

Entity Embeddings of Categorical Variables

Subject

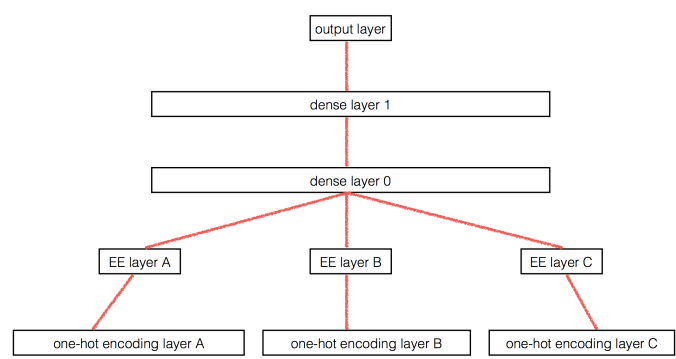
3rd place kaggle solution  
Rossmann sales prediction

very simple model

Discussed in fast.ai course

<https://www.youtube.com/watch?v=1-NYPQw5THU>

@7:30min



No RNN

Use emeddings

feature	data type	number of values	EE dimension
store	nominal	1115	10
day of week	ordinal	7	6
day	ordinal	31	10
month	ordinal	12	6
year	ordinal	3 (2013-2015)	2
promotion	binary	2	1
state	nominal	12	6

Dense layer

two layers    1000 and 500 neurons

no dropout    did not improve result

ReLU activation

Output layer one neuron with sigmoid activation

Time series

Mean absolute percentage error  
MAPE

$$M = \frac{100}{n} \sum_{t=1}^n \left| \frac{A_t - F_t}{A_t} \right|,$$

where  $A_t$  is the actual value and  $F_t$  is the forecast value.

Results

xgboost	
max_depth	10
eta	0.02
objective	reg:linear
colsample_bytree	0.7
subsample	0.7
num_round	3000

random forest	
n_estimators	200
max_depth	35
min_samples_split	2
min_samples_leaf	1

KNN	
n_neighbors	10
weights	distance
p	1

TABLE II. Parameters of models used to compare with neural networks. If a parameter is not specified, the default choice of scikit-learn (for random forests and KNN) and xgboost was taken.

method	MAPE	MAPE (with EE)
KNN	0.290	0.116
random forest	0.158	0.108
gradient boosted trees	0.152	0.115
neural network	0.101	0.093

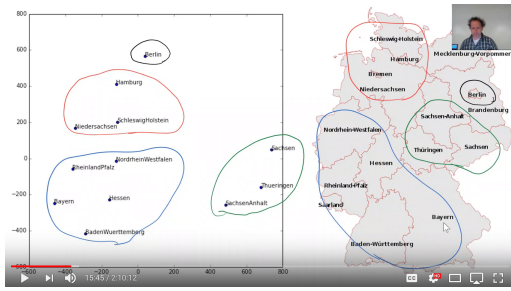
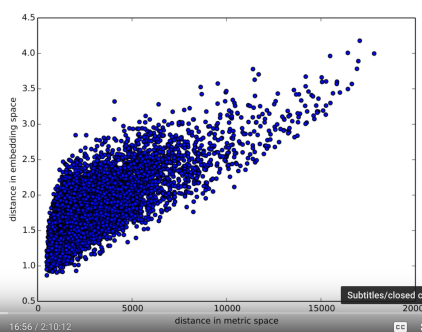
Comparing NN with trees and KNN

Projections of embeddings

t-Distributed Stochastic Neighbor Embedding

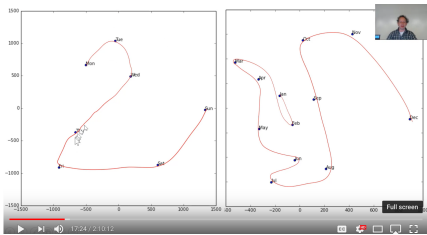
<https://lvdmaaten.github.io/tsne/>

t-SNE



state

metric distance to embeddings space distance



day of the week  
and month

<https://github.com/fastai/courses/tree/master/deeplearning2>

```
def add_datepart(df):
    df.Date = pd.to_datetime(df.Date)
    df["Year"] = df.Date.dt.year
    df["Month"] = df.Date.dt.month
    df["Week"] = df.Date.dt.week
    df["Day"] = df.Date.dt.day

add_datepart(weather)
add_datepart(googletrend)
add_datepart(train)
```

always create the following features

```
columns = ["Date", "Store", "Promo", "StateHoliday", "StateHoliday"]

class XgbModel(object):
    def __init__(self, F10):
        self.F10 = F10
        self.Xgb = xgb.XGBModel()
        self.Xgb.fit(X, y)

    def predict(self, F10):
        if F10 is None:
            return self.Xgb.predict(X)
        else:
            return self.Xgb.predict(X + F10)
```

create durations, i.e duration since last holiday and until next holiday

working code for tf 1.4