure 12). While Linear Regression shows a better balance between True Positives and True Negatives (as well as False Positives and False Negatives), the False Negative rate is lower in Random Forests than in Linear Regression. The increase of False Positives in Linear Regression is not very high though (17% compared to 21% for False Negatives), but it is good to keep this in mind depending on what specific purpose the model is to be used for.

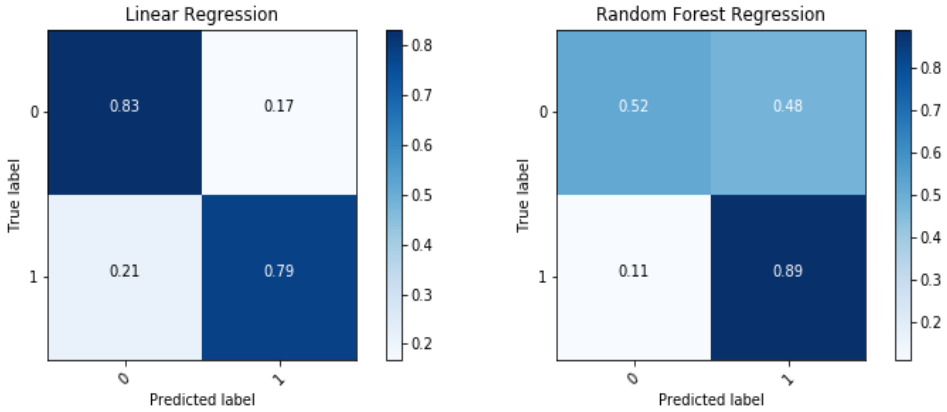


Figure 12: Confusion Matrices for Linear Regression and Random Forests

TF-IDF

Another tool used to improve text analysis models is Term Frequency-Inverse Document Frequency (TF-IDF). TF-IDF is an alternative feature representation to the Bag-of-Words representation this study has used so far. Instead of simply normalizing the counts, TF-IDF accounts for not only the word count, but also the amount of times a word appears in all users' posts. So, if every user has used a specific word several times, TF-IDF will give less importance to this word. Figure 13 shows the results of running classification models where TF-IDF was used.

	Training Accuracy	Test Accuracy (10%)	Cross-Validation (k=3)
Baseline	59.0%	59.4%	59.0%
Logistic Regression	93.1%	74.4%	81.4%
Random Forests	98.2%	83.5%	84.6%

Figure 13: TF-IDF Classification Results

Logistic Regression is overfitting by a high amount (almost 20%) and performs a bit worse than Random Forests. However, Random Forests does not have a high improvement, and is still overfitting by about 15%.

Regression and TF-IDF

Combining regression models and TF-IDF is what turns out to yield the best results (as seen in Figure 14). Linear Regression has an accuracy of about 88%, and the overfitting rate is only about 5% (still a decent amount, but much lower than the 10-20% seen in previous attempts).

	Training Accuracy	Test Accuracy (10%)	Cross-Validation (k=3)
Baseline	59.0%	58.6%	59.0%
Linear Regression	97.0%	91.7%	88.2%
Random Forests	91.5%	80.5%	75.3%

Figure 14: Regression and TF-IDF Results

Looking at the confusion matrix in Figure 15, the True Positive and True Negative rates are both high and approximately even. The False Positive rate is also higher than the False Negative rate, so this is good from the lens of the disease prevention model as well.

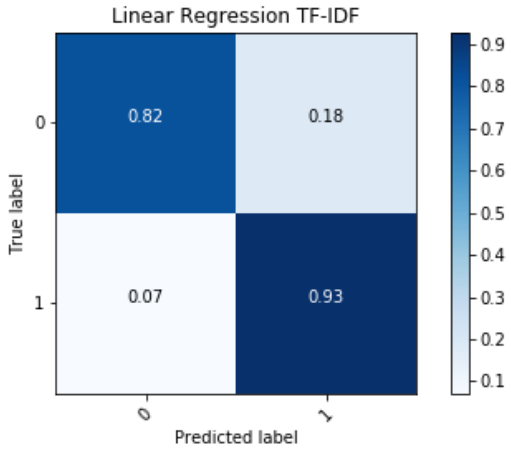


Figure 15: Confusion Matrix for TF-IDF and Linear Regression

It is also possible to look at the words that the linear regression model has the highest coefficients for (the ones that have the highest impact on predicting if a user is depressed or not). In order from lowest to highest, here are the top ten words: “wit”, “headed”, “jeg”, “im”, “tomarrow”, “cleverbot”, “past”, “derby”, “hates”, “slightly”. While some of these words seem unrelated and many are misspelled, some are seen that make sense. “Im” is a first-person pronoun, and there is research showing that the use of more first-person pronouns is a signifier of greater depression (Calvo, et al.). Furthermore, the word “hates” appears, and we could also extract that a person using the word “past” may be more depressed because they are ruminating on the past too often. To confirm any of these findings though, the model needs to be run on a larger dataset.

Feature Engineering

Another technique to improve the models is Feature Engineering. Feature Engineering alters the features used when training models in order to focus on features that have the highest correlations with the target variable. In this case, the features are all the unique words used across everyone’s posts; one technique for feature engineering in a text analysis problem like this is stopword removal. As mentioned previously, the Python library NLTK has a stopword list that

can be used. However, this is an interesting problem in social media because there have been studies that show that the use of first person pronouns shows a greater tendency for depression and other psychological symptoms (Calvo, et al.). By manually altering the stopwords list, we can include many of these first-person pronouns such as “my” and “I”. Performing this alteration did not change the results by any significant amount though.

Other Feature Engineering techniques include limiting the number of features used as predictors. This is especially something to note in this problem because there are over 100,000 unique words (including misspellings, slang, and so on), but only 1000 samples. In machine learning, it is typical to have more samples compared to the number of features. Techniques to limit the features include most commonly used features and mutual information. The first simply picks the top used features—for example, ten thousand features—and ignores all other words. Mutual information looks at both the counts of the words and the target variable, and outputs the correlations these features have with the target variable if the user is depressed. The words with the top correlations can then be taken, and all other words can be removed. Both these techniques were applied to the problem, but unfortunately did not yield any results either.

An alternative feature engineering method is to use different features altogether (instead of the individual words). Since there has been some research showing that the use of first person pronouns signifies a higher risk of depression, it could be useful to use parts of speech as features (Calvo, et al.). Using a Part-of-Speech tagger from NLTK, 45 different parts of speech are used as the features that the models are trained on (results shown in Figure 16).

	Training Accuracy	Test Accuracy (10%)	Cross-Validation (k=3)
Baseline	59.0%	59.4%	59.0%
Logistic Regression	61.8%	60.2%	60.0%
Random Forests	98.1%	85.0%	76.8%

Figure 16: Part-of-Speech Results

Logistic Regression does not perform above baseline, and while Random Forest is performing better, there is still 13% of overfitting.

There can still be more work done in feature engineering for this problem, especially if there is more data. The Part-of-Speech tagger used in these results does not separate first-person pronouns from third-person pronouns, so that would be something to investigate in future work.

Precision-Recall

There are other ways of evaluating the use of classifiers than just simple accuracy percentages; one of these ways is precision-recall. In problems such as this one, precision is the ability to not label negative samples as positive, and recall is the ability to find all the positive samples. We have seen through our initial confusion matrices that our models are easily able to achieve a high recall rate (finding all the depressed users) but have trouble with precision (not having such a high false positive rate). So, instead of focusing on precision-recall together, it is more useful to look at precision by itself.

When machine learning models make their predictions, the predictions are also associated with the probability that algorithm has deemed the prediction to be a positive sample. In this case, since this is a two-class problem, if that probability is over 50%, then the user has been predicted to be at risk for depression.

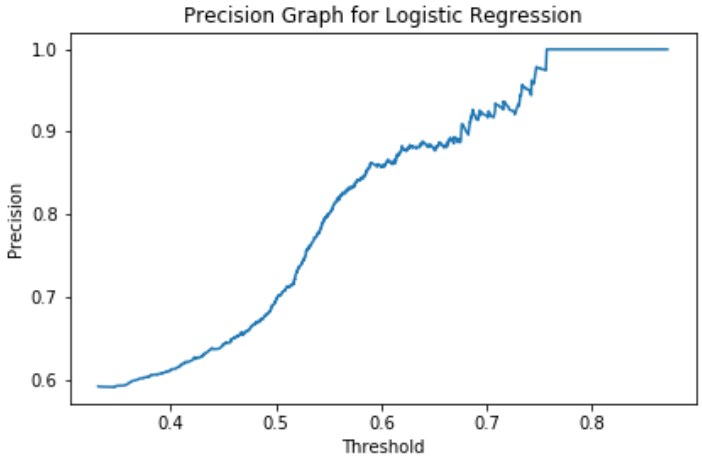


Figure 17: Precision Graph for Logistic Regression

These probabilities can be used as thresholds and we can see how the precision rate changes as the threshold increases (Figure 17 shows this for Logistic Regression). There is a steep increase initially before the slope

decreases, but it does continue to rise. This can be useful in settings where the threshold can be increased, and a higher precision is achieved on those that are still being classified.

Classifying Extremely Depressed Users

These probabilities of predictions can also be viewed against the total depression scores. As depression scores increase and users are more heavily depressed, it is important to make sure these users are being accurately classified since they could be at higher risk for dangerous behaviors. While this is the case when using a regression model, we can also look at this for the classification model of Random Forests as shown in Figure 18.

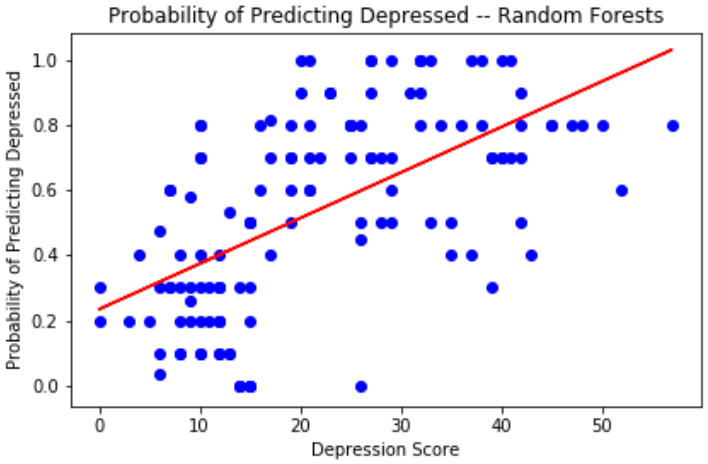


Figure 18: Probability of Predicting Depressed vs. Total Depression Score

While the fit is not perfect and there is a decent amount of noise, there is a positive trend—meaning that the more depressed a user is, the higher chance the model has to classify them as depressed. This is a positive sign that shows us that the model can be used in situations where it is important to specifically target users with a high risk of behaviors associated with depression.

Verifying Results

Because of the small amount of data there is access to, it is hard to verify if these results will be valid in real-time use. Furthermore, the non-depressed data has been duplicated, so even though there appears to be little overfitting in the TF-IDF and regression model, it is possible that there is overfitting occurring that cannot be currently seen.

One way to see if the results will still hold is to train the model on the duplicated data, but test on the original data with no repeats. Unfortunately, when this is attempted, the results found previously do not hold (including the TF-IDF with regression model). One possibility is that these models are simply overfitting to the duplicated data and that is why the accuracies were previously so high. Either way, it is difficult to be sure without having a higher sample of data to train on—preferably a sample that has a more even ratio of depressed and non-depressed users without the need to duplicate any data.

Conclusion

In this supervised learning problem, the features appear to not be as strongly correlated with the target variable; this work shows that such a problem necessitates more data. It is seen that the strongest model employs TF-IDF and regression, but it is unclear if this is a result of overfitting to the duplicated data or not.

While the final results are uncertain, it is also possible to see other methods that can be used to test the models such as precision and examining model probabilities compared to total depression scores. These evaluations showed positive signs but need to be checked when more data is available for this problem as well.

The first step for future work is obtaining a larger sample size, specifically targeting Facebook users who cannot be classified as depressed. Once a larger sample size is obtained, not only can all the models and methods described in this paper be checked, but it will also be possible to move on to new techniques as well. For example, there is much more work to be done in feature engineering. Some examples include more specific part-of-speech tagging, feature limitations, and grouping together words that are misspelled but have the same meaning. Another method to be attempted is using ordinal regression to predict the individual question responses

and then adding those results up to create a predicted value. This problem is clearly an important one, and working on it in these ways can show if it truly is possible to predict depression based off social media.

Appendix

Software

Publicly Available Software/Code	Description
AsterixDB	Big Data Management System used to query and filter data
PANDAS	Python dataframes used to store and manipulate data before using for Machine Learning purposes
NLTK	Python library used for English stopwords list and Part-of-Speech tagging
SciKit-Learn	Python library used for running and evaluating Machine Learning algorithms, as well as creating Bag-of-Words and TF-IDF feature representations from the text of the status updates

Code Written by Me	Description
AsterixDB importing and Dataset Creation	Importing myPersonality CSV and JSON files into AsterixDB and creating Dataset Types to hold this data
AsterixDB into PANDAS	Scripts with queries that moved data in a manner I wanted
Depression Score Calculation	Function to calculate the total depression score of a user
Preprocessing of Data (in Python)	Scripts for cleaning and manipulating data (including transforming data into TF-IDF and BOW matrices using SciKit-Learn)
Scripts to run Algorithms	Scripts using SciKit-Learn to train and test models
Part-of-Speech Feature Representation	Function using NLTK's POS Tagging, and then manipulating that data into a NumPy array that can be used by SciKit-Learn

Jupyter Notebook Description

The following Jupyter notebook demonstrates some of the main high-level concepts in my project. It goes through the process of moving the data from AsterixDB over to PANDAS, as well as the cleaning and preparation of the depression data and status updates. It shows the calculation of the total depression scores, as well as the classification to if the user is at risk for depression or not. We then see the data being merged together by the unique user ID so we have

both our features and target variable. Then we duplicate the non-depressed data and prepare our data to be used by machine learning algorithms implemented by SciKit-Learn. This notebook shows how to do this for our most accurate model of TF-IDF with Regression. We also see how as the probability of predicting depressed (using Random Forest classification) increases as the total depression score increases.

Works Cited

Calvo, Rafael A., et al. "Natural Language Processing in Mental Health Applications Using Non-Clinical Texts." *Natural Language Engineering*, vol. 23, no. 05, 30 Jan. 2017, pp. 649–685, doi:10.1017/s1351324916000383.

Inkster, Becky, et al. "A Decade into Facebook: Where Is Psychiatry in the Digital Age?" *The Lancet Psychiatry*, vol. 3, no. 11, Nov. 2016, pp. 1087–1090., doi:10.1016/s2215-0366(16)30041-4.

Liu, Pan, et al. "Do Facebook Status Updates Reflect Subjective Well-Being?" *Cyberpsychology, Behavior, and Social Networking*, vol. 18, no. 7, 7 Nov. 2015, pp. 373–379., doi:10.1089/cyber.2015.0022.

"Major Depression." *National Institute of Mental Health,* U.S. Department of Health and Human Services, Nov. 2017, www.nimh.nih.gov/health/statistics/major-depression.shtml.

Markovijk, Dejan, et al. *Mining Facebook Data for Predictive Personality Modeling*. International Conference on Weblogs and Social Media, 2013, *Mining Facebook Data for Predictive Personality Modeling.*

In [1]:
```
import pandas as pd
import numpy as np

import json
import requests
import pickle

import matplotlib.pyplot as plt

import itertools

from sklearn.feature_extraction.text import CountVectorizer
from sklearn.feature_extraction.text import TfidfVectorizer

from sklearn.feature_selection import mutual_info_classif

from sklearn.model_selection import cross_val_score
from sklearn.model_selection import cross_val_predict
from sklearn.metrics import accuracy_score

from sklearn.metrics import confusion_matrix

from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier
from sklearn.svm import SVC

from sklearn.ensemble import RandomForestRegressor
from sklearn.linear_model import LinearRegression

from sklearn.model_selection import train_test_split

from sklearn import preprocessing

from nltk.corpus import stopwords
nltkEnglishStopwords = stopwords.words('english')
```

In [53]:
```
# taken from Sci-Kit Learn, used to create confusion matrix graphics

def plot_confusion_matrix(cm, classes,
                          normalize=False,
                          title='Confusion matrix',
                          cmap=plt.cm.Blues):
    """
    This function prints and plots the confusion matrix.
    Normalization can be applied by setting `normalize=True`.
    """
    if normalize:
        cm = cm.astype('float') / cm.sum(axis=1)[:, np.newaxis]
        print("Normalized confusion matrix")
    else:
        print('Confusion matrix, without normalization')

    print(cm)

    plt.imshow(cm, interpolation='nearest', cmap=cmap)
    plt.title(title)
    plt.colorbar()
    tick_marks = np.arange(len(classes))
    plt.xticks(tick_marks, classes, rotation=45)
    plt.yticks(tick_marks, classes)

    fmt = '.2f' if normalize else 'd'
    thresh = cm.max() / 2.
    for i, j in itertools.product(range(cm.shape[0]), range(cm.shape[1])):
        plt.text(j, i, format(cm[i, j], fmt),
                 horizontalalignment="center",
                 color="white" if cm[i, j] > thresh else "black")

    plt.tight_layout()
    plt.ylabel('True label')
    plt.xlabel('Predicted label')
```

Importing from AsterixDB into PANDAS Dataframes

In [2]:
```
url = 'http://localhost:19002/query/service'
head = {'Content-Type': 'application/json'}
```

Importing CES-D Data

In [5]:
```
query = 'SELECT * FROM depressionUsers;'
payload = {'statement': query}
response = requests.post(url, data=payload)
answer = response.json()['results']
cesdItemizedScores = pd.DataFrame(columns=['userid', 'q1', 'q2', 'q3', 'q4',
'q5', 'q6', 'q7', 'q8', 'q9', 'q10', 'q11', \
                                           'q12', 'q13', 'q14', 'q15', 'q16',
'q17', 'q18', 'q19', 'q20'])
```

In [6]:
```
for i in range(0, len(answer)):
    cesdItemizedScores = cesdItemizedScores.append({'userid': answer[i]['depre
ssionUsers']['userid'], \
                                                    'q1': answer[i]['depressio
nUsers']['q1'], \
                                                    'q2': answer[i]['depressio
nUsers']['q2'], \
                                                    'q3': answer[i]['depressio
nUsers']['q3'], \
                                                    'q4': answer[i]['depressio
nUsers']['q4'], \
                                                    'q5': answer[i]['depressio
nUsers']['q5'], \
                                                    'q6': answer[i]['depressio
nUsers']['q6'], \
                                                    'q7': answer[i]['depressio
nUsers']['q7'], \
                                                    'q8': answer[i]['depressio
nUsers']['q8'], \
                                                    'q9': answer[i]['depressio
nUsers']['q9'], \
                                                    'q10': answer[i]['depressi
onUsers']['q10'], \
                                                    'q11': answer[i]['depressi
onUsers']['q11'], \
                                                    'q12': answer[i]['depressi
onUsers']['q12'], \
                                                    'q13': answer[i]['depressi
onUsers']['q13'], \
                                                    'q14': answer[i]['depressi
onUsers']['q14'], \
                                                    'q15': answer[i]['depressi
onUsers']['q15'], \
                                                    'q16': answer[i]['depressi
onUsers']['q16'], \
                                                    'q17': answer[i]['depressi
onUsers']['q17'], \
                                                    'q18': answer[i]['depressi
onUsers']['q18'], \
                                                    'q19': answer[i]['depressi
onUsers']['q19'], \
                                                    'q20': answer[i]['depressi
onUsers']['q20']}, ignore_index=True)
```

In [7]:
```
len(cesdItemizedScores)
```

Out[7]: 6561

There are 6561 users in the CES-D Depression Inventory.

In [8]:
```
cesdItemizedScores.head()
```

Out[8]:

	userid	q1	q2	q3	q4	q5	q6	q7	q8	q9	...	q11	q12	q
0	[redacted]										...			
1	[redacted]										...			
2	[redacted]										...			
3	[redacted]										...			
4	[redacted]										...			

5 rows × 21 columns

Importing Status Updates

We only want to take Status Updates made by users represented in CES-D Dataset, so we filter for that in our query.

In [3]:
```
query = 'SELECT su2.messageid, su2.userid, su2.message FROM statusUpdates AS s
u2 WHERE su2.userid IN \
    (FROM statusUpdates su, depressionUsers du WHERE su.userid = du.userid SEL
ECT VALUE su.userid);'
payload = {'statement':query}
response = requests.post(url, data=payload)
answer = response.json()['results']
```

In [6]:
```
individualStatusUpdates = pd.DataFrame(columns=['messageid', 'userid', 'messag
e'])

for i in range(0, len(answer)):
    individualStatusUpdates = individualStatusUpdates.append({'messageid': ans
wer[i]['messageid'], \
                                                           'userid': answer
[i]['userid'], \
                                                           'message': answe
r[i]['message']}, ignore_index=True)
```

In [7]:
```
len(individualStatusUpdates)
```

Out[7]: 197230

In [8]:
```
individualStatusUpdates.head()
```

Out[8]:

	messageid	userid	message
0	[redacted]	[redacted]	[redacted]
1	[redacted]	[redacted]	[redacted]
2	[redacted]	[redacted]	[redacted]
3	[redacted]	[redacted]	[redacted]
4	[redacted]	[redacted]	[redacted]

After filtering, we are left with only 197230 status updates.

Cleaning and Preparing Data

CES-D Cleaning and Preparation

We first need to remove all the data where users have not responded to all questions in the inventory (signified by a 0 or -1 denoted by myPersonality)

In [12]:
```
toDrop = []

for index, row in cesdItemizedScores.iterrows():
    if row['q1'] == 0 or row['q1'] == -1 or row['q2'] == 0 or row['q2'] == -1
or row['q3'] == 0 or row['q3'] == -1 or \
    row['q4'] == 0 or row['q4'] == -1 or row['q5'] == 0 or row['q5'] == -1 or
row['q6'] == 0 or row['q6'] == -1 or \
    row['q7'] == 0 or row['q7'] == -1 or row['q8'] == 0 or row['q8'] == -1 or
row['q9'] == 0 or row['q9'] == -1 or \
    row['q10'] == 0 or row['q10'] == -1 or row['q11'] == 0 or row['q11'] == -1
 or row['q12'] == 0 or row['q12'] == -1 or \
    row['q13'] == 0 or row['q13'] == -1 or row['q14'] == 0 or row['q14'] == -1
 or row['q15'] == 0 or row['q15'] == -1 or \
    row['q16'] == 0 or row['q16'] == -1 or row['q17'] == 0 or row['q17'] == -1
 or row['q18'] == 0 or row['q18'] == -1 or \
    row['q19'] == 0 or row['q19'] == -1 or row['q20'] == 0 or row['q20'] == -1
:
        toDrop.append(index)

cesdItemizedScores = cesdItemizedScores.drop(toDrop)
```

In [13]:
```
len(cesdItemizedScores)
```

Out[13]: 5833

We also need to subtract one from every question answer (to get standardized scores, questions must range from 0 to 3, but myPersonality's range from 1 to 4).

In [14]:
```
cesdItemizedStandardScores = pd.DataFrame(columns=cesdItemizedScores.columns)

for index, row in cesdItemizedScores.iterrows():
    cesdItemizedStandardScores = cesdItemizedStandardScores.append({'userid':
row['userid'], 'q1': row['q1']-1, \
                                                                    'q2': row[
'q2']-1, 'q3': row['q3']-1, \
                                                                    'q4': row[
'q4']-1, 'q5': row['q5']-1, \
                                                                    'q6': row[
'q6']-1, 'q7': row['q7']-1, \
                                                                    'q8': row[
'q8']-1, 'q9': row['q9']-1, \
                                                                    'q10': row
['q10']-1, 'q11': row['q11']-1, \
                                                                    'q12': row
['q12']-1, 'q13': row['q13']-1, \
                                                                    'q14': row
['q14']-1, 'q15': row['q15']-1, \
                                                                    'q16': row
['q16']-1, 'q17': row['q17']-1, \
                                                                    'q18': row
['q18']-1, 'q19': row['q19']-1, \
                                                                    'q20': row
['q20']-1}, ignore_index=True)
```

We can then write helper functions to calculate the total depression score for each user.

In [16]:
```
def calculateDepressionScore(qScores):
    score = 0
    reverseScoreQuestionIndeces = [3, 7, 11, 15]
    for i in range(0, len(qScores)):
        if i in reverseScoreQuestionIndeces:
            score += findReverseScore(qScores[i])
        else:
            score += qScores[i]

    return score

def findReverseScore(score):
    if score == 3:
        return 0
    if score == 2:
        return 1
    if score == 1:
        return 2
    if score == 0:
        return 3
```

In [17]:
```
cesdTotalScores = pd.DataFrame(columns=['userid', 'totalScore'])

for index, row in cesdItemizedStandardScores.iterrows():
    score = calculateDepressionScore([row['q1'], row['q2'], row['q3'], row['q
4'], row['q5'], row['q6'], row['q7'], \
                                      row['q8'], row['q9'], row['q10'], row['q
11'], row['q12'], row['q13'], row['q14'], \
                                      row['q15'], row['q16'], row['q17'], row[
'q18'], row['q19'], row['q20']])
    cesdTotalScores = cesdTotalScores.append({'userid': row['userid'], 'totalS
core': score}, ignore_index=True)
```

Status Updates Preparation

We need to create a combination of each user's status updates to be stored, instead of the current representation we have with individual status updates. PostgreSQL has a string_agg function to help us do this, but as this is not yet implemented in AsterixDB, we will use PANDAS to do so.

In [22]:
```
groupedStatusUpdates = pd.DataFrame(individualStatusUpdates.groupby('userid')[
'message'].apply(lambda x : ' '.join(x)))
```

In [27]:
```
groupedStatusUpdates = groupedStatusUpdates.reset_index()
```

In [28]:
```
len(groupedStatusUpdates)
```

Out[28]: 1047

In [29]:
```
groupedStatusUpdates.head()
```

Out[29]:

	userid	message
0	[redacted]	[redacted]
1	[redacted]	[redacted]
2	[redacted]	[redacted]
3	[redacted]	[redacted]
4	[redacted]	[redacted]

Merging Status Updates and Depression Scores

In [30]:
```
statusUpdatesWithDepressionScores = cesdTotalScores.join(groupedStatusUpdates.
set_index('userid'), how='inner', on='userid')
```

In [31]:
```
statusUpdatesWithDepressionScores.head()
```

Out[31]:

	userid	totalScore	message
0	[redacted]	[redacted]	[redacted]
2	[redacted]	[redacted]	[redacted]
3	[redacted]	[redacted]	[redacted]
7	[redacted]	[redacted]	[redacted]
3700	[redacted]	[redacted]	[redacted]

Creating Depression Classifications

We can also create a depression classification based off the NIMH threshold of 16.

In [32]:
```
isDepressedList = []
depressionCutoff = 16

for index, row in statusUpdatesWithDepressionScores.iterrows():
    if row['totalScore'] >= depressionCutoff:
        isDepressedList.append(1)
    else:
        isDepressedList.append(0)

statusUpdatesWithDepressionScores["isDepressed"] = isDepressedList
```

In [33]:
```
statusUpdatesWithDepressionScores.head()
```

Out[33]:

	userid	totalScore	message	isDepressed
0	[redacted]	[redacted]	[redacted]	[redacted]
2	[redacted]	[redacted]	[redacted]	[redacted]
3	[redacted]	[redacted]	[redacted]	[redacted]
7	[redacted]	[redacted]	[redacted]	[redacted]
3700	[redacted]	[redacted]	[redacted]	[redacted]

Duplicating Non-Depressed Data

Finally, we will duplicate the non-depressed data so that the two classes are more equally represented.

In [37]:
```
notDepressed = statusUpdatesWithDepressionScores[statusUpdatesWithDepressionSc
ores.isDepressed == 0]
duplicatedData = statusUpdatesWithDepressionScores
duplicatedData = duplicatedData.append([notDepressed], ignore_index=True)
```

In [38]:
```
np.count_nonzero(duplicatedData["isDepressed"])/len(duplicatedData["isDepresse
d"])
```

Out[38]: 0.5900527505651846

In [39]:
```
np.count_nonzero(statusUpdatesWithDepressionScores["isDepressed"])/len(statusU
pdatesWithDepressionScores["isDepressed"])
```

Out[39]: 0.7421800947867299

Depressed users now represent about 60% of the data (as opposed to about 75% before).

Model Training and Evaluation with TF-IDF and Regression

We will now prepare our features for training using TF-IDF, as well as put our target variables in NumPy arrays that can be used by the SciKit-Learn library.

In [41]:
```
vectorizer = TfidfVectorizer(analyzer="word", stop_words=nltkEnglishStopwords)
tfidfmatrix = vectorizer.fit_transform(duplicatedData["message"])
depressionClassifications = np.array(duplicatedData["isDepressed"])
depressionScores = np.array(duplicatedData["totalScore"])
```

This demonstration shows linear regression being used with TF-IDF, and then the corresponding confusion matrix.

In [42]:
```
X_train, X_test, y_train_scores, y_test_scores = train_test_split(tfidfmatrix,
 depressionScores, test_size=0.1)
y_train = np.array([0 if y<16 else 1 for y in y_train_scores])
y_test = np.array([0 if y<16 else 1 for y in y_test_scores])
```

In [43]:
```
lr = LinearRegression()
lr.fit(X_train, y_train_scores)
```

Out[43]:
```
LinearRegression(copy_X=True, fit_intercept=True, n_jobs=1, normalize=False)
```

In [44]:
```
lr_train_predicted_scores = lr.predict(X_train)
lr_train_predicted = np.array([0 if r<16 else 1 for r in lr_train_predicted_sc
ores])

lr_test_predicted_scores = lr.predict(X_test)
lr_test_predicted = np.array([0 if r<16 else 1 for r in lr_test_predicted_scor
es])
```

In [45]:
```
accuracy_score(y_train, lr_train_predicted)
```

Out[45]:
```
0.97571189279731996
```

In [46]:
```
accuracy_score(y_test, lr_test_predicted)
```

Out[46]:
```
0.91729323308270672
```

In [47]:
```
np.count_nonzero(y_train)/len(y_train)
```

Out[47]:
```
0.5896147403685092
```

In [48]:
```
np.count_nonzero(y_test)/len(y_test)
```

Out[48]:
```
0.5939849624060151
```

We see that we get a train accuracy of 97.6%, and a test accuracy of 91.7% (compared to our baseline accuracies of 59.0% and 59.4%).

In [50]:
```
lr = LinearRegression()
lr_predicted_scores = cross_val_predict(lr, tfidfmatrix, depressionScores)
lr_predicted = np.array([0 if l<16 else 1 for l in lr_predicted_scores])
```

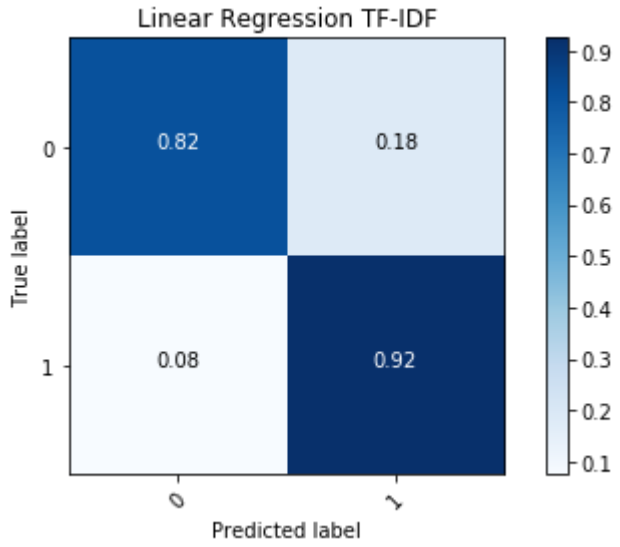
In [51]:
```
accuracy_score(depressionClassifications, lr_predicted)
```

Out[51]: 0.88018085908063304

Using cross-validation, we get a confusion matrix of 88.0%.

In [54]:
```
plt.clf()
cnf_matrix = confusion_matrix(depressionClassifications, lr_predicted)
class_names = [0, 1]
plot_confusion_matrix(cnf_matrix, classes=class_names, normalize=True, title=
"Linear Regression TF-IDF")
plt.tight_layout()
plt.show()
```

```
Normalized confusion matrix
[[ 0.81617647  0.18382353]
 [ 0.07535121  0.92464879]]
```



We can see our true positive rates and true negative rates are both high, and that our false positive rate is higher than the false negative rate (good from a disease prevention viewpoint).

Probabilities of Predicting Depressed vs. Total Depression Score

Using a random forest classification model, we can see that the probability of predicting depressed increases as total depression score increases. This is useful for situtations where we want to make sure we identify depressed individuals who may be at higher risk for more dangerous behaviors.

In [55]:
```
rf = RandomForestClassifier()
rf.fit(X_train, y_train)
```

Out[55]:
```
RandomForestClassifier(bootstrap=True, class_weight=None, criterion='gini',
            max_depth=None, max_features='auto', max_leaf_nodes=None,
            min_impurity_decrease=0.0, min_impurity_split=None,
            min_samples_leaf=1, min_samples_split=2,
            min_weight_fraction_leaf=0.0, n_estimators=10, n_jobs=1,
            oob_score=False, random_state=None, verbose=0,
            warm_start=False)
```

In [56]:
```
rf.score(X_train, y_train)
```

Out[56]:
```
0.98743718592964824
```

In [57]:
```
rf.score(X_test, y_test)
```

Out[57]:
```
0.84210526315789469
```

In [58]:
```
probs = rf.predict_proba(X_test)
```

In [63]:
```
def plotProbsAndScores(probs, y_test_scores):
    plt.clf()

    depressedProbs = [p[1] for p in probs]

    plt.scatter(y_test_scores, depressedProbs, color="blue")
    fit = np.polyfit(np.array(y_test_scores, dtype=float), np.array(depressedP
robs, dtype=float), deg=1)
    plt.plot(y_test_scores, fit[0]*y_test_scores + fit[1], color="red")

    plt.xlabel("Depression Score")
    plt.ylabel("Probability of Predicting Depressed")
    plt.title("Probability of Predicting Depressed -- Random Forests")
    plt.tight_layout()
    plt.show()
```

In [62]:
```
plotProbsAndScores(probs, y_test_scores)
```