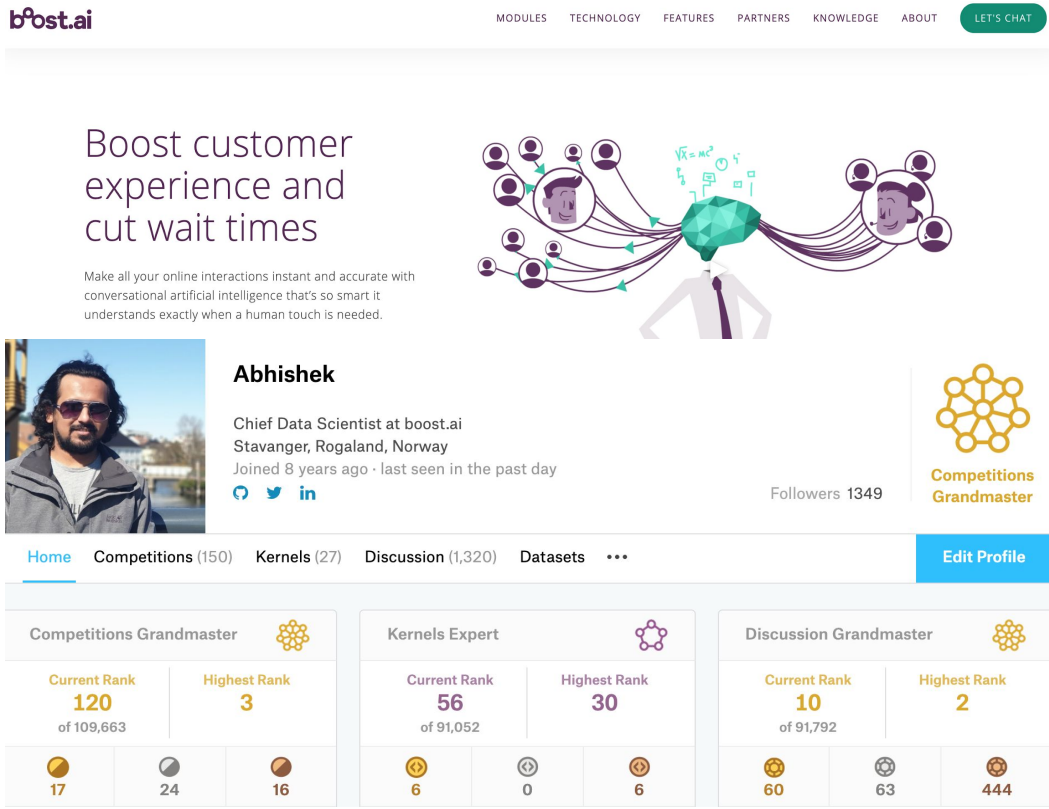
Approaching (almost) Any Machine Learning Problem

@abhi1thakur

Who am I?

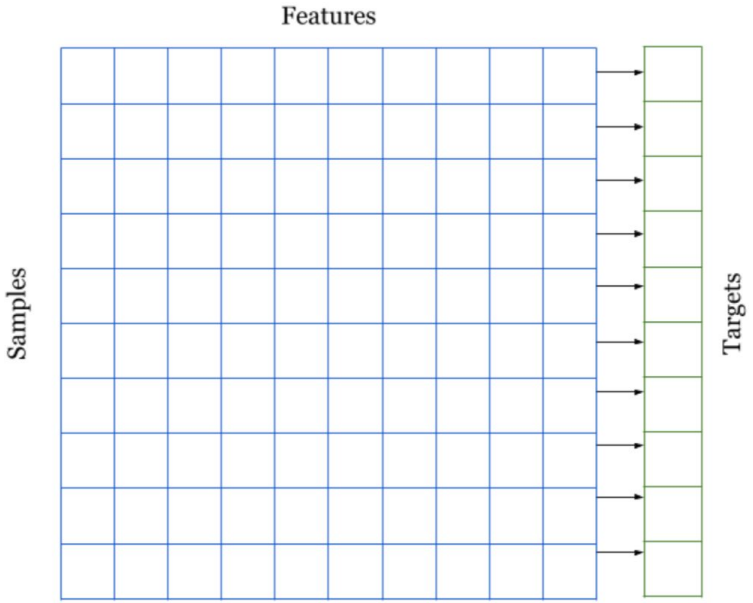
- Chief Data Scientist @ boost.ai
- Top Kaggle rank # 3
- Double Grandmaster
  - Competitions
  - Discussions



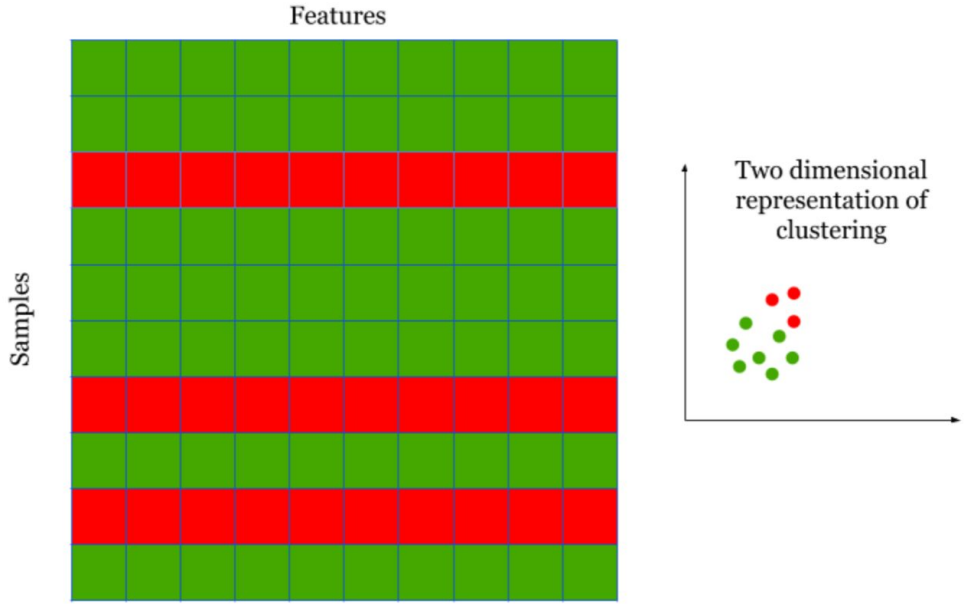
Agenda

- Types of machine learning problems
- Evaluation metrics
- Cross validation
- Categorical data handling
- Numeric data handling
- Text data handling
- Image data handling..... No No
- Hyperparameter tuning

Supervised vs. Un-supervised



Supervised vs. Un-supervised



Classification vs. Regression

- Classification
  - Predicting a class
  - Discrete
- Regression
  - Predicting a value
  - Continuous

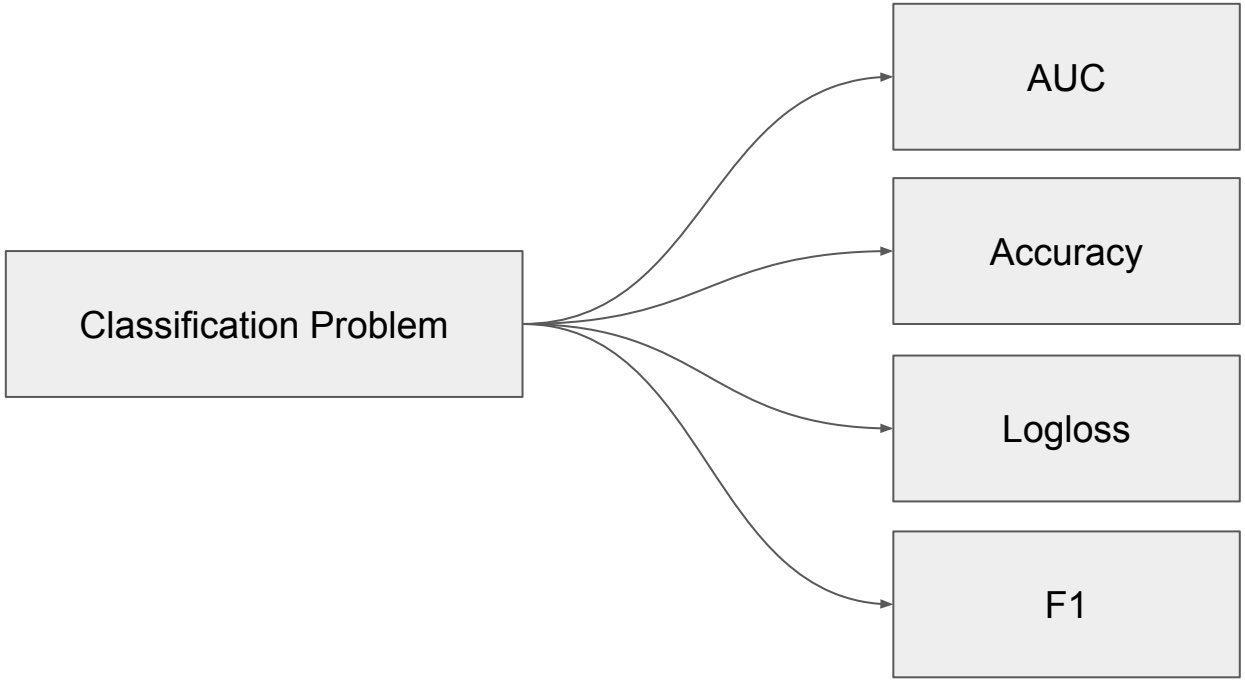
Classification vs. Regression

- **Single column, binary values**
  - classification problem, one sample belongs to one class only and there are only two classes
- **Single column, real values**
  - regression problem, prediction of only one value
- **Multiple column, binary values**
  - classification problem, one sample belongs to one class, but there are more than two classes
- **Multiple column, real values**
  - regression problem, prediction of multiple values
- **Multilabel**
  - classification problem, one sample can belong to several classes

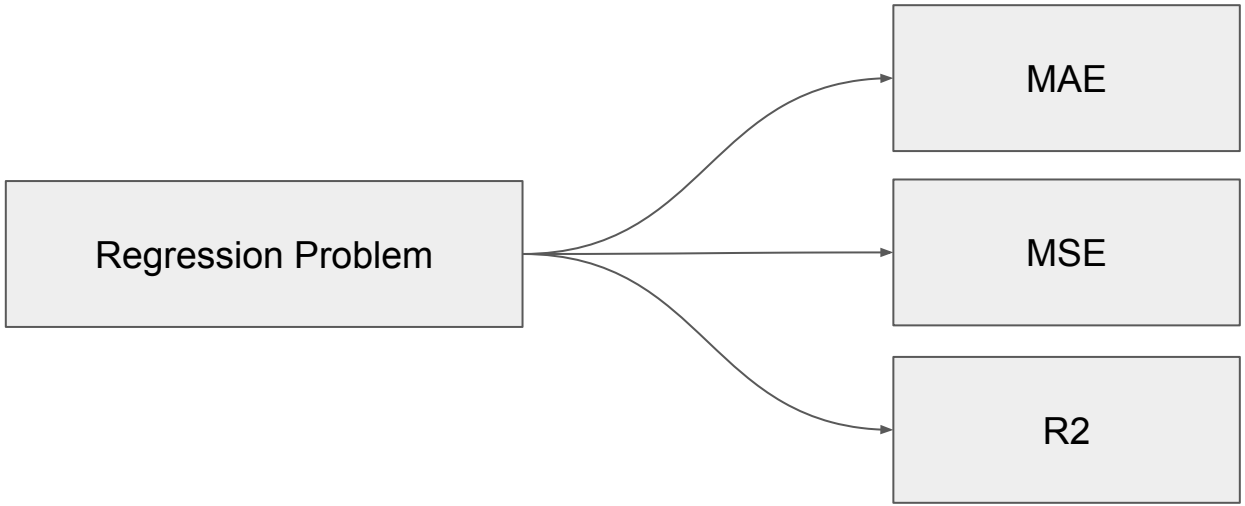
Evaluation metrics

- Classification accuracy
- ROC AUC
- MAE
- RMSE
- Logloss
- AP@k
- And many more

Choosing the appropriate metric



Choosing the appropriate metric



Let’s take a look at some data

1	39353	85475	117961	118300
1	17183	1540	117961	118343
1	36724	14457	118219	118220
1	36135	5396	117961	118343
1	42680	5905	117929	117930
0	45333	14561	117951	117952
1	25993	17227	117961	118343
1	19666	4209	117961	117969
1	31246	783	117961	118413
1	78766	56683	118079	118080
1	4675	3005	117961	118413
1	15030	94005	117902	118041
1	79954	46608	118315	118463
1	4675	50997	91261	118026
1	95836	18181	117961	118343

Let's take a look at some data

```
df.var_0.value_counts()
```

```
1     30872
0      1897
Name: var_0, dtype: int64
```

1	39353	85475	117961	118300
1	17183	1540	117961	118343
1	36724	14457	118219	118220
1	36135	5396	117961	118343
1	42680	5905	117929	117930
0	45333	14561	117951	117952
1	25993	17227	117961	118343
1	19666	4209	117961	117969
1	31246	783	117961	118413
1	78766	56683	118079	118080
1	4675	3005	117961	118413
1	15030	94005	117902	118041
1	79954	46608	118315	118463
1	4675	50997	91261	118026
1	95836	18181	117961	118343

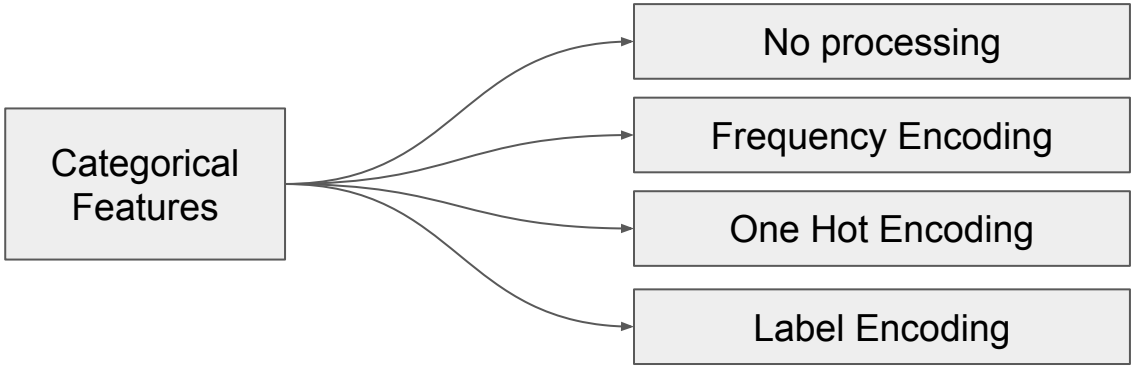
Let's take a look at some data

```
df.var_1.value_counts()
```

```
4675     839
79092    484
25993    409
75078    409
3853     404
6977     299
75834    299
32270    295
42085    247
17308    239
1020     236
13878    220
```

1	39353	85475	117961	118300
1	17183	1540	117961	118343
1	36724	14457	118219	118220
1	36135	5396	117961	118343
1	42680	5905	117929	117930
0	45333	14561	117951	117952
1	25993	17227	117961	118343
1	19666	4209	117961	117969
1	31246	783	117961	118413
1	78766	56683	118079	118080
1	4675	3005	117961	118413
1	15030	94005	117902	118041
1	79954	46608	118315	118463
1	4675	50997	91261	118026
1	95836	18181	117961	118343

Handling categorical data



Handling rare categorical data

```
20897      104
33642       99
           ...
35046        1
92378        1
100413       1
79728        1
79792        1
36894        1
28635        1
59370        1
```

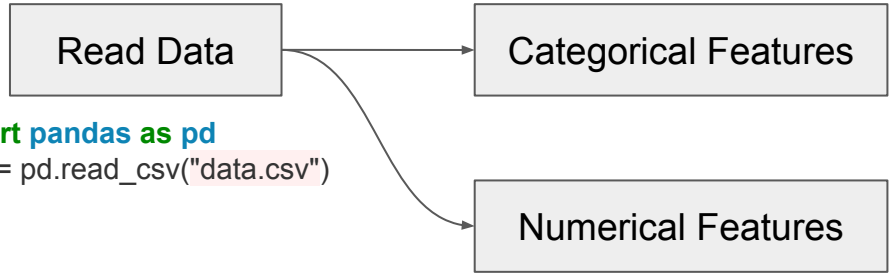
```
group = df.groupby('feature')
group.filter(lambda x: len(x) >= 100)
df.loc[df[col].value_counts()[df[col]].values < 10, col] = "RARE_VALUE"
```

Let’s take a look at some other data

B	C	0.569745	0.594646	0.714843
D	C	0.338312	0.366307	0.304496
D	C	0.381398	0.373424	0.774425
D	C	0.327915	0.32157	0.602642
B	D	0.204687	0.202213	0.432606
D	D	0.366788	0.359249	0.726792
D	D	0.334828	0.352251	0.382931
B	C	0.644013	0.785706	0.242416
C	C	0.682315	0.669033	0.361191
B	C	0.863052	0.879347	0.294523
D	D	0.550529	0.538473	0.715009
B	D	0.644013	0.665644	0.799124
D	C	0.327915	0.32157	0.818358
B	C	0.257148	0.253044	0.477578
B	C	0.880469	0.871011	0.251278
D	C	0.385085	0.377003	0.340325
D	D	0.457203	0.447145	0.205651
B	C	0.678924	0.665644	0.407411
D	D	0.678924	0.665644	0.310796
B	C	0.388786	0.40609	0.830931

Building a pipeline

```
from sklearn.preprocessing import OneHotEncoder
categorical_features = ["var_1", "var_2", ...]
ohe = OneHotEncoder()
ohe.fit(data[categorical_features])
transformed_data = ohe.transform(data[categorical_features])
```




```
import pandas as pd
data = pd.read_csv("data.csv")
```

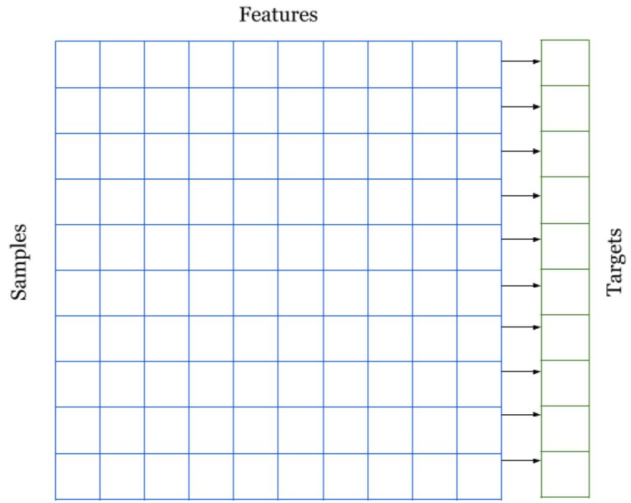
```
target = data.target
num_features = data.drop(["target", "id"] + categorical_features,
                         axis=1)
```

Building a pipeline

Categorical Features

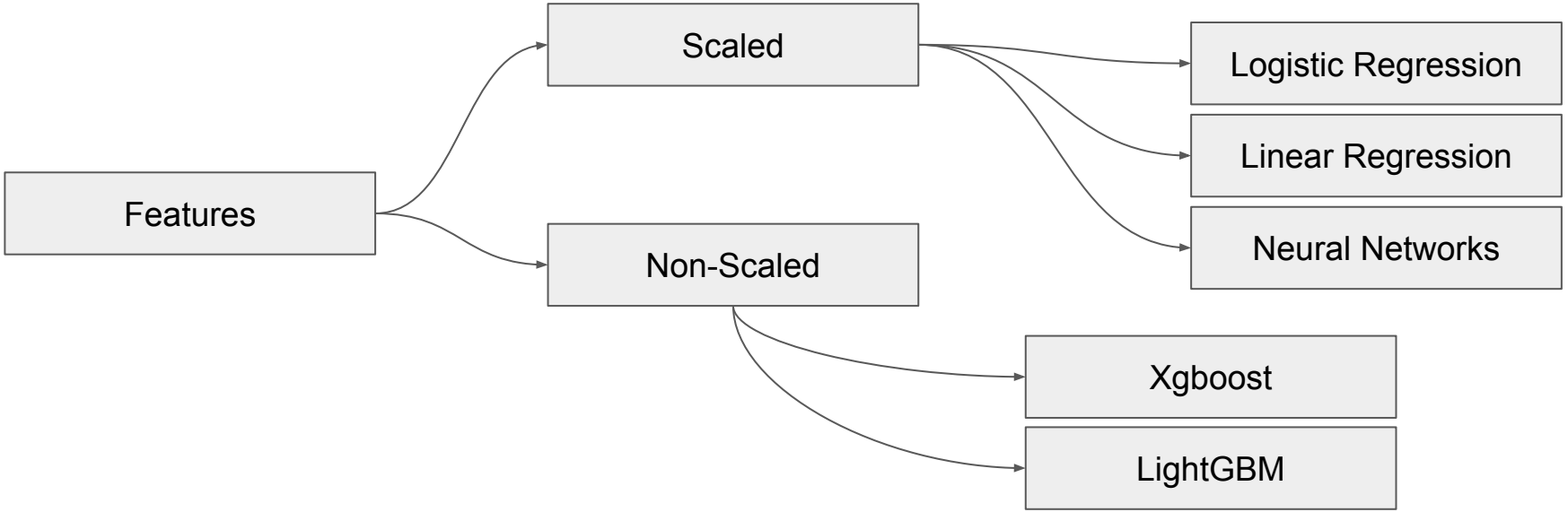
Numerical Features

```
from scipy import sparse
features = sparse.hstack((num_features, categorical_features))
```

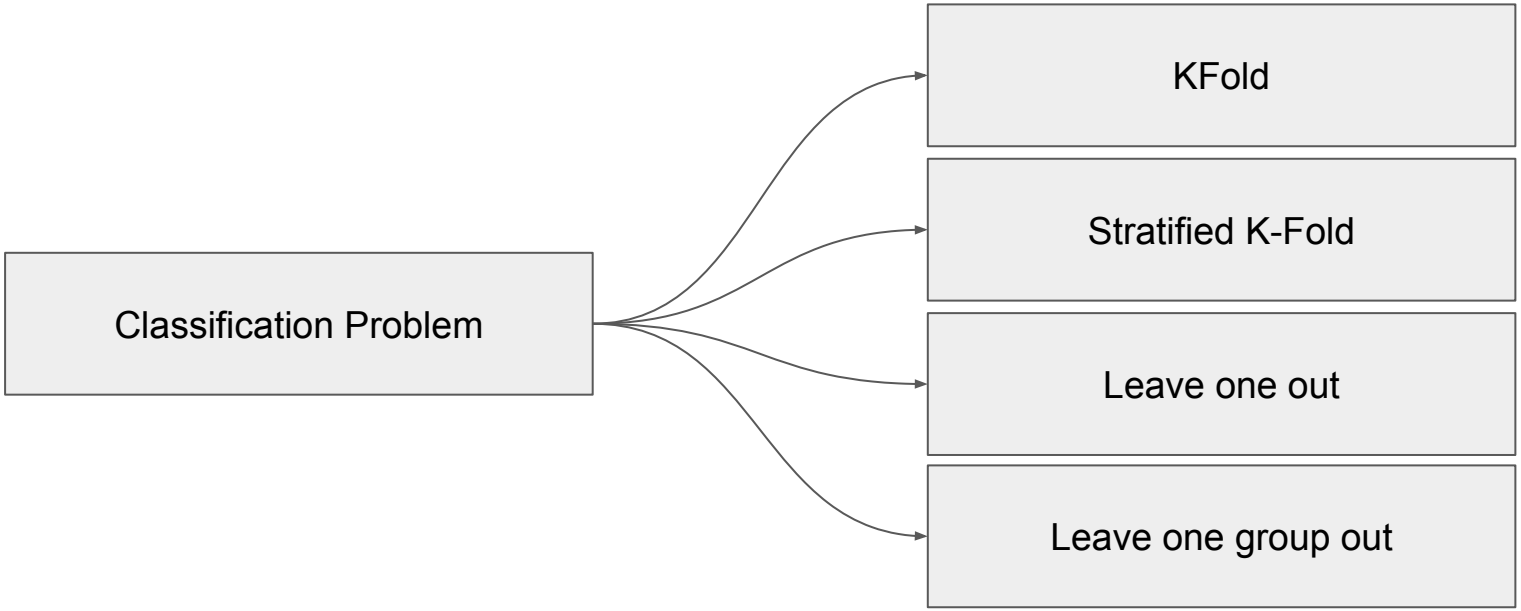


Building a model

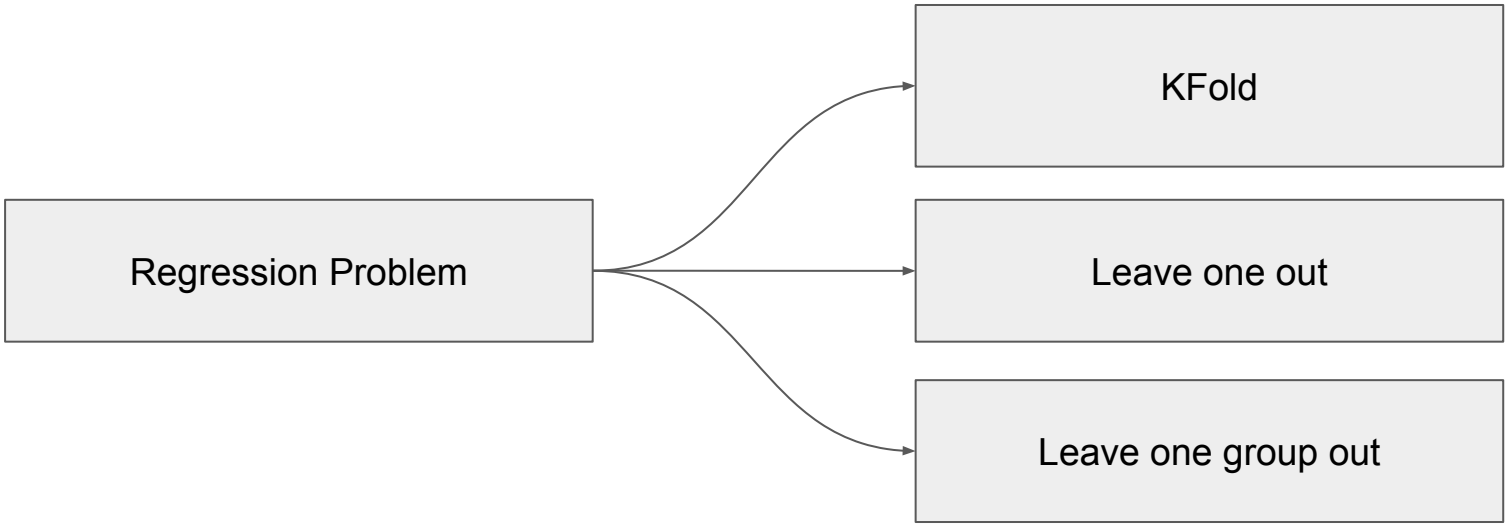
```
from sklearn.preprocessing import StandardScaler
from sklearn.preprocessing import RobustScaler
from sklearn.preprocessing import MinMaxScaler
```



Cross-validation



Cross-validation



Putting it all together: the beginning

```
import os
import shutil
import pandas as pd
import numpy as np
from sklearn.model_selection import StratifiedKFold

NFOLDS = 5
RANDOM_STATE = 42

script_name = os.path.basename(__file__).split('.')[0]
MODEL_NAME = "{0}__folds{1}".format(script_name, NFOLDS)
```

Putting it all together: reading the data

```
print("Reading training data")
train = pd.read_csv('../input/train.csv')
test = pd.read_csv('../input/test.csv')

y = train.target.values
train_ids = train.ID_code.values
train = train.drop(['id', 'target'], axis=1)
feature_list = train.columns

test_ids = test.ID_code.values
test = test[feature_list]

X = train.values.astype(float)
X_test = test.values.astype(float)
```

Putting it all together: the backbone pt.1

```
clfs = []
folds = StratifiedKFold(n_splits=NFOLDS, shuffle=True, random_state=RANDOM_STATE)
oof_preds = np.zeros((len(train), 1))
test_preds = np.zeros((len(test), 1))
```

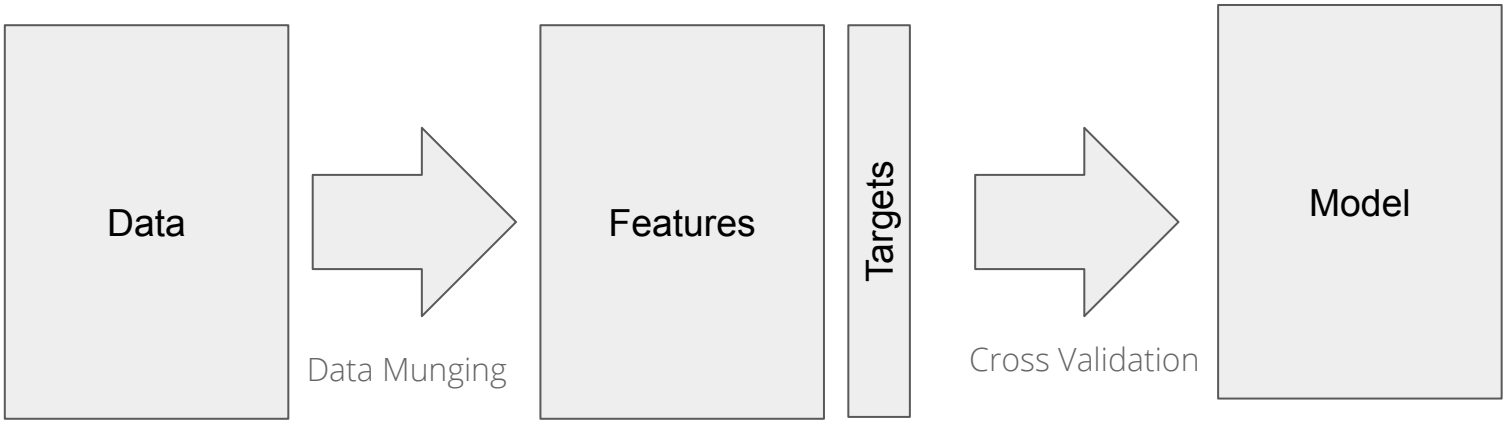
Putting it all together: the backbone pt.2

```
for fold_, (trn_, val_) in enumerate(folds.split(y, y)):
    print("Current Fold: {}".format(fold_))
    trn_x, trn_y = X[trn_, :], y[trn_]
    val_x, val_y = X[val_, :], y[val_]

    clf = DEFINE MODEL HERE

    # FIT MODEL HERE

    val_pred = GENERATE PREDICTIONS FOR VALIDATION DATA
    test_fold_pred = GENERATE PREDICTIONS FOR TEST DATA

    print("AUC = {}".format(metrics.roc_auc_score(val_y, val_pred)))
    oof_preds[val_, :] = val_pred.reshape((-1, 1))
    test_preds += test_fold_pred.reshape((-1, 1))

test_preds /= NFOLDS
```

Putting it all together: the end

```
roc_score = metrics.roc_auc_score(y, oof_preds.ravel())
print("Overall AUC = {}".format(roc_score))

print("Saving OOF predictions")
oof_preds = pd.DataFrame(np.column_stack((train_ids, oof_preds.ravel())), columns=['id', 'target'])
oof_preds.to_csv('../kfolds/{}__{}.csv'.format(MODEL_NAME, str(roc_score)), index=False)

print("Saving code to reproduce")
shutil.copyfile(os.path.basename(__file__), '../model_source/{}__{}.py'.format(MODEL_NAME, str(roc_score)))
```

Putting it all together



Something else for categories

1	39353	85475	117961	118300
1	17183	1540	117961	118343
1	36724	14457	118219	118220
1	36135	5396	117961	118343
1	42680	5905	117929	117930
0	45333	14561	117951	117952
1	25993	17227	117961	118343
1	19666	4209	117961	117969
1	31246	783	117961	118413
1	78766	56683	118079	118080
1	4675	3005	117961	118413
1	15030	94005	117902	118041
1	79954	46608	118315	118463
1	4675	50997	91261	118026
1	95836	18181	117961	118343

Entity embeddings: data

MachineIdentifier	EngineVersion	AppVersion	AvSigVersion	AVProductStatesIdentifier	AVProductsInstalled	Wdft_IsGamer	Wdft_RegionIdentifier	HasDetections
0000028988387b115f69f31a3bf04f09	1.1.15100.1	4.18.1807.18075	1.273.1735.0	53447.0	1.0	0.0	10.0	0
000007535c3f730efa9ea0b7ef1bd645	1.1.14600.4	4.13.17134.1	1.263.48.0	53447.0	1.0	0.0	8.0	0
000007905a28d863f6d0d597892cd692	1.1.15100.1	4.18.1807.18075	1.273.1341.0	53447.0	1.0	0.0	3.0	0
00000b11598a75ea8ba1beea8459149f	1.1.15100.1	4.18.1807.18075	1.273.1527.0	53447.0	1.0	0.0	3.0	1
000014a5f00daa18e76b81417eeb99fc	1.1.15100.1	4.18.1807.18075	1.273.1379.0	53447.0	1.0	0.0	1.0	1

Generating entity embeddings

```
train = pd.read_csv('../input/train.csv', dtype=dtypes)
test = pd.read_csv('../input/test.csv', dtype=dtypes)
test['HasDetections'] = -1
test = test[train.columns]
data = pd.concat([train, test])

usable_cols = []
for c in dtypes:
    if dtypes[c] == 'category' or data[c].nunique() < 15000:
        usable_cols.append(c)
usable_cols += ['HasDetections', 'MachineIdentifier']
usable_cols = list(set([x for x in usable_cols]))
```

Generating entity embeddings

```
for c in data.columns:
    if c == 'HasDetections' or c == 'MachineIdentifier':
        continue
    lbl = preprocessing.LabelEncoder()
    data[c] = lbl.fit_transform(data[c].astype(str))
```

Generating entity embeddings

```
model, cols = create_model(data)
```

Generating entity embeddings

```
train_data = data[data.HasDetections != -1]
test_data = data[data.HasDetections == -1]
test_idx = test_data.MachineIdentifier.values

train_len = int(len(train_data) * 0.9)

xtrain, xvalid = train_data.iloc[:train_len], train_data.iloc[train_len:]
```

Generating entity embeddings

```
model.fit_generator(generator=data_generator(xtrain, cols, batch_size, True),
                    steps_per_epoch=np.ceil(len(xtrain)/batch_size),
                    validation_data=data_generator(xvalid, cols, batch_size, False),
                    validation_steps=np.ceil(len(xvalid)/batch_size),
                    epochs=1000,
                    verbose=1)
```


That's all Folks!

Generating entity embeddings

```
def create_model(data):
    col_order = []
    inputs = []
    outputs = []
    for c in data.columns:
        num_unique_values = int(data[c].nunique())
        embed_dim = int(min(np.ceil((num_unique_values)/2), 50))
        inp = Input(shape=(1,))
        out = Embedding(num_unique_values + 1, embed_dim,
                        name=c, embeddings_initializer='he_normal')(inp)
        out = SpatialDropout1D(0.3)(out)
        out = Reshape(target_shape=(embed_dim,))(out)
        inputs.append(inp)
        outputs.append(out)
        col_order.append(c)
    x = Concatenate()(outputs)
    x = BatchNormalization()(x)
    x = Dropout(0.3)(x)
    x = Dense(1024, activation="relu")(x)
    x = Dropout(0.3)(x)
    x = BatchNormalization()(x)
    x = Dense(32, activation="relu")(x)
    x = Dropout(0.3)(x)
    x = BatchNormalization()(x)
    y = Dense(1, activation="sigmoid")(x)

    model = Model(inputs=inputs, outputs=y)
    model.compile(loss='binary_crossentropy', optimizer='adam')
    return model, col_order
```

Generating entity embeddings

```
def data_generator(data, cols, batch_size, shuffle, test_mode=False):
    num_batches = np.ceil(len(data) / batch_size)
    idx = np.arange(len(data))
    ctr = 0
    X = data[cols].values
    if test_mode is False:
        y = data.HasDetections.values
    if shuffle:
        np.random.shuffle(idx)
    while True:
        b_idx = idx[batch_size * ctr: batch_size * (ctr + 1)]
        if test_mode is False:
            xb = X[b_idx, :]
            yb = y[b_idx]
            ctr += 1
            yield [xb[:, k] for k in range(xb.shape[1])], yb
```

Handling of categorical data

- Convert to numbers : LabelEncoder
- Convert to one hot: OneHotEncoder
- Convert to binary: LabelBinarizer
- Convert to counts
- Convert to embeddings: tf/keras
- Creating more categories?
- Using factorization machines? (libfm/libffm)

Feature engineering from numerical features

- Transformations

```
data["new_feature"] = data["feature"].apply(np.log)
data["new_feature"] = data["feature"].apply(np.exp)
```

Feature engineering from numerical features

- Transformations
- Binning

```
for c in num_cols:
    dx = pd.DataFrame(data[c], columns=[c])
    data[c + "_bin"] = pd.cut(data[c], bins=100, labels=False)
```

Feature engineering from numerical features

- Transformations
- Binning
- Interactions

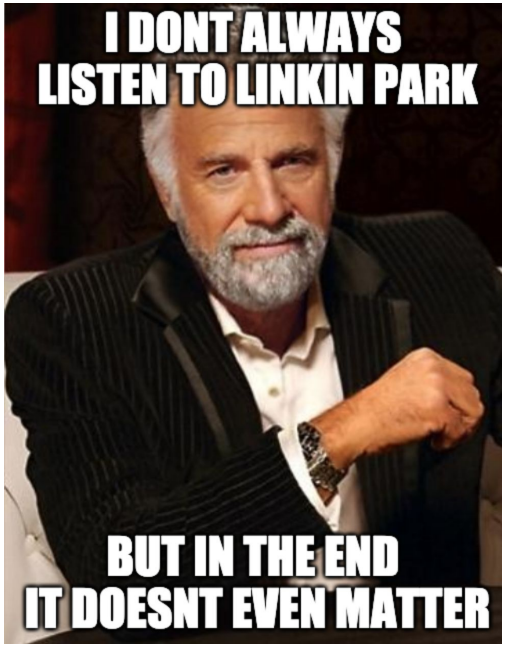
```
from sklearn.preprocessing import PolynomialFeatures

pf = PolynomialFeatures(degree=2,
                        interaction_only=False,
                        include_bias=False)
pf.fit(training_matrix)
transformed_training = pf.transform(training_matrix)

>>> [a, b, a**2, ab, b**2]
```

Selecting the best ones

- Recursively eliminating the features
- Based on model

- Select top N features: SelectKBest
- Selecting a percentile: SelectPercentile
- Mutual information based
- Chi2 based

Selecting the best ones




When I make spaghetti I
throw a noodle against
the wall to see if it sticks.
If it does it's done.

When the data changes shape

first_active_month	card_id	feature_1	feature_2	feature_3	target
2017-06-01	C_ID_92a2005557	5	2	1	-0.820283
2017-01-01	C_ID_3d0044924f	4	1	0	0.392913
2016-08-01	C_ID_d639edf6cd	2	2	0	0.688056
2017-09-01	C_ID_186d6a6901	4	3	0	0.142495
2017-11-01	C_ID_cdbd2c0db2	1	3	0	-0.159749

When the data changes shape

card_id	city_id	category_1	installments	category_3	merchant_category_id	merchant_id	month_lag	purchase_amount	purchase_date	category_2	state_id	subsector_id
C_ID_4e6213e9bc	88	N	0	A	80	M_ID_e020e9b302	-8	-0.703331	2017-06-25 15:33:07	1.0	16	37
C_ID_4e6213e9bc	88	N	0	A	367	M_ID_86ec983688	-7	-0.733128	2017-07-15 12:10:45	1.0	16	16
C_ID_4e6213e9bc	88	N	0	A	80	M_ID_979ed661fc	-6	-0.720386	2017-08-09 22:04:29	1.0	16	37
C_ID_4e6213e9bc	88	N	0	A	560	M_ID_e6d5ae8ea6	-5	-0.735352	2017-09-02 10:06:26	1.0	16	34
C_ID_4e6213e9bc	88	N	0	A	80	M_ID_e020e9b302	-11	-0.722865	2017-03-10 01:14:19	1.0	16	37

Creating aggregate features

```
def features(df):
    df.loc[:, 'year'] = df['purchase_date'].dt.year
    df.loc[:, 'weekofyear'] = df['purchase_date'].dt.weekofyear
    df.loc[:, 'month'] = df['purchase_date'].dt.month
    df.loc[:, 'dayofweek'] = df['purchase_date'].dt.dayofweek
    df.loc[:, 'weekend'] = (df.purchase_date.dt.weekday >=5).astype(int)
    df.loc[:, 'hour'] = df['purchase_date'].dt.hour
    aggs = {}
    aggs['month'] = ['nunique', 'mean']
    aggs['hour'] = ['nunique', 'mean']
    aggs['weekofyear'] = ['nunique', 'mean']
    aggs['merchant_id'] = ['nunique']
    aggs['merchant_category_id'] = ['nunique']
    aggs['purchase_amount'] = ['sum','max','min','mean','var']
    aggs['purchase_date'] = ['max','min']
    aggs['card_id'] = ['size']
    agg_df = df.groupby('card_id').agg(aggs)
    agg_df = agg_df.reset_index()
    return agg_df
```

- Mean
- Max
- Min
- Unique
- Skew
- Kurtosis
- Kstat
- Percentile

Creating aggregate features

```
def features(df):
    df.loc[:, 'year'] = df['purchase_date'].dt.year
    df.loc[:, 'weekofyear'] = df['purchase_date'].dt.weekofyear
    df.loc[:, 'month'] = df['purchase_date'].dt.month
    df.loc[:, 'dayofweek'] = df['purchase_date'].dt.dayofweek
    df.loc[:, 'weekend'] = (df.purchase_date.dt.weekday >=5).astype(int)
    df.loc[:, 'hour'] = df['purchase_date'].dt.hour
    aggs = {}
    aggs['month'] = ['nunique', 'mean']
    aggs['hour'] = ['nunique', 'mean']
    aggs['weekofyear'] = ['nunique', 'mean']
    aggs['merchant_id'] = ['nunique']
    aggs['merchant_category_id'] = ['nunique']
    aggs['purchase_amount'] = ['sum','max','min','mean','var', q2]
    aggs['purchase_date'] = ['max','min']
    aggs['card_id'] = ['size']
    agg_df = df.groupby('card_id').agg(aggs)
    agg_df = agg_df.reset_index()
    return agg_df
```

- Mean
- Max
- Min
- Unique
- Skew
- Kurtosis
- Kstat
- Percentile
- And many more

```
def q2(x):
    return np.quantile(x, 0.25)
```

When there is just one feature

acoustic_data	time_to_failure
-1	1.446796536445618
2	1.461698055267334
6	1.469099998474121
7	1.466999292373657
8	1.460597276687622
8	1.459597229957581
1	1.466996431350708
2	1.458499670028687
3	1.459597349166870
5	1.448898434638977
17	1.458497762680054

- Ongoing Kaggle competition
- Training set: one large file with two columns
- Test set: Files with 150k rows


NO DATA
NO CRY

When there is just one feature

```
iter_df = pd.read_csv(filename, iterator=True, chunksize=150000,
                      dtype={'acoustic_data': np.float64,
                             'time_to_failure': np.float64})
for counter, df in enumerate(iter_df):
    x = df.acoustic_data.values
    y = df.time_to_failure.values[-1]
    seg_id = 'train_' + str(counter)
    yield seg_id, x, y
```

➢ Divide training set into chunks

https://www.kaggle.com/abhishek/quite-a-few-features-1-51

When there is just one feature

```
feature_dict = {}
feature_dict['mean'] = np.mean(x)
feature_dict['max'] = np.max(x)
feature_dict['min'] = np.min(x)
feature_dict['std'] = np.std(x)
feature_dict['var'] = np.var(x)
feature_dict['ptp'] = np.ptp(x)
feature_dict['percentile_10'] = np.percentile(x, 10)

feature_dict['abs_energy'] = feature_calculators.abs_energy(x)
feature_dict['count_above_mean'] = feature_calculators.count_above_mean(x)
feature_dict['count_below_mean'] = feature_calculators.count_below_mean(x)
feature_dict['mean_abs_change'] = feature_calculators.mean_abs_change(x)
feature_dict['mean_change'] = feature_calculators.mean_change(x)
```

```
from tsfresh.feature_extraction import feature_calculators
```

- Divide training set into chunks
- Create features on the chunks

https://www.kaggle.com/abhishek/quite-a-few-features-1-51

When there is no numerical feature

	id	qid1	qid2	question1	question2	is_duplicate
0	0	1	2	What is the step by step guide to invest in sh...	What is the step by step guide to invest in sh...	0
1	1	3	4	What is the story of Kohinoor (Koh-i-Noor) Dia...	What would happen if the Indian government sto...	0
2	2	5	6	How can I increase the speed of my internet co...	How can Internet speed be increased by hacking...	0
3	3	7	8	Why am I mentally very lonely? How can I solve...	Find the remainder when [math]23^{24}[/math] i...	0
4	4	9	10	Which one dissolve in water quikly sugar, salt...	Which fish would survive in salt water?	0

Handling text data

- Length of question1
- Length of question2
- Difference in the two lengths
- Character length of question1 without spaces
- Character length of question2 without spaces
- Number of words in question1
- Number of words in question2
- Number of common words in question1 and question2

Handling text data

- Basic feature set: fs-1

```
data['len_q1'] = data.question1.apply(lambda x: len(str(x)))
data['len_q2'] = data.question2.apply(lambda x: len(str(x)))
data['diff_len'] = data.len_q1 - data.len_q2
data['len_char_q1'] = data.question1.apply(lambda x: len(''.join(set(str(x).replace(' ', '')))))
data['len_char_q2'] = data.question2.apply(lambda x: len(''.join(set(str(x).replace(' ', '')))))
data['len_word_q1'] = data.question1.apply(lambda x: len(str(x).split()))
data['len_word_q2'] = data.question2.apply(lambda x: len(str(x).split()))
data['common_words'] = data.apply(lambda x:
     len(set(str(x['question1']).lower().split()).intersection(set(str(x['question2']).lower().split()))), axis=1)
```

Fuzzy features

- Also known as approximate string matching
- Number of “primitive” operations required to convert string to exact match
- Primitive operations:
  - Insertion
  - Deletion
  - Substitution
- Typically used for:
  - Spell checking
  - Plagiarism detection
  - DNA sequence matching
  - Spam filtering

Fuzzy features

- pip install fuzzywuzzy

- Uses Levenshtein distance
- QRatio
- WRatio
- Token set ratio
- Token sort ratio
- Partial token set ratio
- Partial token sort ratio
- etc. etc. etc.

Fuzzy features

```
data['fuzz_qratio'] = data.apply(lambda x: fuzz.QRatio(str(x['question1']), str(x['question2'])), axis=1)
data['fuzz_WRatio'] = data.apply(lambda x: fuzz.WRatio(str(x['question1']), str(x['question2'])), axis=1)
data['fuzz_partial_ratio'] = data.apply(lambda x: fuzz.partial_ratio(str(x['question1']), str(x['question2'])), axis=1)
data['fuzz_partial_token_set_ratio'] = data.apply(lambda x: fuzz.partial_token_set_ratio(str(x['question1']), str(x['question2'])),
                                                  axis=1)
data['fuzz_partial_token_sort_ratio'] = data.apply(lambda x: fuzz.partial_token_sort_ratio(str(x['question1']),
                                                                                           str(x['question2'])), axis=1)
data['fuzz_token_set_ratio'] = data.apply(lambda x: fuzz.token_set_ratio(str(x['question1']), str(x['question2'])), axis=1)
data['fuzz_token_sort_ratio'] = data.apply(lambda x: fuzz.token_sort_ratio(str(x['question1']), str(x['question2'])), axis=1)
```

TF-IDF

- TF(t) = Number of times a term t appears in a document / Total number of terms in the document
- IDF(t) = log(Total number of documents / Number of documents with term t in it)
- TF-IDF(t) = TF(t) * IDF(t)

```
tfidf = TfidfVectorizer(min_df=3,  max_features=None,
            strip_accents='unicode', analyzer='word',token_pattern=r'\w{1,}',
            ngram_range=(1, 2), use_idf=1, smooth_idf=1, sublinear_tf=1,
            stop_words = 'english')
```

SVD

- Latent semantic analysis
- scikit-learn version of SVD
- 120 components

```
svd = decomposition.TruncatedSVD(n_components=120)
xtrain_svd = svd.fit_transform(xtrain)
xtest_svd = svd.transform(xtest)
```

A combination of TF-IDF and SVD

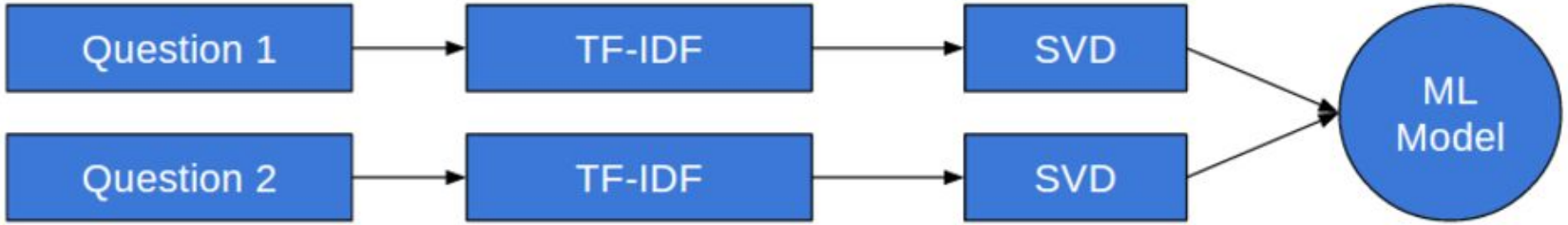
1.

A combination of TF-IDF and SVD

2.

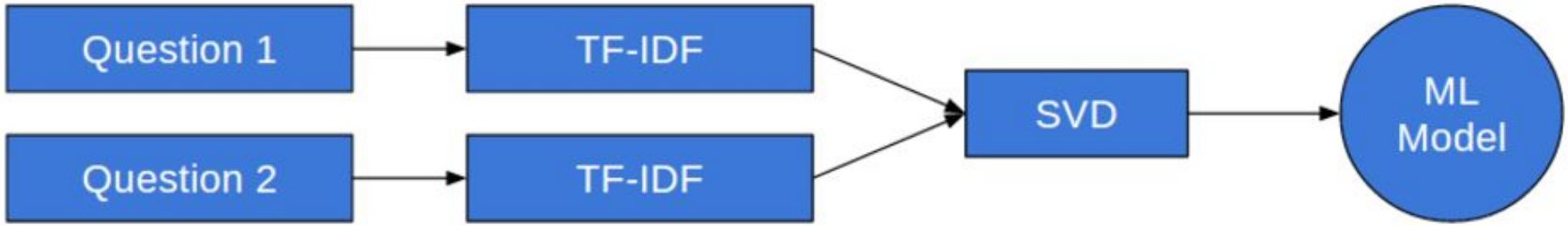
A combination of TF-IDF and SVD

3.



A combination of TF-IDF and SVD

4.



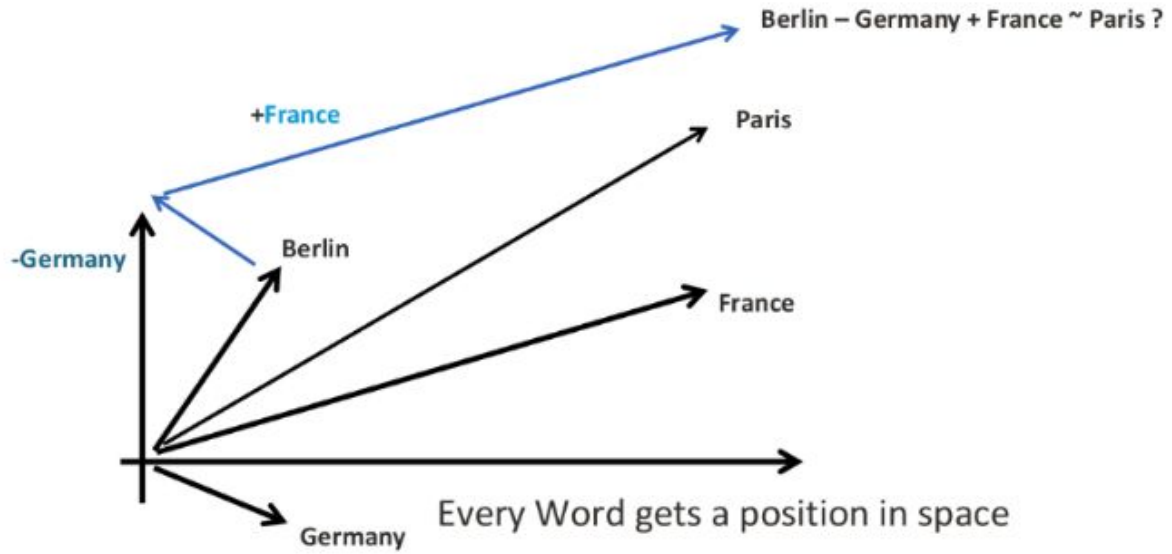
A combination of TF-IDF and SVD

5.

Word2Vec Features

- Multi-dimensional vector for all the words in any dictionary
- Always great insights
- Very popular in natural language processing tasks
- Google news vectors 300d

Word2Vec Features

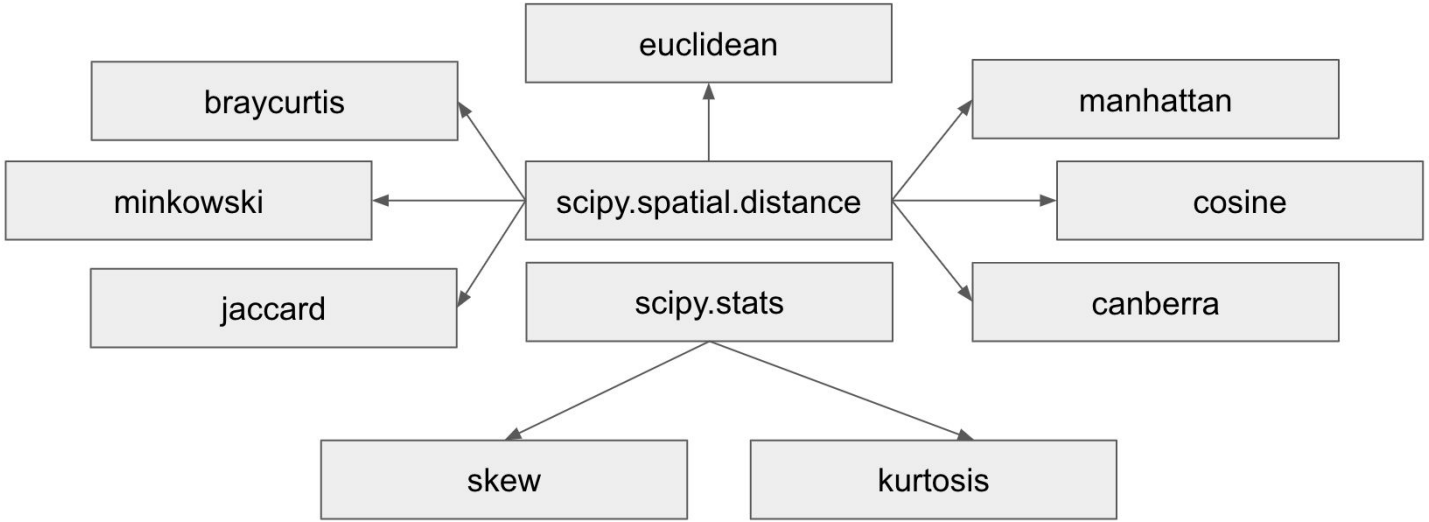


Word2Vec Features

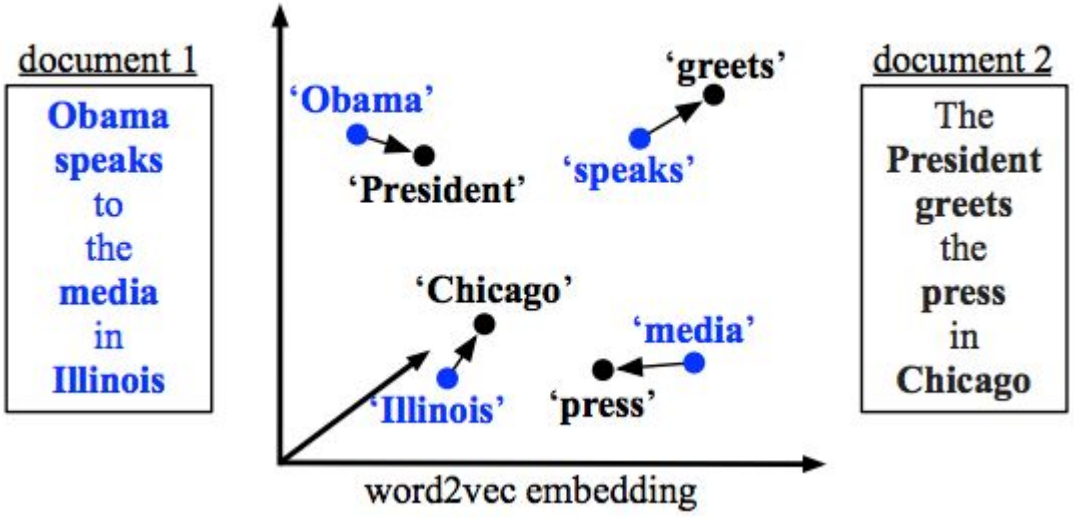
- Representing words
- Representing sentences

```
def sent2vec(s):
    words = str(s).lower()
    words = word_tokenize(words)
    words = [w for w in words if not w in stop_words]
    words = [w for w in words if w.isalpha()]
    M = []
    for w in words:
        M.append(model[w])
    M = np.array(M)
    v = M.sum(axis=0)
    return v / np.sqrt((v ** 2).sum())
```

Word2Vec Features



Word2Vec Features: WMD



Kusner, M., Sun, Y., Kolkin, N. & Weinberger, K.. (2015). From Word Embeddings To Document Distances.

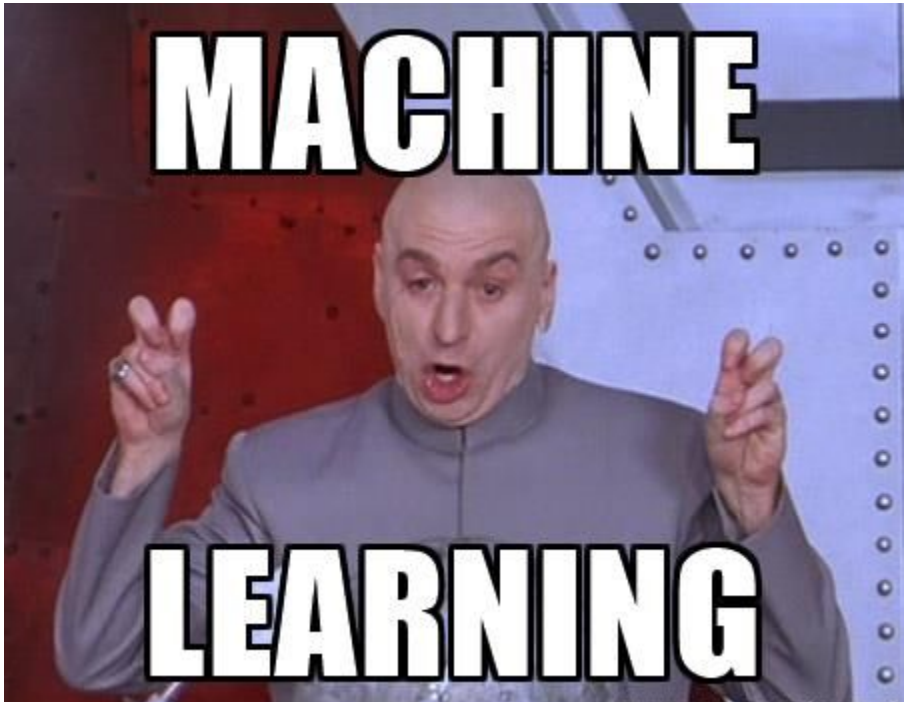
Feature snapshot

question1	What is the story of Kohinoor (Koh-i-Noor) Dia...
question2	What would happen if the Indian government sto...
is_duplicate	0
len_q1	51
len_q2	88
diff_len	-37
len_char_q1	21
len_char_q2	29
len_word_q1	8
len_word_q2	13
common_words	4
fuzz_qratio	66
fuzz_WRatio	86
fuzz_partial_ratio	73
fuzz_partial_token_set_ratio	100
fuzz_partial_token_sort_ratio	75
fuzz_token_set_ratio	86
fuzz_token_sort_ratio	63

Feature snapshot

question1	What is the story of Kohinoor (Koh-i-Noor) Dia...
question2	What would happen if the Indian government sto...
is_duplicate	0
wmd	3.77235
norm_wmd	1.3688
cosine_distance	0.512164
cityblock_distance	14.1951
jaccard_distance	1
canberra_distance	177.588
euclidean_distance	1.01209
minkowski_distance	0.45591
braycurtis_distance	0.592655
skew_q1vec	0.00873466
skew_q2vec	0.0947038
kur_q1vec	0.28401
kur_q2vec	-0.034444

Machine learning models



Machine learning models

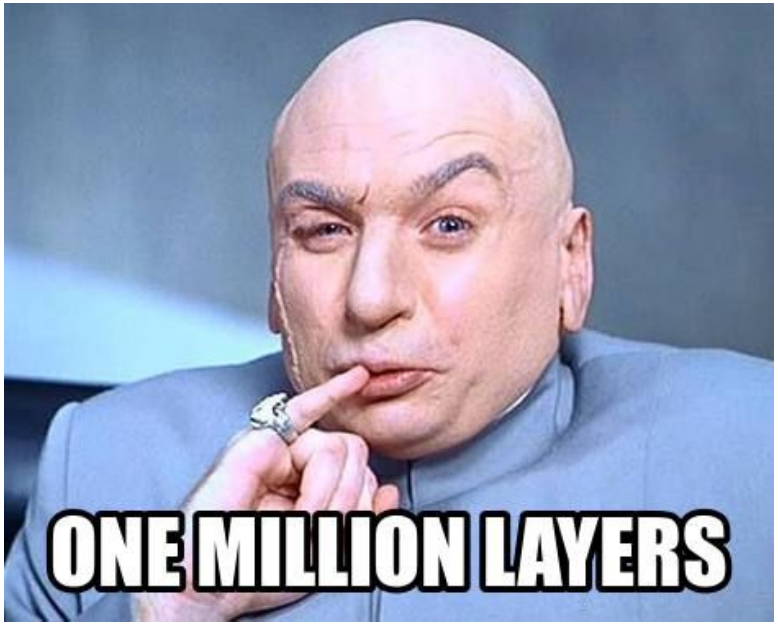
- Logistic regression
- Xgboost
- 5 fold cross-validation
- Accuracy as a comparison metric (also, precision + recall)
- Why accuracy?

Results

Feature Set	Logistic Regression Accuracy	Xgboost Accuracy
Basic features (fs1)	0.658	0.721
Basic features + fuzzy features (fs1 + fs2)	0.660	0.738
Basic features + fuzzy features + w2v features (fs1 + fs2 + fs4)	0.676	0.766
W2v vector features (fs5)	x	0.78
Basic features + fuzzy features + w2v features + w2v vector features (fs1 + fs2 + fs4 + fs5)	**x**	**0.814**
TFIDF-SVD features (fs3-1)	0.777	0.749
TFIDF-SVD features (fs3-2)	0.804	0.748
TFIDF-SVD features (fs3-3)	0.706	0.763
TFIDF-SVD features (fs3-4)	0.700	0.753
TFIDF-SVD features (fs3-5)	0.714	0.759

x = I didn't bother training these models.

Deep learning to the rescue



LSTM

- Long short term memory
- A type of RNN
- Learn long term dependencies
- Used two LSTM layers

1-D CNN

- One dimensional convolutional layer
- Temporal convolution
- Simple to implement:

```
for i in range(sample_length):
    y[i] = 0
    for j in range(kernel_length):
        y[i] += x[i-j] * h[j]
```

Embedding layers

- Simple layer
- Converts indexes to vectors
- [[4], [20]] -> [[0.25, 0.1], [0.6, -0.2]]

Time distributed dense layer

- TimeDistributed wrapper around dense layer
- TimeDistributed applies the layer to every temporal slice of input
- Followed by Lambda layer
- Implements “translation” layer used by Stephen Merity (keras snli model)

```
model1 = Sequential()
model1.add(Embedding(len(word_index) + 1,
                     300,
                     weights=[embedding_matrix],
                     input_length=40,
                     trainable=False))
model1.add(TimeDistributed(Dense(300, activation='relu')))
model1.add(Lambda(lambda x: K.sum(x, axis=1), output_shape=(300,)))
```

GloVe Embeddings

- Count based model
- Dimensionality reduction on co-occurrence counts matrix
- word-context matrix -> word-feature matrix
- Common Crawl
- 840B tokens, 2.2M vocab, 300d vectors

Jeffrey Pennington, Richard Socher, and Christopher D. Manning. 2014. GloVe: Global Vectors for Word Representation

Handling text data before training

- Tokenize data
- Convert text data to sequences

```
tk = text.Tokenizer(nb_words=200000)
max_len = 40
tk.fit_on_texts(list(data.question1.values) + list(data.question2.values.astype(str)))
x1 = tk.texts_to_sequences(data.question1.values)
x1 = sequence.pad_sequences(x1, maxlen=max_len)
x2 = tk.texts_to_sequences(data.question2.values.astype(str))
x2 = sequence.pad_sequences(x2, maxlen=max_len)
word_index = tk.word_index
```

Handling text data before training

➢ Initialize GloVe embeddings

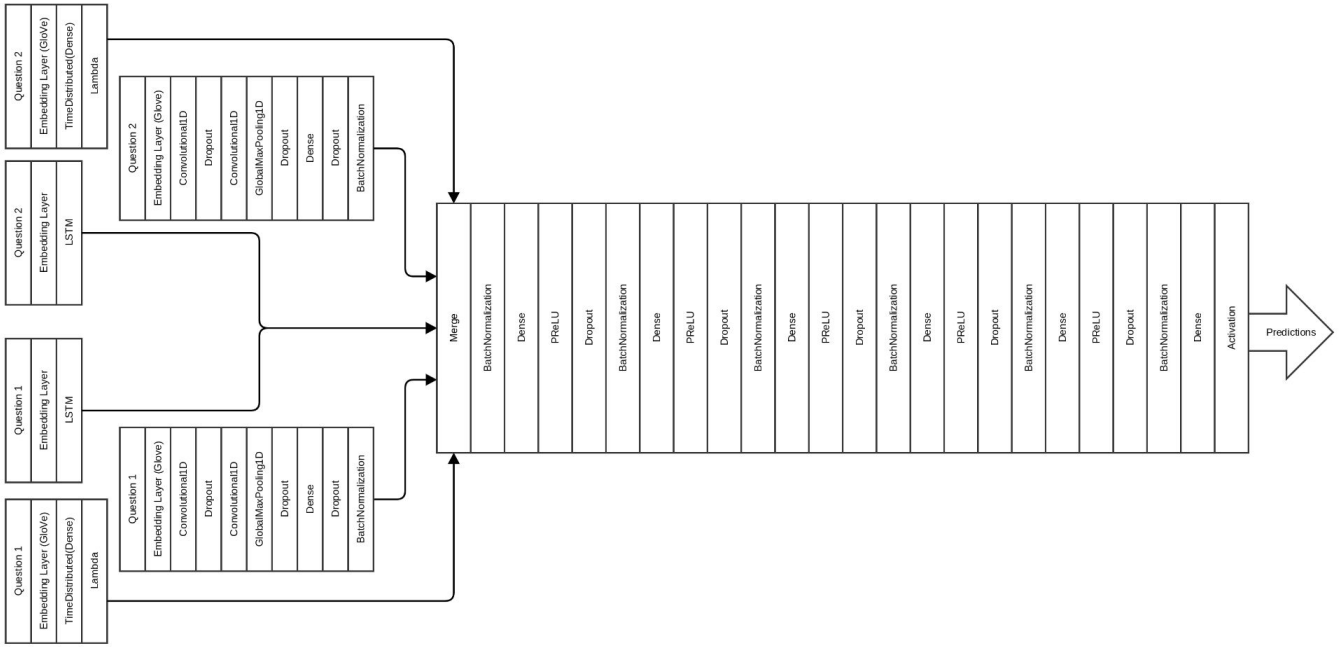
```
embeddings_index = {}
f = open('glove.840B.300d.txt')
for line in tqdm(f):
    values = line.split()
    word = values[0]
    coefs = np.asarray(values[1:], dtype='float32')
    embeddings_index[word] = coefs
f.close()
```

Handling text data before training

- Create the embedding matrix

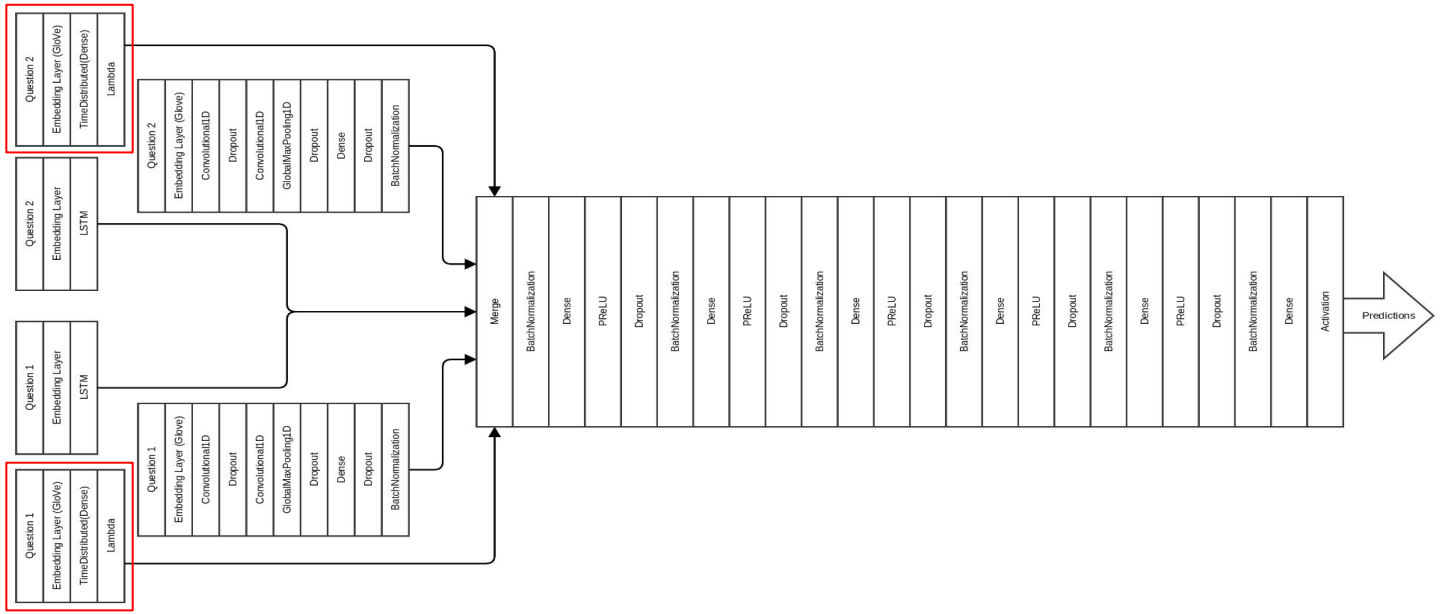
```
embedding_matrix = np.zeros((len(word_index) + 1, 300))
for word, i in tqdm(word_index.items()):
    embedding_vector = embeddings_index.get(word)
    if embedding_vector is not None:
        embedding_matrix[i] = embedding_vector
```

Final deep learning model

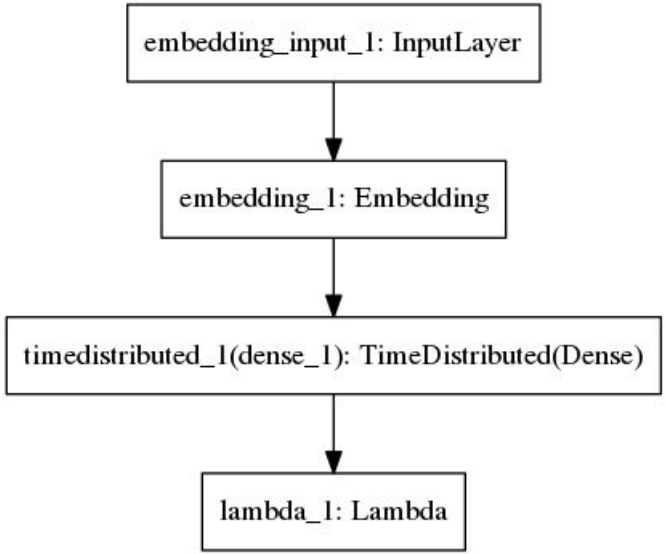


Final deep learning model

Deep learning model

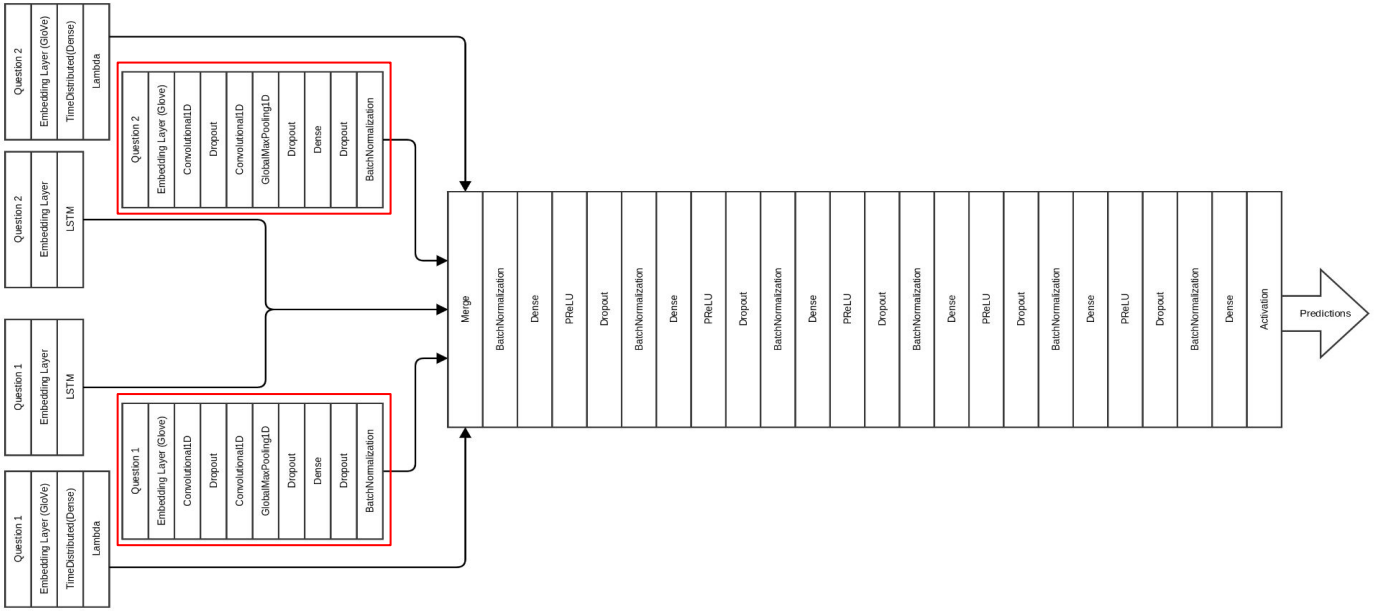


Deep learning model

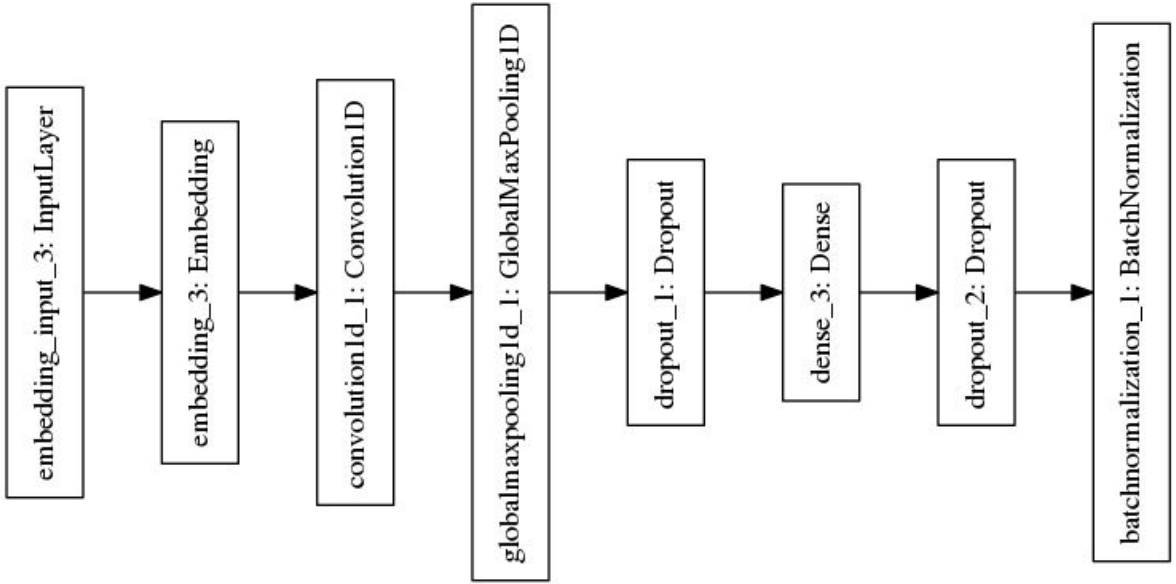



```
model1 = Sequential()
model1.add(Embedding(len(word_index) + 1,
                     300,
                     weights=[embedding_matrix],
                     input_length=40,
                     trainable=False))

model1.add(TimeDistributed(Dense(300, activation='relu')))
model1.add(Lambda(lambda x: K.sum(x, axis=1),
           output_shape=(300,)))
```

Deep learning model



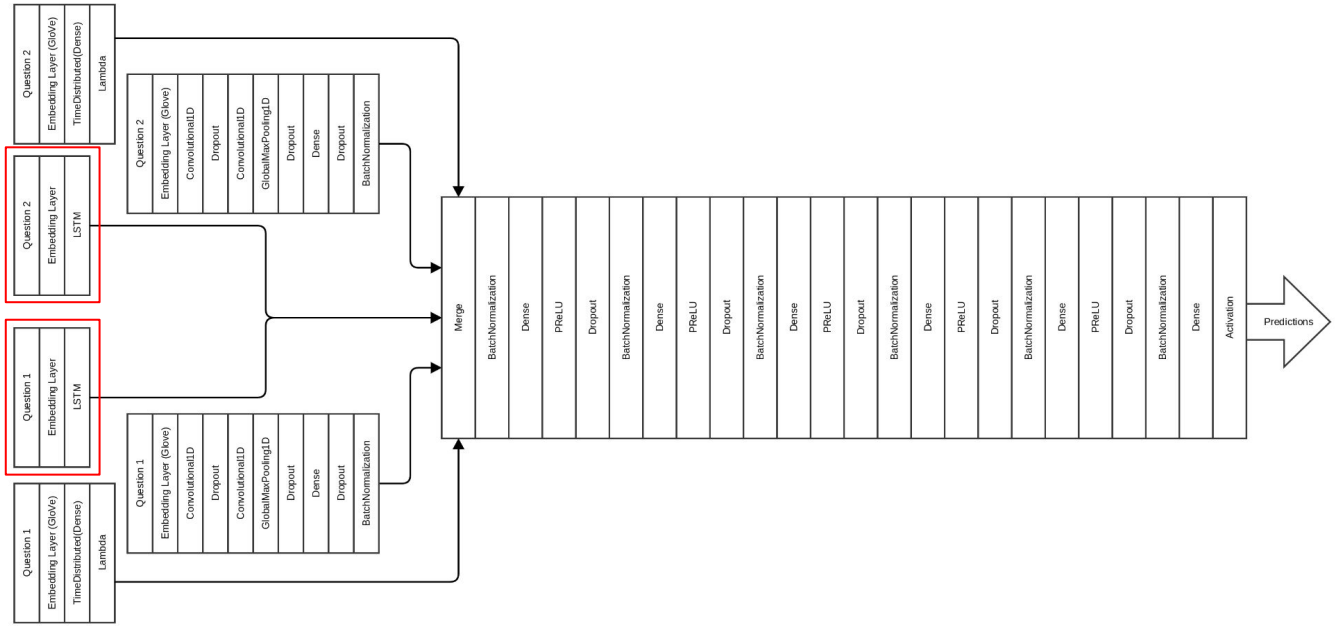
Deep learning model



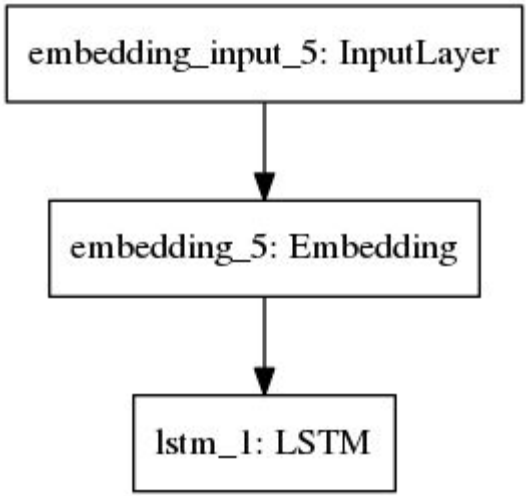
Deep learning model

```
model3 = Sequential()
model3.add(Embedding(len(word_index) + 1,
                     300,
                     weights=[embedding_matrix],
                     input_length=40,
                     trainable=False))
model3.add(Convolution1D(nb_filter=nb_filter,
                         filter_length=filter_length,
                         border_mode='valid',
                         activation='relu',
                         subsample_length=1))
model3.add(Dropout(0.2))
.
.
.
model3.add(Dense(300))
model3.add(Dropout(0.2))
model3.add(BatchNormalization())
```

Deep learning model

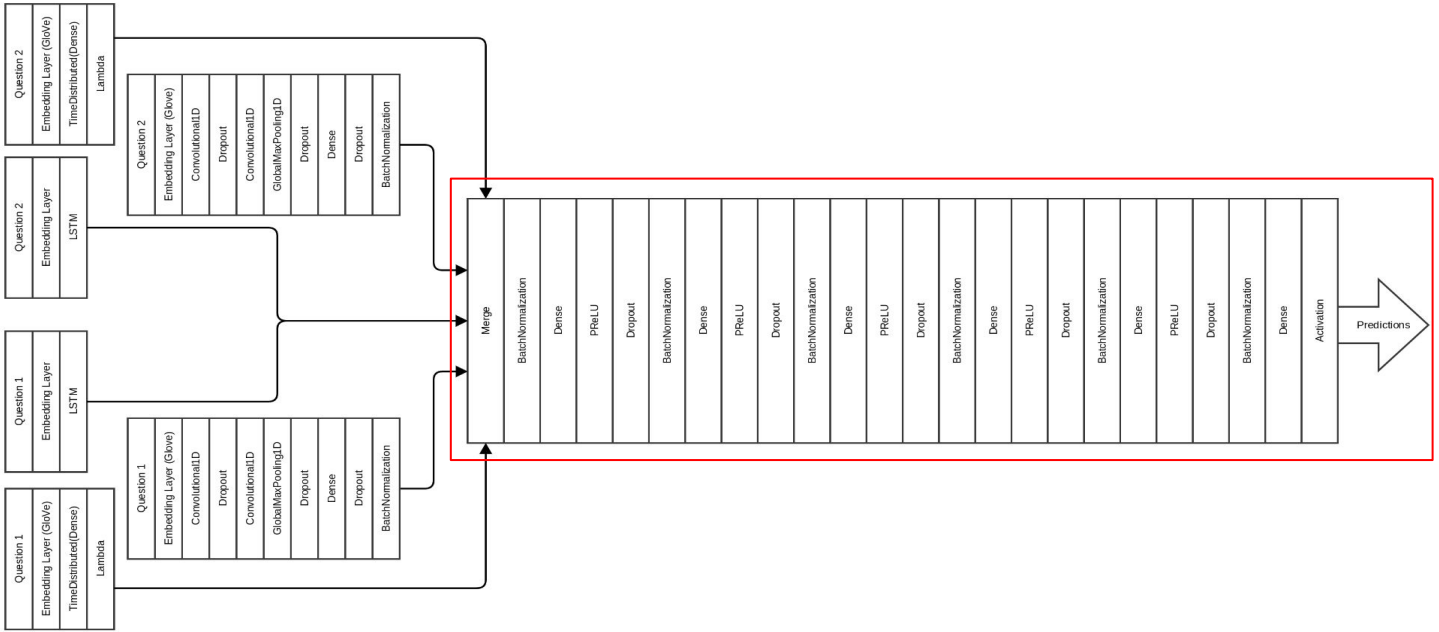


Deep learning model

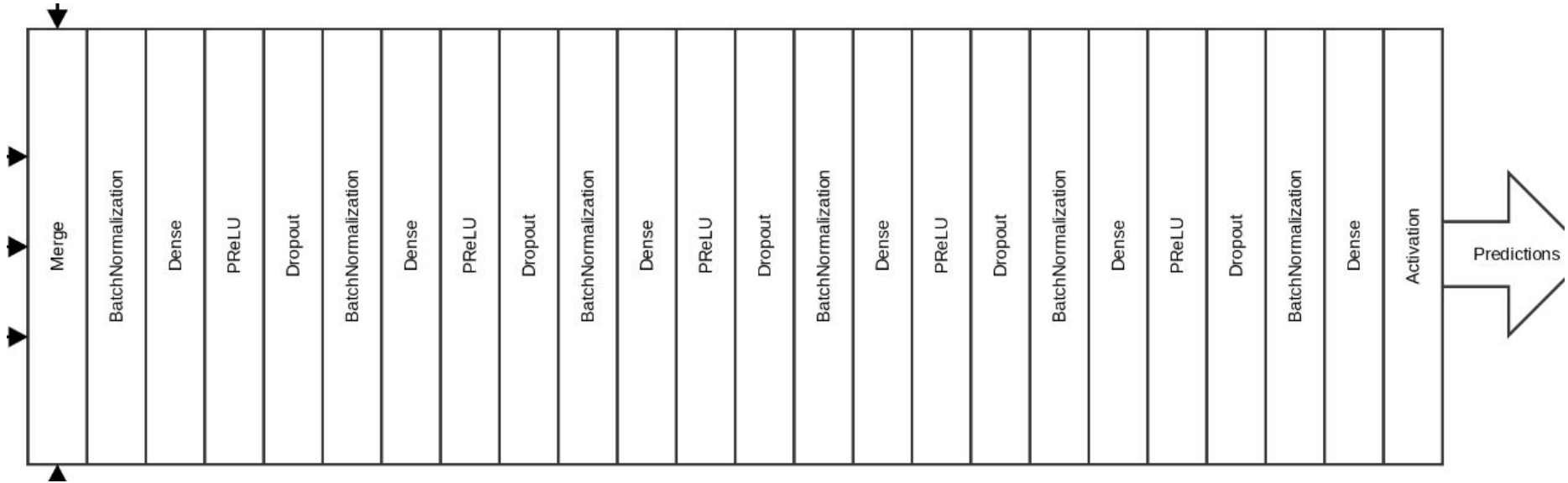



```
model5 = Sequential()
model5.add(Embedding(len(word_index) + 1, 300,
input_length=40,
                     dropout=0.2))
model5.add(LSTM(300, dropout_W=0.2, dropout_U=0.2))

model6 = Sequential()
model6.add(Embedding(len(word_index) + 1, 300,
input_length=40,
                     dropout=0.2))
model6.add(LSTM(300, dropout_W=0.2, dropout_U=0.2))
```

Deep learning model



Deep learning model



Time to train the model

- Total params: 174,913,917
- Trainable params: 60,172,917
- Non-trainable params: 114,741,000

- NVIDIA Titan X


JUST WAITING ON MY
GIRLFRIEND TO GET READY
quickmeme.com

Final results

Feature Set	Logistic Regression Accuracy	Xgboost Accuracy
Basic features (fs1)	0.658	0.721
Basic features + fuzzy features (fs1 + fs2)	0.660	0.738
Basic features + fuzzy features + w2v features (fs1 + fs2 + fs4)	0.676	0.766
W2v vector features (fs5)	x	0.78
Basic features + fuzzy features + w2v features + w2v vector features (fs1 + fs2 + fs4 + fs5)	**x**	**0.814**
TFIDF-SVD features (fs3-1)	0.777	0.749
TFIDF-SVD features (fs3-2)	0.804	0.748
TFIDF-SVD features (fs3-3)	0.706	0.763
TFIDF-SVD features (fs3-4)	0.700	0.753
TFIDF-SVD features (fs3-5)	0.714	0.759

x = I didn't bother training these models.

The deep network was trained on an NVIDIA TitanX and took approximately 300 seconds for each epoch and took 10-15 hours to train. **This network achieved an accuracy of 0.848 (~0.85).**

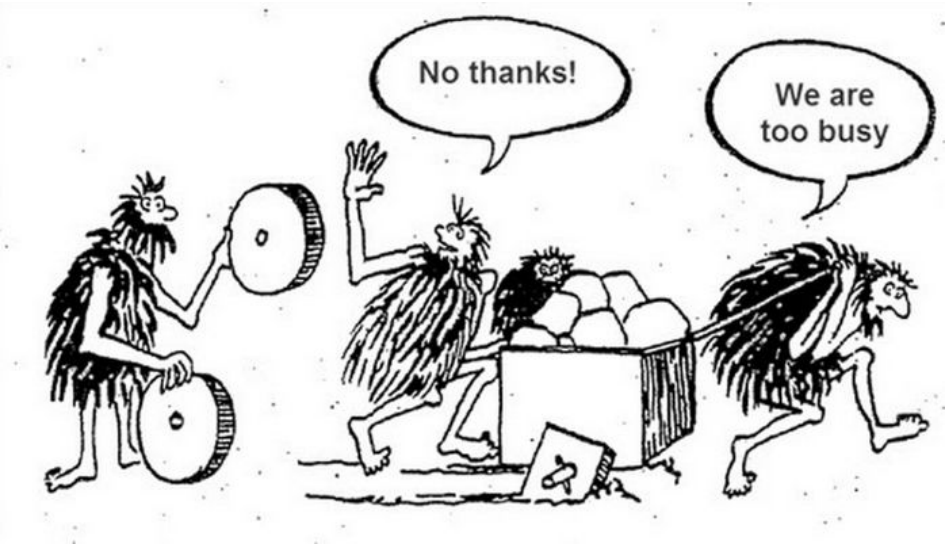
Final results

- The deepnet gives near state-of-the-art result
- BiMPM model accuracy: 88%

Be more creative with text data

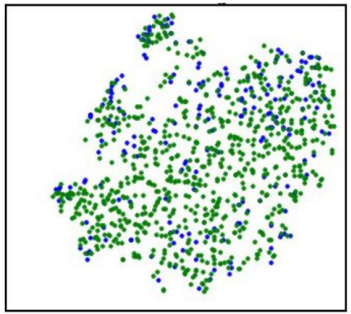
1. Language detection
2. Clean up
3. Tokenization
4. Stemming
5. Synonyms & stop words
6. Spell correction
7. Compound splitting
8. Entity recognition

Fine-tuning often gives good results

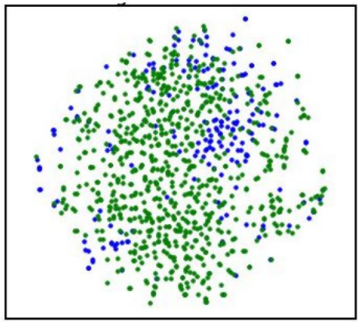


- It is faster
- It is better (not always)
- Why reinvent the wheel?

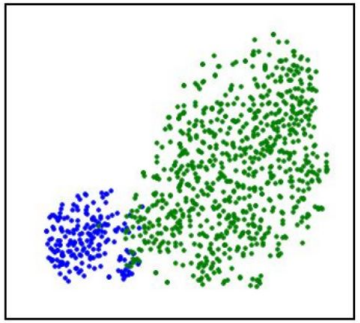
Fine-tuning often gives good results



Random Weights



Pre-Trained



Fine-Tuned

Hyper-parameter tuning

- Grid search
- Random search
- Bayesian optimization
- Optimize by hand

Hyper-parameter tuning using scikit-opt

```
# Initializing a CatBoostClassifier
clf = CatBoostClassifier(thread_count=2,
                         loss_function='Logloss',
                         verbose = False)
```

```
# Defining your search space
search_spaces = {'iterations': Integer(10, 300),
                 'depth': Integer(1, 8),
                 'learning_rate': Real(0.01, 1.0, 'log-uniform'),
                 'random_strength': Real(1e-9, 10, 'log-uniform'),
                 'bagging_temperature': Real(0.0, 1.0),
                 'border_count': Integer(1, 255),
                 'l2_leaf_reg': Integer(2, 30),
                 'scale_pos_weight':Real(0.01, 1.0, 'uniform')}
```

https://github.com/lmassaron/kaggledays-2019-gbdt

Hyper-parameter tuning using scikit-opt

```
# Setting up BayesSearchCV
opt = BayesSearchCV(clf,
                    search_spaces,
                    scoring=roc_auc,
                    cv=skf,
                    n_iter=100,
                    n_jobs=1,  # use just 1 job with CatBoost in order to avoid segmentation fault
                    return_train_score=False,
                    refit=True,
                    optimizer_kwargs={'base_estimator': 'GP'},
                    random_state=42)
```

https://github.com/lmassaron/kaggledays-2019-gbdt

Hyper-parameter tuning by hand or grid

Model	Parameters to optimize	Good range of values
Linear Regression	• fit_intercept • normalize	• True / False • True / False
Ridge	• alpha • Fit_intercept • Normalize	• 0.01, 0.1, 1.0, 10, 100 • True/False • True/False
k-neighbors	• N_neighbors • p	• 2, 4, 8, 16 …. • 2, 3
SVM	• C • Gamma • class_weight	• 0.001, 0.01…..10...100...1000 • 'Auto', RS* • 'Balanced' , None
Logistic Regression	• Penalty • C	• L1 or l2 • 0.001, 0.01…..10...100
Naive Bayes (all variations)	NONE	NONE
Lasso	• Alpha • Normalize	• 0.1, 1.0, 10 • True/False
Random Forest	• N_estimators • Max_depth • Min_samples_split • Min_samples_leaf • Max features	• 120, 300, 500, 800, 1200 • 5, 8, 15, 25, 30, None • 1, 2, 5, 10, 15, 100 • 1, 2, 5, 10 • Log2, sqrt, None
Xgboost	• Eta • Gamma • Max_depth • Min_child_weight • Subsample • Colsample_bytree • Lambda • alpha	• 0.01,0.015, 0.025, 0.05, 0.1 • 0.05-0.1,0.3,0.5,0.7,0.9,1.0 • 3, 5, 7, 9, 12, 15, 17, 25 • 1, 3, 5, 7 • 0.6, 0.7, 0.8, 0.9, 1.0 • 0.6, 0.7, 0.8, 0.9, 1.0 • 0.01-0.1, 1.0 , RS* • 0, 0.1, 0.5, 1.0 RS*

Sugar

Understanding the data

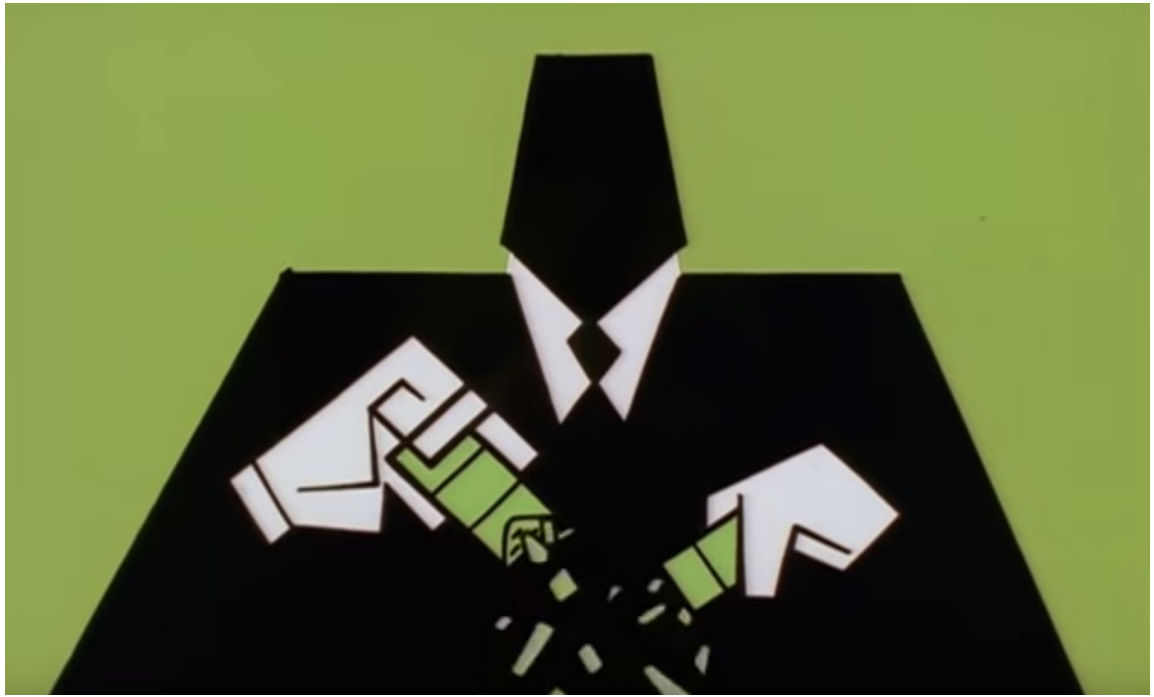
Exploring the data

Spice

Pre-processing

Feature engineering

Feature selection



All the things that are nice

A good cross validation

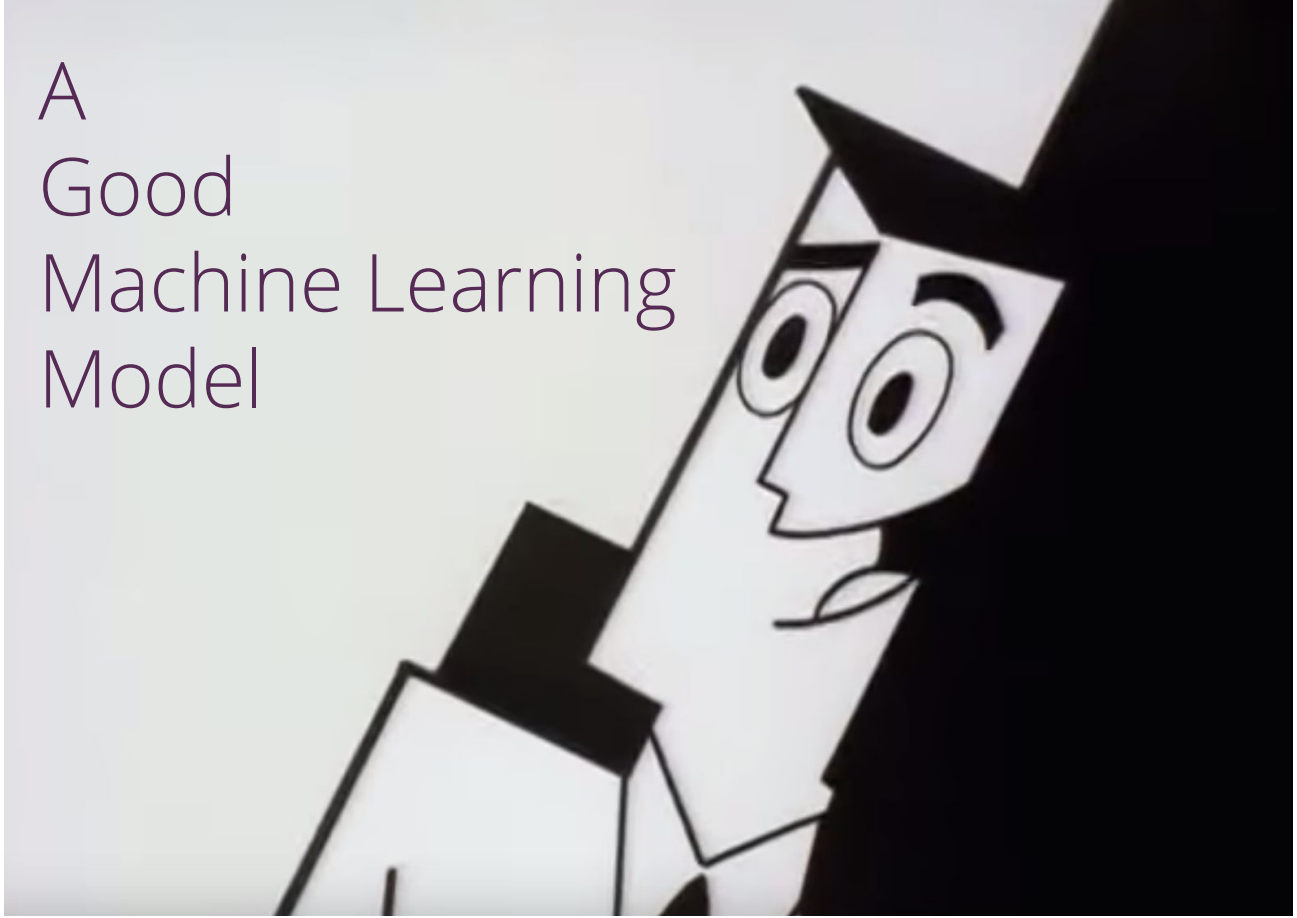
Low Error Rate

Simple or combination of models

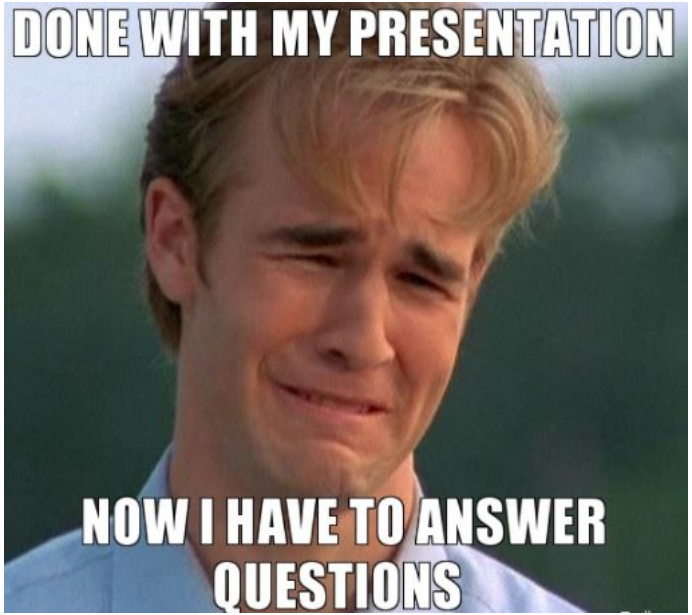
Post-processing

Chemical X

A Good Machine Learning Model



Thank you



- e-mail: abhishek4@gmail.com
- linkedin: bit.ly/thakurabhishek
- kaggle: kaggle.com/abhishek
- tweet me: @abhi1thakur

If everything fails, use xgboost!