Machine Learning Engineer Nanodegree

Capstone Project

Adarsha Badarinath

April 2017

NOTE: A capstone proposal has already been accepted. Here are the links to proposal and the review of the proposal. A copy of the Kaggle data can be found here for the purpose of verification in-case the original data changes. A few of the below aspects are already described in the proposal.

- I. Definition
    - Project Overview
    - Problem Statement
    - Metrics
- II. Analysis
    - Data Exploration
    - Exploratory Visualization
    - Algorithms and Techniques
    - Benchmark

Decision Trees

This modal helps in choosing the feature with the highest information and making a branch. It does it again with the remaining features until we run-out-of features or a specific depth. A decision tree has only burst nodes (splitting paths) but no sink nodes (converging paths).

There are several parameters/tuning options.

- Which “feature” do we expose to the tree in the data
- How is “information gain” calculated
- What’s the max depth allowed

sklearn.model_selection.GridSearchCV - provides a lot more parameters and fine tuning options.

Decision trees have a tendency to overfit to the data. And *hence we will be using* ***Random Forest***. Which is an ensemble of decision trees.

Support Vector Machines

Support vector machines are machine learning algorithms which draw boundaries in N dimensional data and try to maximize this boundary gap to give the best separation.

They can be used in both supervised and unsupervised learning scenarios. For classification technique like ours it is a very good algorithm to try.

Neural Networks

They are modeled based on neurons of brain, which take multiple inputs and combines them with a probabilistic dynamic weight and sends a single output. They primarily produce a regression output but can also be used for classification purposes. The real advantage is in when we “back propagate” during training. We let the network predict based on random values at first and then go back and tune the weights based on how far off was it from the actual value. Networks can be constructed in many forms by combining many “neurons” both in depth and width.

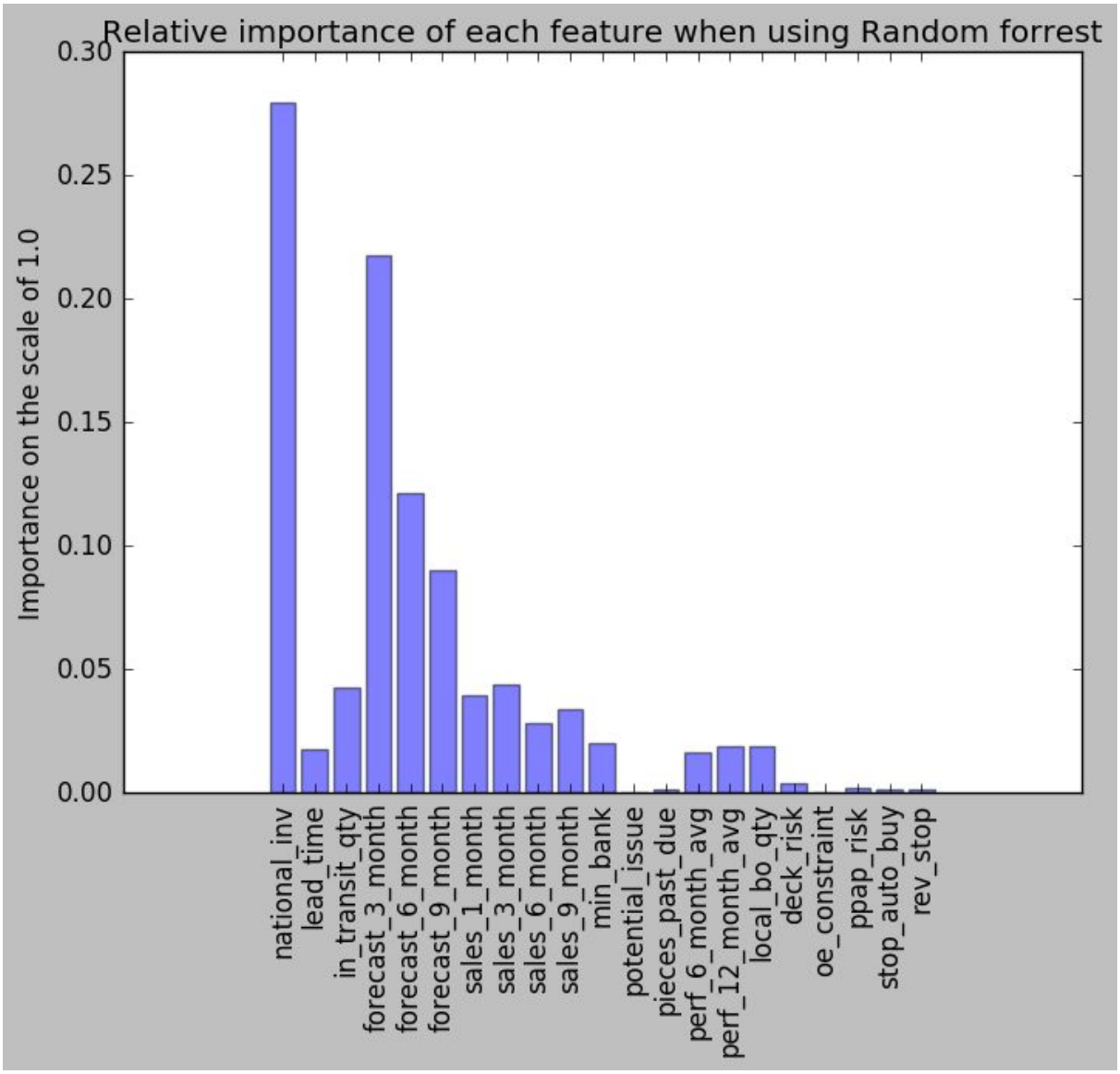
III. Methodology

Data Preprocessing

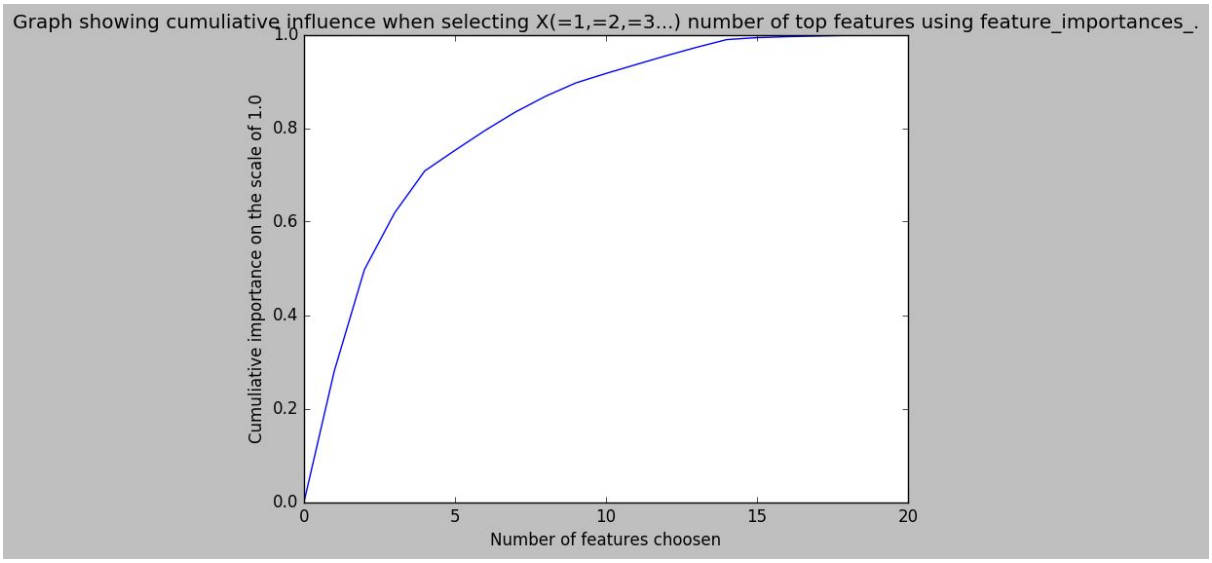
1. Data is in form CSV needs to be imported.
2. Data was originally cleaned to not contain any null/empty values. No such row was found as the kaggle data was pretty well cleaned up already.
3. Certain boolean columns need to be manually converted to a 0/1 integer value from ‘Yes’/’No’ String
4. Train and test split was done and we have different CSV files written into disk.
5. We split into X , y by dropping ‘went_on_backorder’ column for X and storing this column in ‘y’ a.k.a our desired prediction

Preprocessing for Decision Trees model

- Did no preprocessing of data for and supplied the full X,y for fit() function
- The MAX_DEPTH parameter feature was tuned with the help of RandomForestClassifier estimator using feature_importances_ array. We find that relative importance of each feature is as below.



When we produce a graph showing cumulative influences of features, we find that after the first 15 best features the others have very small influences. This helped us choose the parameter to grid search as either [10,15] for max depth



Preprocessing for Support Vector Machines model

Apart from the standard preprocessing explained above, no special preprocessing is needed for this algorithm. We however do grid search to find best parameters which is explained later.

Preprocessing for Neural Networks model

Neural networks need to have their input regularized to avoid large variations. Thus we preprocess the data in the cells to be >= 0 and <= 1.

Neural Networks need more balanced data. Hence I dropped 80% of class=0 .. i.e where backorder = NO. This enabled a more balanced dataset where the classes are more evenly distributed. Below is the code which does it

```
for i, row in X_regularized.iterrows():
    if y_copy[i] == 0 and randint(0,10000) < 8000:
        rows_to_drop.append(i)

X_regularized.drop(rows_to_drop,inplace=True)
y_copy.drop(rows_to_drop,inplace=True)
```

Implementation and Refinement

Decision Trees

I have implemented both DecisionTreeClassifier & RandomForestClassifier (which is an ensemble of decision trees). Both required normal preprocessing explained before. Analysis was made to identify top features as shown before.

To find the best model I applied GridSearchCV to find the best parameters. In both cases StratifiedKFold split was used to make sure we are not tuning to the training data.

Tuned parameters for DecisionTreeClassifier

{'max_depth':[10,15]}

Tuned parameters for RandomForestClassifier

[{'n_estimators': [100,200],

'criterion': ['entropy','gini'],

'max_depth':[10,15]}]

For RandomForestClassifier I also hyper-tuned over score ['precision', 'f1'].

```
cv = StratifiedKFold(n_splits=5)
clf = GridSearchCV(DecisionTreeClassifier(random_state=42),
tuned_parameters, cv=cv,verbose=10, n_jobs=4)
clf.fit(X, y)
```

```
cv = StratifiedKFold(n_splits=5)clf =
GridSearchCV(RandomForestClassifier(class_weight='balanced'),
tuned_parameters, cv=cv,scoring='%s_macro' % score, verbose=10,
n_jobs=4)    clf.fit(X, y)
```

Support Vector Machines

I tried first a normal SVC() classifier from sklearn. But then I found that it is not well suited for my dataset. From sklearn documentation it says “The fit time complexity is more than quadratic with the number of samples which makes it hard to scale to dataset with more than a couple of 10000 samples.”.

Thus I used a LinearSVC which scales well with more data. My input were for the GridSearchCV was as follows

```
tuned_parameters = [{'C' : [0.5,1.0,1.5],
                     'loss' : ['hinge','squared_hinge']}]
scores = ['precision', 'f1']
```

I choose the above tuned_parameters on a few experimentation. I then choose to choose the “scoring” function as ‘precision’ and ‘f1’ as those are my most important parameter to tune to.

```
cv = StratifiedKFold(n_splits=4)
clf = GridSearchCV(LinearSVC(random_state=42), tuned_parameters,
scoring='%s_macro' % score, cv=cv, verbose=10, n_jobs=8)
clf.fit(X, y)
```

Neural Networks

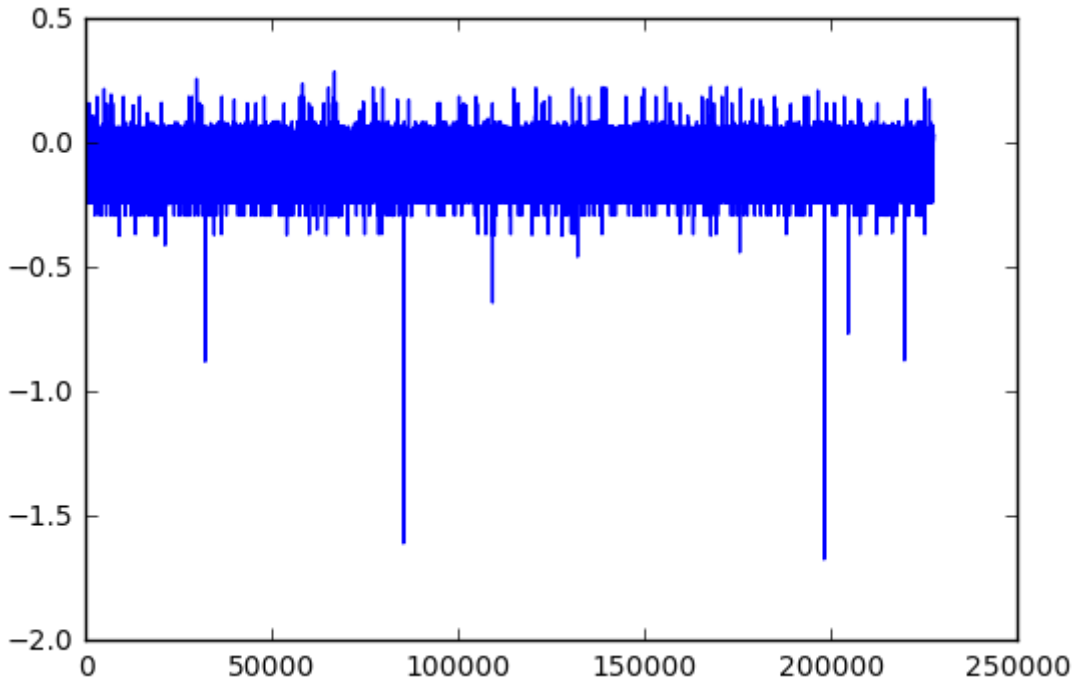
I used keras front end for Neural Networks to run on a TensorFlow backend on AWS graphics card for fast iteration and experimentation.

Below is my model summary

```
Layer (type)                 Output Shape              Param #     Connected to
=================================================================
dense_51 (Dense)             (None, 21)                462         dense_input_17[0][0]
_________________________________________________________________
activation_34 (Activation)   (None, 21)                0           dense_51[0][0]
_________________________________________________________________
dense_52 (Dense)             (None, 14)                308         activation_34[0][0]
_________________________________________________________________
activation_35 (Activation)   (None, 14)                0           dense_52[0][0]
_________________________________________________________________
dense_53 (Dense)             (None, 7)                 105         activation_35[0][0]
_________________________________________________________________
dense_54 (Dense)             (None, 1)                 8           dense_53[0][0]
=================================================================
Total params: 883
Trainable params: 883
Non-trainable params: 0
_________________________________________________________________
```

I trained it for 4 Epoch and batch_size=6400. Finally I predicted on the Regularized TEST data.

I plotted the predicted regularized output for classes. Since NN predict a value between 0 & 1, I had to draw this choose a value below/above which we predict backorder=true.



From above I arrived at

```
if (classes[i]) > 0.1 or classes[i] < -0.35:
    classes_pred[i] = 1
else:
    classes_pred[i] = 0
```

IV. Results

Model Evaluation and Validation and Justification

(Green - better than benchmark. Darker the better)

	Best params (if available)	**Precision**(0/1 & Total)		**F1 score** (0/1 & Total)	
Benchmark model					
		1.00 / 0.07	.99	0.94 / 0.13	.93

Decision Trees					
Decision Tree	{'max_depth': 10}	0.99 / 0.29	.98	0.99 / 0.03	.99
Random Forest (Precision tuned)	{'criterion': 'entropy', 'max_depth': 15, 'n_estimators': 100}	1.00 / 0.10	.99	0.97 / 0.18	.96
Random Forest (f1 tuned)	{'criterion': 'entropy', 'max_depth': 15, 'n_estimators': 200}	1.00 / 0.10	.99	0.97 / 0.18	.96
Support Vector Machines					
Precision tuned	{'loss': 'hinge', 'C': 1.0}	0.99 / 0.02	.98	0.99 / 0.00	.98
F1 tuned	{'loss': 'squared_hinge', 'C': 1.0}	0.99 / 0.05	.98	0.99 / 0.06	.98
Neural Network					
		0.99 / 0.05	.98	0.99 / 0.01	.98

I have explained why Precision is our main criteria followed by f1 score in the proposal.

The above numbers were generated through stratifiedKfold techniques where possible. This will make sure we do not optimize on the training data and get correct robust result results.

Looking at the results above, Random Forest are the clear winner. A simple decision tree with Max-depth = 10 is also better than the benchmark model. Random forest are better at avoiding overfitting than simple decision tree. Thus we see overall F1 and precision is better for random forest

NOTE: even tho for precision Random forest is 4x better at prediction of a product going into backorder, we see the f1 score is significantly down (3x down) due to overfitting

The final solution is **42% better precision and 38% better f1 score** than benchmark in predicting the product went on backorder (class = 1). Also for predicting

NOT-going-to-be-backordered there is no degradation in precision and 3% increase in f1 score.

Overall I would recommend the any company on benchmark models switch over to this modal to decrease the chance of products being not available for being on backorder.

V. Conclusion

Reflection

The solution to predicting any backorder is to

1. Take in the data for the 15 top features identified
2. Transform any Yes/No values to 1/0 respectively
3. Apply it the previously fit Random forest model with the below parameters
    a. {'criterion': 'entropy', 'max_depth': 15, 'n_estimators': 100}
4. If the prediction is 1, then send a backorder earlier for next week products.

A few interesting stuff I learned are

- We can reduce the feature set be extracting the top features. This not only increases performance due to less dimensionality but also sometimes increases the predictions
- I found that to apply normal SVM on a dataset more than 10k rows its quite difficult and we need to apply LinearSVM which scales better in python
- Neural networks are not well suited in this problem due to very high imbalance of classes. Also the data for backorder true (class=1) is very less
- It took a lot of time to do StatifiedKfold on a Grid search. Due to the many combinations. Thus I had to be more intelligent in choosing which parameter may affect my prediction
- I ran keras on the AWS to make use of the graphic card to run multiple iterations fast and experiment. I learned it in the Udacity self driving Car nano degree and it was useful here

The final solution is a satisfactory solution. This is because business is concerned about 2 things.

1. If they have backorder - this means customer is waiting and its not a good experience.
2. If they send too many inventory into the warehouse, it is wasting space.

Here we have good balance and our solution is better in prediction than benchmark. This can be improved over time after collecting data on how its performing if we fit the model again with new data.

Improvement

1. Our data is very good. I.e it has not many missing peices. However we can design a better system to identify what data should be filled in columns if it is corrupted or not available based on previous data. For example a simple way is to choose the median.
2. We need more data for class=1 where backorder is true. Out data is highly imbalanced and it is a challenge
3. If there are pr-existing solutions we can important from other domain I would try to find out. For example in medical field, we scan the eye to determine eye diseases. Here too the class is very imbalanced between thos who have a particular disease and those who dont. We can research and apply similar techniques to the wearhouse data.

Before submitting, ask yourself. . .

- Does the project report you've written follow a well-organized structure similar to that of the project template?
- Is each section (particularly Analysis and Methodology) written in a clear, concise and specific fashion? Are there any ambiguous terms or phrases that need clarification?
- Would the intended audience of your project be able to understand your analysis, methods, and results?
- Have you properly proof-read your project report to assure there are minimal grammatical and spelling mistakes?
- Are all the resources used for this project correctly cited and referenced?
- Is the code that implements your solution easily readable and properly commented?
- Does the code execute without error and produce results similar to those reported?