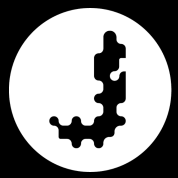
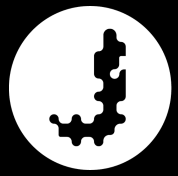


Jigsaw

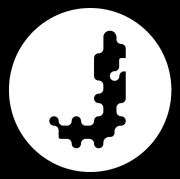
# Improving *Perspective* API Granularity With NBSVM

**Presenter:**
*Francisco McGee*
*Sr. Data Scientist Consultant, TechField*



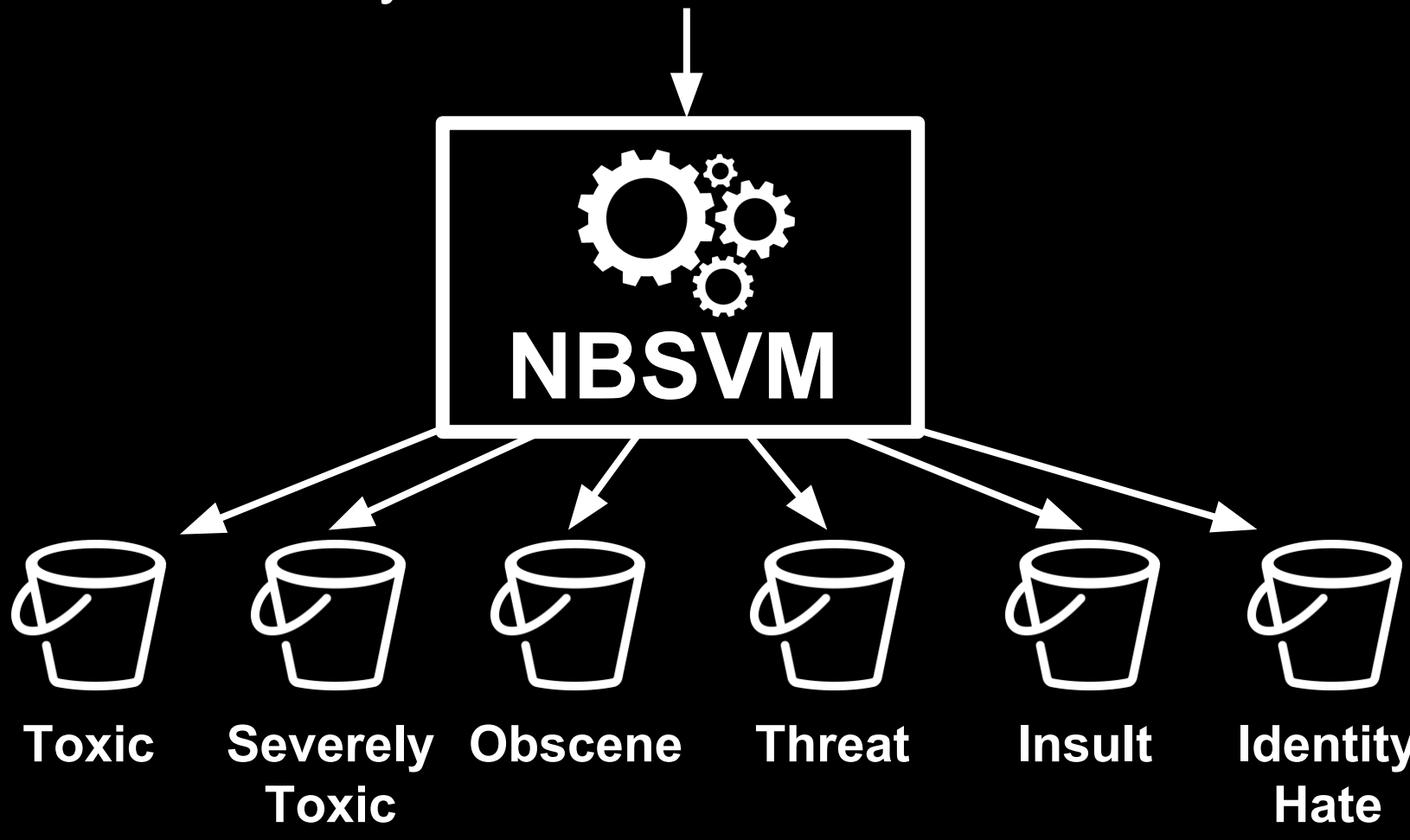
Jigsaw

| comment_id | comment | toxic | Severe _ toxic | obscene | threat | insult | identity_hate |
|---|---|---|---|---|---|---|---|
| 82628967213 | Are you moron or stupid or both? I created my personal page for this, so move all this bloody useful information there if it is necessary. | 1 | 0 | 1 | 0 | 1 | 0 |



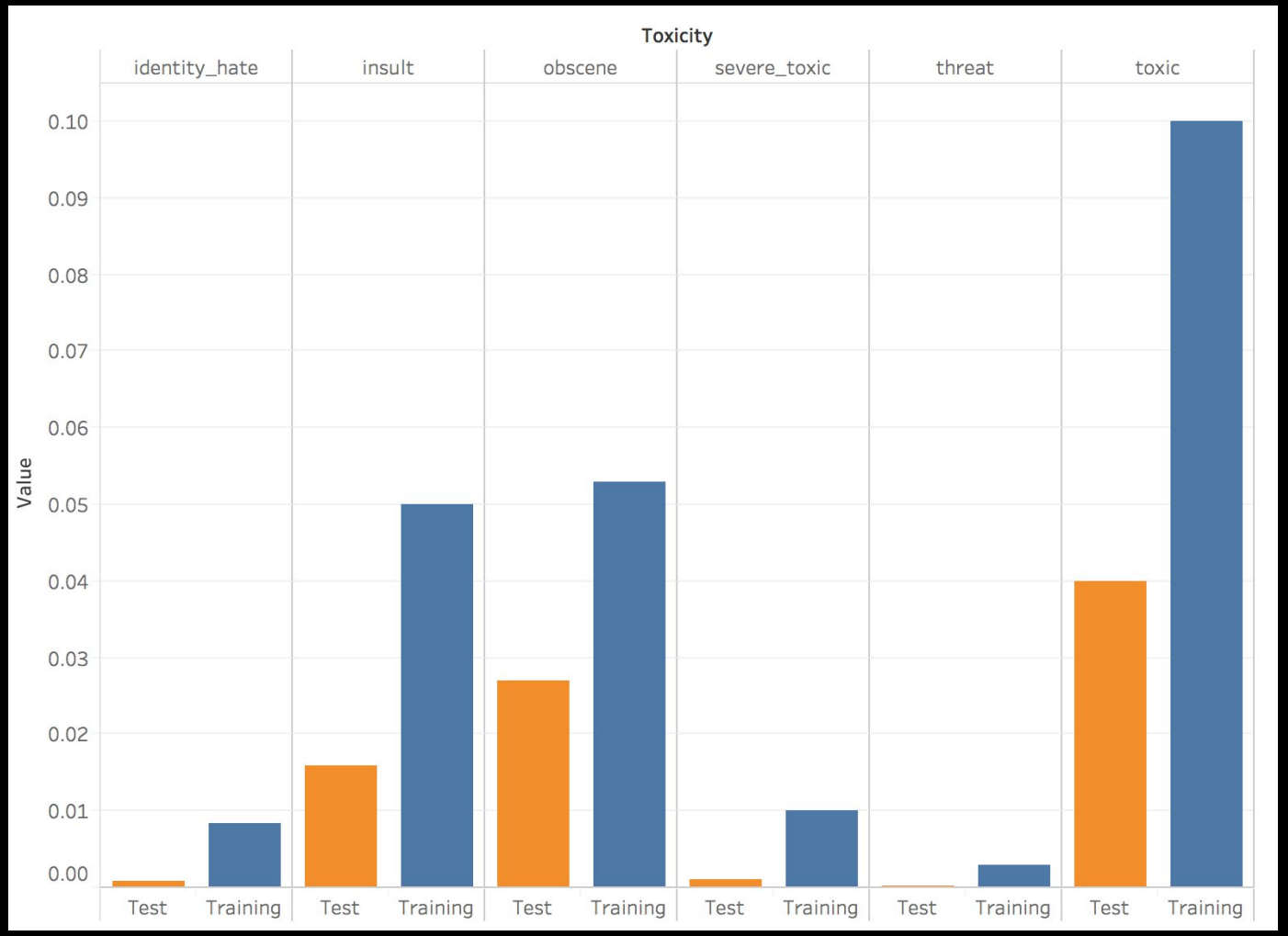
# Jigsaw

“Nonsense?  kiss off, geek. what I said is true.  I'll have your account terminated.”



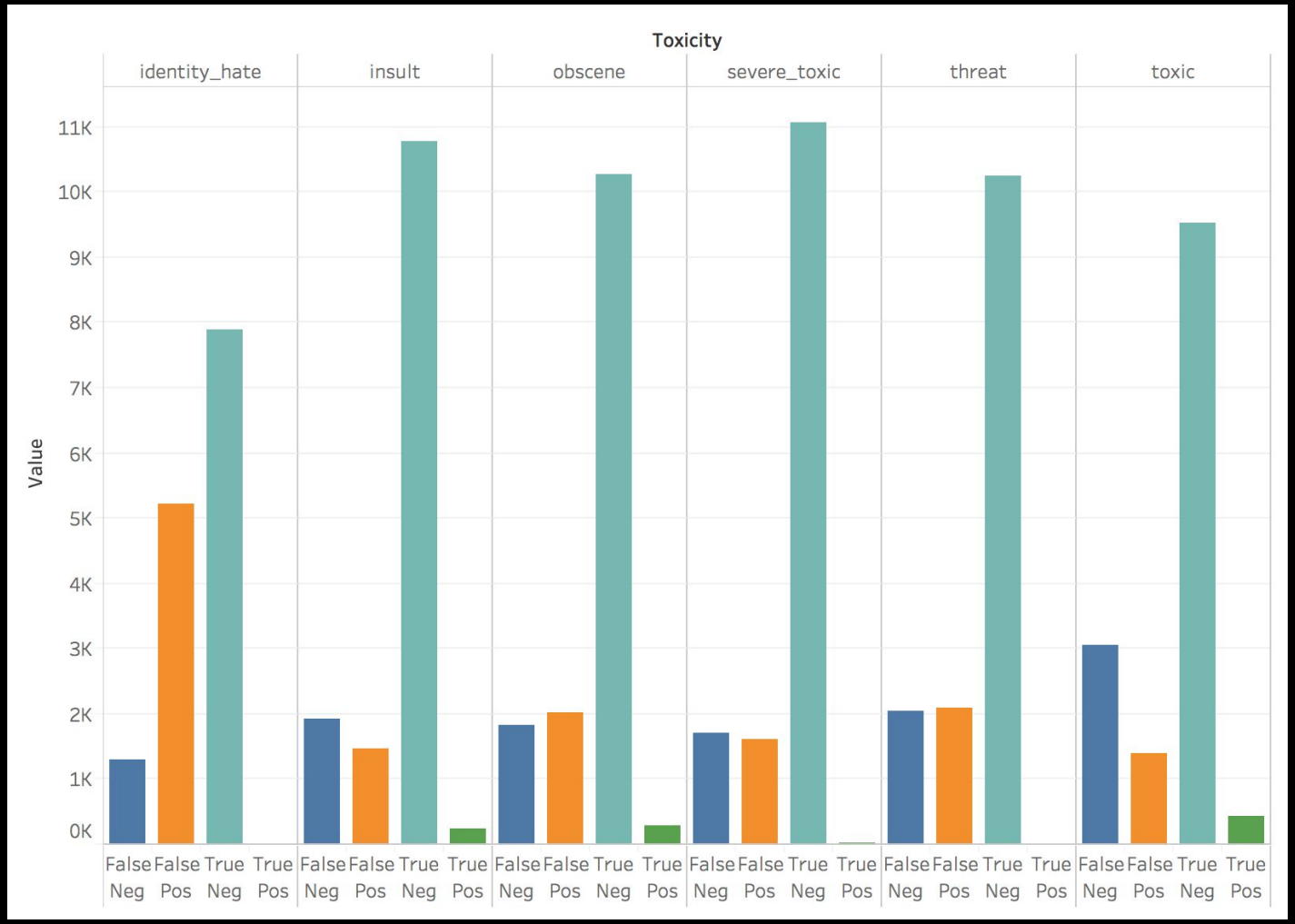
Jigsaw

# Prevalence of Toxicity



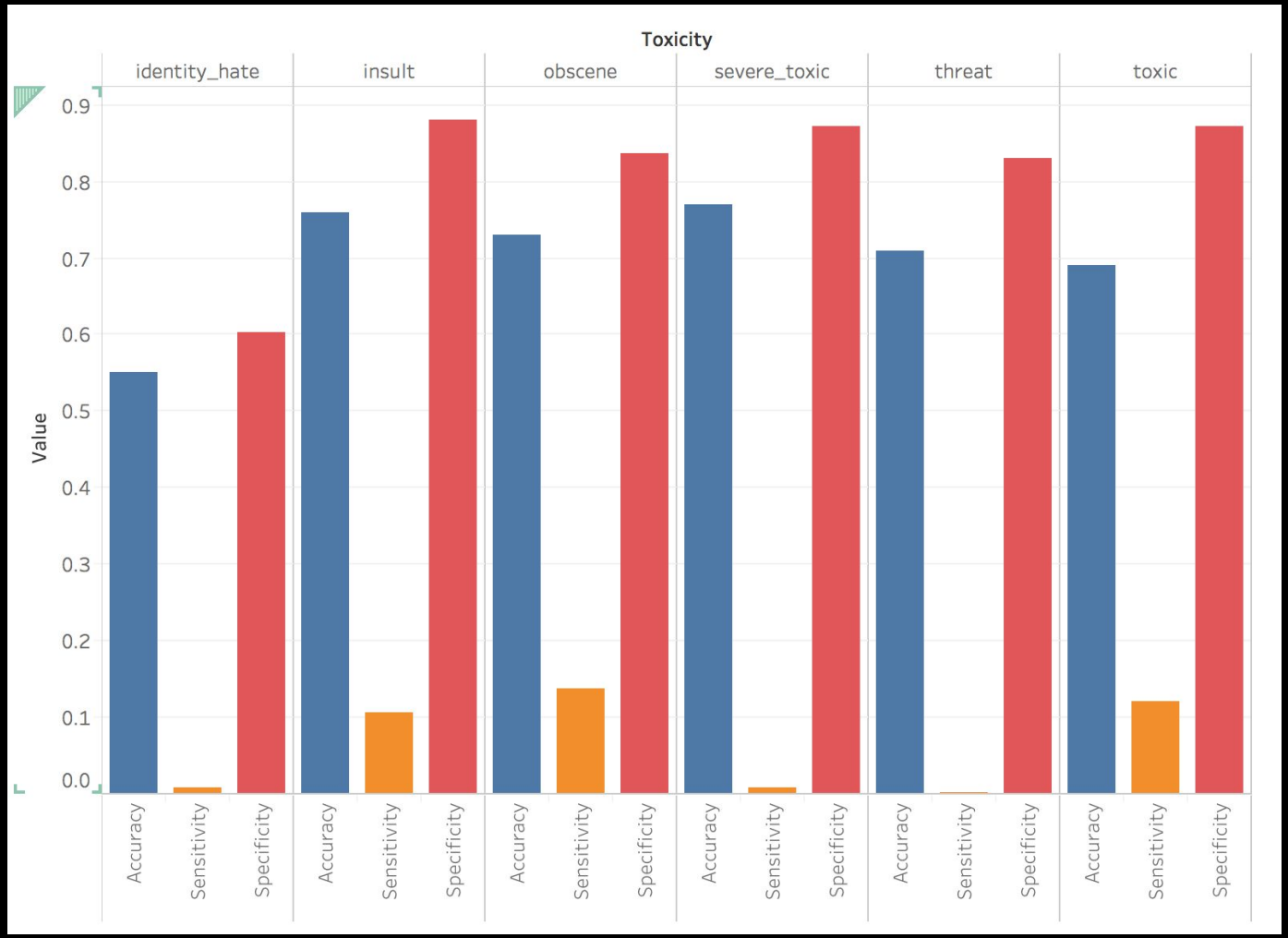
Jigsaw

# Raw Counts



Jigsaw

# Stats



```
# use TF-IDF; better accuracy than CountVectorizer
# create sparse matrix with small number of non-zero elements
# this code takes some time, but is not the bottleneck
n = train.shape[0]
vec = TfidfVectorizer(ngram_range=(1,2), tokenizer=tokenize,
min_df=3, max_df=0.9, strip_accents='unicode', use_idf=1,
smooth_idf=1, sublinear_tf=1 )
NB feature equation
# this code will extract log-count ratios from the ngram vectors,
# turning them into features, which get plugged into the logistic regression
def pr(y_i, y):
p = x[y==y_i].sum(0)
return (p+1) / ((y==y_i).sum()+1)
```

```
# fit a model for one dependent at a time
# this is probably the longest running function in the code
def get_mdl(y):
y = y.values
r = np.log(pr(1,y) / pr(0,y))
m = LogisticRegression(C=4, dual=True)
x_nb = x.multiply(r)
return m.fit(x_nb, y), r

# make a matrix of zeros to plug results into
preds = np.zeros((len(test), len(label_cols)))

# longest run time of code
# duration probably due to get_mdl, but could be the actual prediction
# this loop calculates the number that goes in each cell
for i, j in enumerate(label_cols):
print('fit', j)
m,r = get_mdl(train[j])
preds[:,i] = m.predict_proba(test_x.multiply(r))[:,1]
```