In [1]:
```python
# # Import Libraries
import numpy as np
import pandas as pd
import os
from sklearn.model_selection import PredefinedSplit
import sklearn_crfsuite
import eli5
import scipy
import re
import matplotlib.pyplot as plt
from sklearn.model_selection import learning_curve
from sklearn.model_selection import RandomizedSearchCV
from sklearn_crfsuite import metrics
from sklearn.metrics import make_scorer
from collections import Counter

import warnings
warnings.filterwarnings('ignore')
```

In [2]:
```python
# Change Working Directory
os.chdir('E:\data\Affinity_Solutions')
```

In [3]:
```python
# Read Input File
data_input = pd.read_csv('Summer Internship - Homework Exercise.csv')
```

In [4]:
```python
# Function to obtain train, validation and test sets
def split_train_test_val(split_type, data):
    extract_data = data_input[data_input['dataset'].str.contains((split_type.lower()))]
    extract_data.drop(['dataset'], axis = 1, inplace = True)
    extract_data.reset_index(drop=True, inplace=True)
    if extract_data.empty ==True:
        print("Split Type does not exist.")
        return -1
    else:
        return extract_data
```

In [5]:
```python
#Get Train Data
split_type = 'Train'
train_data = split_train_test_val(split_type,data_input)
```

In [6]:
```python
#Get Test Data
split_type = 'test'
test_data = split_train_test_val(split_type,data_input)
```

In [7]:
```python
#Get Validation Data
split_type = 'Val'
val_data = split_train_test_val(split_type,data_input)
```

In [8]:
```python
# Function to split the words by the digits in the sentence
def split_digit(on):
        oput  = re.findall('\d+',on)
        if len(oput)>0:
            oi = on.split(oput[0])
            return (oi, oput[0])
        else:
            return -1
```

In [9]:
```python
# Function to prepare data to get attributes by splitting with words, # and all digits in the sentence
def prepare_data(o_data):
    final = []

    for o in range( len(o_data)):
        otry = o_data['transaction_descriptor'][o]
        otest_0 = o_data['store_number'][o]

        otemp = []
        onew =  otry.split()

        for i in range(len(onew)):
            otemp.append(onew[i])

        otemp.append('@')
        otemp.append('@')
        otemp.append('@')

        for i in range(len(onew)):
            if onew[i].find('#')>-1:
                on = onew[i].split('#')

                for h in range(len(on)):
                    odigits = split_digit(on[h])

                    if (odigits)!=-1:
                        if len(odigits[0][0]) > 0 : otemp.append(odigits[0][0])
                        if len(odigits[1]) > 0 : otemp.append(odigits[1])
                        if len(odigits[0][1]) > 0 : otemp.append(odigits[0][1])

                    else:
                        if len(on[h]) > 0: otemp.append(on[h])
                        if h==0: otemp.append('#')
            else:

                odigits = split_digit(onew[i])
                if (odigits)!=-1:
                    if len(odigits[0][0]) > 0 : otemp.append(odigits[0][0])
                    if len(odigits[1]) > 0 : otemp.append(odigits[1])
                    if len(odigits[0][1]) > 0 : otemp.append(odigits[0][1])

                else:
                    if len(onew[i]) > 0:  otemp.append(onew[i])

        o_temp = []
        for j in range(len(otemp)):
            if (otemp[j].isdigit() == True and otest_0.isdigit() == True):
                if (int(otemp[j])==int(otest_0)):
                    o_temp.append((int(otemp[j]),'SN'))
                else:
                    o_temp.append((otemp[j],'NSN'))
            elif((otemp[j])==(otest_0)):
                o_temp.append(((otemp[j]),'SN'))
            else:
                o_temp.append((otemp[j],'NSN'))

        final.append(o_temp)

    return final
```

In [10]:
```python
# Prepare data with individual words for all sentences in train, validation and test data
otrain = prepare_data(train_data)
oval = prepare_data(val_data)
otest = prepare_data(test_data)
```

In [11]:
```python
# Derive features for each word
def oword2features(sent, i):

# Current word being extracted
    word = str(sent[i][0])

# Features of the current word
    features = {
        'word.upper()': word.upper(),
        'word[-4:]': word[-4:] if len(word)>3 else '-',
        'word[:4]': word[:4] if len(word)>3 else '-',
        'word[-3:]': word[-3:] if len(word)>2 else '-',
        'word[:3]': word[:3] if len(word)>2 else '-',
        'word[-2:]': word[-2:] if len(word)>1 else '-',
        'word[:2]': word[:2] if len(word)>1 else '-',
        'word[-1:]': word[-1:],
        'word[:1]': word[:1],
        'word[#]':  word.find('#'),
        'word.isdigit()': word.isdigit(),
        'word.hasdigit()': any(char.isdigit() for char in word),
        'word.hasalpha()': any(char.isalpha() for char in word),
    }

# Extracting features of the word one before the current word
    if i > 0:
        word1 = str(sent[i-1][0])
        features.update({
            '-1:word.upper()': word1.upper(),
            '-1:word[-4:]': word1[-4:] if len(word1)>3 else '-',
            '-1:word[:4]': word1[:4] if len(word1)>3 else '-',
            '-1:word[-3:]': word1[-3:] if len(word1)>2 else '-',
            '-1:word[:3]': word1[:3] if len(word1)>2 else '-',
            '-1:word[-2:]': word1[-2:] if len(word1)>1 else '-',
            '-1:word[:2]': word1[:2] if len(word1)>1 else '-',
            '-1:word[-1:]': word1[-1:],
            '-1:word[:1]': word1[:1],
            '-1:word[#]': word1.find('#'),
            '-1:word.isdigit()': word1.isdigit(),
            '-1:word.hasdigit()': any(char.isdigit() for char in word1),
            '-1:word.hasalpha()': any(char.isalpha() for char in word1),
        })

# Extracting features of the word two before the current word
    if i > 1:
        word2 = str(sent[i-2][0])
        features.update({
            '-2:word.lower()': word2.upper(),
            '-2:word[-4:]': word2[-4:] if len(word2)>3 else '-',
            '-2:word[:4]': word2[:4] if len(word2)>3 else '-',
            '-2:word[-3:]': word2[-3:] if len(word2)>2 else '-',
            '-2:word[:3]': word2[:3] if len(word2)>2 else '-',
            '-2:word[-2:]': word2[-2:] if len(word2)>1 else '-',
            '-2:word[:2]': word2[:2] if len(word2)>1 else '-',
            '-2:word[-1:]': word2[-1:] ,
            '-2:word[:1]': word2[:1] ,
            '-2:word[#]': word2.find('#'),
            '-2:word.isdigit()': word2.isdigit(),
            '-2:word.hasdigit()': any(char.isdigit() for char in word2),
            '-2:word.hasalpha()': any(char.isalpha() for char in word2),
        })

# Extracting features of the word three before the current word
    if i > 2:
        word3 = str(sent[i-3][0])
        features.update({
            '-3:word.upper()': word3.upper(),
            '-3:word[-4:]': word3[-4:] if len(word3)>3 else '-',
            '-3:word[:4]': word3[:4] if len(word3)>3 else '-',
            '-3:word[-3:]': word3[-3:] if len(word3)>2 else '-',
            '-3:word[:3]': word3[:3] if len(word3)>2 else '-',
            '-3:word[-2:]': word3[-2:] if len(word3)>1 else '-',
            '-3:word[:2]': word3[:2] if len(word3)>1 else '-',
            '-3:word[-1:]': word3[-1:] ,
            '-3:word[:1]': word3[:1] ,
            '-3:word[#]': word3.find('#'),
            '-3:word.isdigit()': word3.isdigit(),
            '-3:word.hasdigit()': any(char.isdigit() for char in word3),
            '-3:word.hasalpha()': any(char.isalpha() for char in word3),
        })

# Extracting features of the word one after the current word
    if i < len(sent)-1:
        word1 = str(sent[i+1][0])
        features.update({
            '+1:word.upper()': word1.upper(),
            '+1:word[-3:]': word1[-3:] if len(word1)>2 else '0',
            '+1:word[:2]': word1[:3] if len(word1)>2 else '0',
            '+1:word[-2:]': word1[-2:] if len(word1)>1 else '0',
            '+1:word[:2]': word1[:2] if len(word1)>1 else '0',
            '+1:word[-1:]': word1[-1:],
            '+1:word[:1]': word1[:1],
            '+1:word.isdigit()': word.isdigit(),
            '+1:word.hasdigit()': any(char.isdigit() for char in word1),
            '+1:word.hasalpha()': any(char.isalpha() for char in word1),
        })

    return features
```

In [12]:
```python
# Function to extract labels for the dependent variable(y)
def osent2labels(sent):
    return [label for token, label in sent]

# Function to extract features of each word from a sentence consisting the store number
def osent2features(sent):
    return [oword2features(sent, i) for i in range(len(sent))]
```

In [13]:
```python
# Preparing data for feature extraction by splitting with words with spaces, '#' and digits combination
otrain = prepare_data(train_data)
oval = prepare_data(val_data)
otest = prepare_data(test_data)
```

In [14]:
```python
# Extract features and labels for dependent and independent variables
X_train = [osent2features(s) for s in otrain]
y_train = [osent2labels(s) for s in otrain]

X_val= [osent2features(s) for s in oval]
y_val = [osent2labels(s) for s in oval]

X_test = [osent2features(s) for s in otest]
y_test = [osent2labels(s) for s in otest]
```

Training

In [15]:
```python
# Conditional Random Fields (CRF) model and training
crf = sklearn_crfsuite.CRF(
    algorithm='lbfgs',
    c1=0.2,
    c2=0.15,
    max_iterations=250,
    all_possible_transitions=True)

crf.fit(X_train, y_train)
```

Out[15]:
```
CRF(algorithm='lbfgs', all_possible_transitions=True, c1=0.2, c2=0.15,
    keep_tempfiles=None, max_iterations=250)
```

In [16]:
```python
# Defining possible labels from the dataset
labels = ['SN', 'NSN']
```

In [17]:
```python
# Predicting store number for the test data and comparing F-1 Score
y_pred = crf.predict(X_test)
print("The F-1 Score obtained from the test data: ", round(metrics.flat_f1_score(y_test, y_pred,
                      average='weighted', labels=labels), 3))
```

```
The F-1 Score obtained from the test data:  0.989
```

In [18]:
```python
#Sorting the labels and printing classification report
sorted_labels = sorted(labels, key=lambda name: (name[1:], name[0]))


print("\n The Classification Report obtained : \n", metrics.flat_classification_report(
    y_test, y_pred, labels=sorted_labels, digits=3))

print(metrics.flat_classification_report(y_test, y_pred, labels=sorted_labels, digits=3))
```

```
 The Classification Report obtained :
               precision    recall  f1-score   support

          SN      0.939     0.969     0.953       127
         NSN      0.996     0.992     0.994       955

    accuracy                          0.989      1082
   macro avg      0.967     0.980     0.974      1082
weighted avg      0.989     0.989     0.989      1082

              precision    recall  f1-score   support

          SN      0.939     0.969     0.953       127
         NSN      0.996     0.992     0.994       955

    accuracy                          0.989      1082
   macro avg      0.967     0.980     0.974      1082
weighted avg      0.989     0.989     0.989      1082
```

Randomized Search CV

In [19]:
```python
%%time
# Defining Parameters for search
crf = sklearn_crfsuite.CRF(
    algorithm='lbfgs',
    max_iterations=800,
    all_possible_transitions=True)

params_space = {
    'c1': scipy.stats.expon(scale=0.2),
    'c2': scipy.stats.expon(scale=0.3),
}

# Using F-1 metric
f1_scorer = make_scorer(metrics.flat_f1_score,
                        average='weighted', labels=labels)

# Generating Split index for Train and Validation data
split_index = [-1]*len(X_train) + [0]*len(X_val)

# Combining Train and Validation data with the pre-defined split as above
X = np.concatenate((X_train, X_val), axis=0)
y = np.concatenate((y_train, y_val), axis=0)
pds = PredefinedSplit(test_fold = split_index)

# Loading the model for training
rs = RandomizedSearchCV(crf, params_space,
                        cv=pds,
                        verbose=1,
                        n_jobs=-1,
                        n_iter=500,
                        scoring=f1_scorer)
# Training the model
rs.fit(X, y)
```

```
Fitting 1 folds for each of 500 candidates, totalling 500 fits
[Parallel(n_jobs=-1)]: Using backend LokyBackend with 16 concurrent workers.
[Parallel(n_jobs=-1)]: Done  18 tasks      | elapsed:    3.4s
[Parallel(n_jobs=-1)]: Done 168 tasks      | elapsed:    8.4s
[Parallel(n_jobs=-1)]: Done 418 tasks      | elapsed:   16.7s
[Parallel(n_jobs=-1)]: Done 500 out of 500 | elapsed:   19.5s finished
Wall time: 20.1 s
```

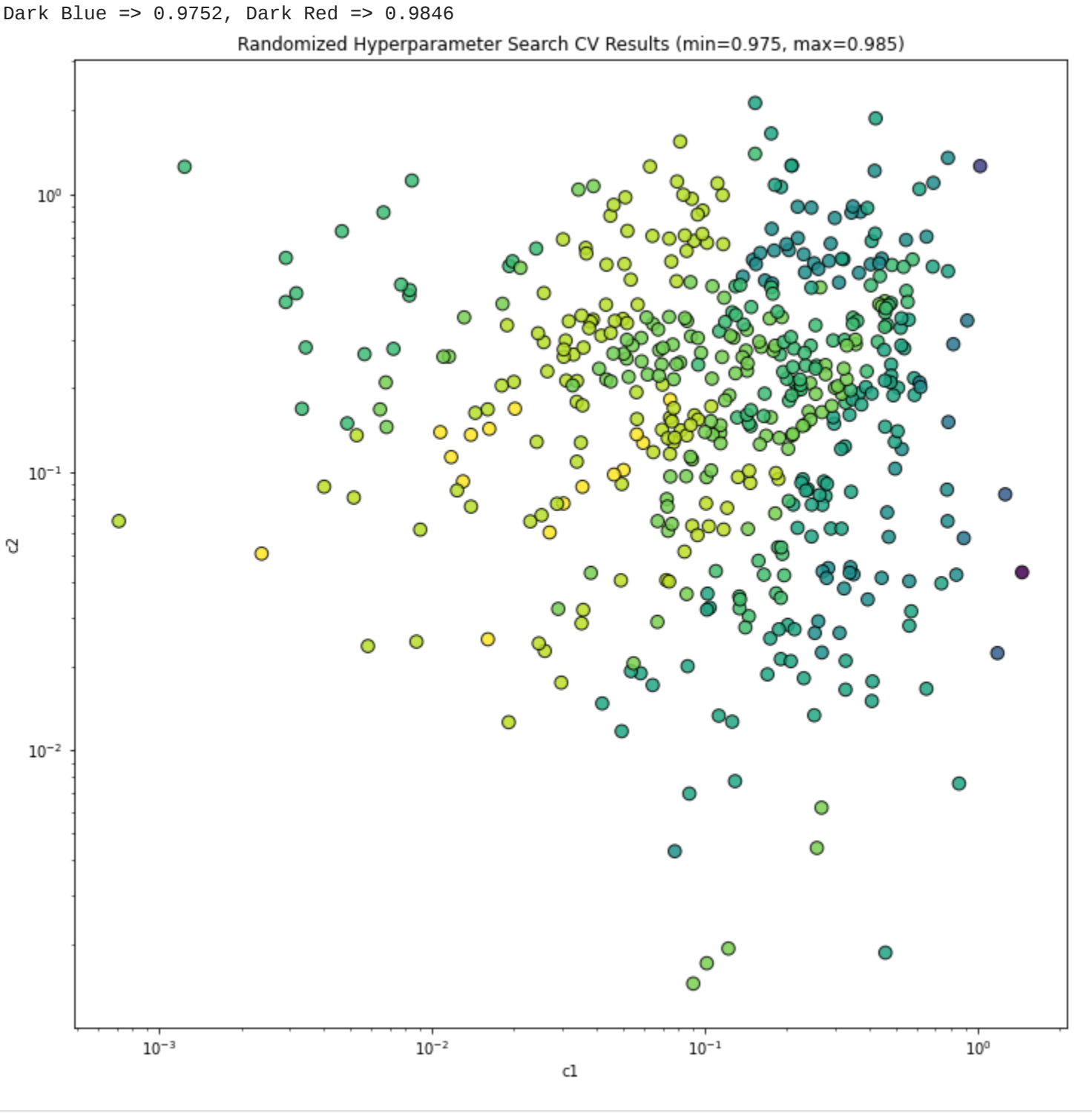
Out[19]:
```
RandomizedSearchCV(cv=PredefinedSplit(test_fold=array([-1, -1, ...,  0,  0])),
                   estimator=CRF(algorithm='lbfgs',
                                 all_possible_transitions=True,
                                 keep_tempfiles=None, max_iterations=800),
                   n_iter=500, n_jobs=-1,
                   param_distributions={'c1': <scipy.stats._distn_infrastructure.rv_frozen object at 0x000001B8C93E2F70>,
                                        'c2': <scipy.stats._distn_infrastructure.rv_frozen object at 0x000001B8C93D42B0>},
                   scoring=make_scorer(flat_f1_score, average=weighted, labels=['SN', 'NSN']),
                   verbose=1)
```

In [20]:
```python
# Prinnting Best Parameters from the randomized search with Best Score and Model Size
print('best params:', rs.best_params_)
print('best CV score:', rs.best_score_)
print('model size: {:0.2f}M'.format(rs.best_estimator_.size_ / 1000000))
```

```
best params: {'c1': 0.030370717954232088, 'c2': 0.07704766779555547}
best CV score: 0.9845533282086765
model size: 0.09M
```

In [21]:
```python
# Extracting and Plotting different c1 and c2 data with the mean test score
_x = rs.cv_results_['param_c1'].data
_y = rs.cv_results_['param_c2'].data
_c = rs.cv_results_['mean_test_score']

fig = plt.figure()
fig.set_size_inches(12, 12)
ax = plt.gca()
ax.set_yscale('log')
ax.set_xscale('log')
ax.set_xlabel('c1')
ax.set_ylabel('c2')
ax.set_title("Randomized Hyperparameter Search CV Results (min={:0.3}, max={:0.3})".format(min(_c), max(_c)))

ax.scatter(_x, _y, c=_c, s=75, alpha=0.85, edgecolors=[0,0,0])

print("Dark Blue => {:0.4}, Dark Red => {:0.4}".format(min(_c), max(_c)))
```

```
Dark Blue => 0.9752, Dark Red => 0.9846
```



In [22]:
```python
# Assigning Best Estimator
crf = rs.best_estimator_

y_pred = crf.predict(X_test)

print("The F-1 Score obtained from the test data = {:0.3}" .format(metrics.flat_f1_score(y_test, y_pred,
                      average='weighted', labels=labels)))

print("\n The Classification Report obtained : \n", metrics.flat_classification_report(
    y_test, y_pred, labels=sorted_labels, digits=3))
```

```
The F-1 Score obtained from the test data = 0.991

 The Classification Report obtained :
               precision    recall  f1-score   support

          SN      0.953     0.969     0.961       127
         NSN      0.996     0.994     0.995       955

    accuracy                          0.991      1082
   macro avg      0.975     0.981     0.978      1082
weighted avg      0.991     0.991     0.991      1082
```

In [23]:
```python
# Function to print the transaction
def print_transitions(trans_features):
    for (label_from, label_to), weight in trans_features:
        print("%-6s -> %-7s %0.3f" % (label_from, label_to, weight))
```

In [24]:
```python
# Print top 4 transition features
print("Top likely transitions:")
print_transitions(Counter(crf.transition_features_).most_common(4))
```

```
Top likely transitions:
NSN    -> NSN     1.235
NSN    -> SN      0.993
SN     -> NSN     -0.390
SN     -> SN      -1.732
```

In [25]:
```python
# Top 15 best and worst features affecting the model with the given training data
def print_state_features(state_features):
    for (attr, label), weight in state_features:
        print("%0.6f %-8s %s" % (weight, label, attr))
```

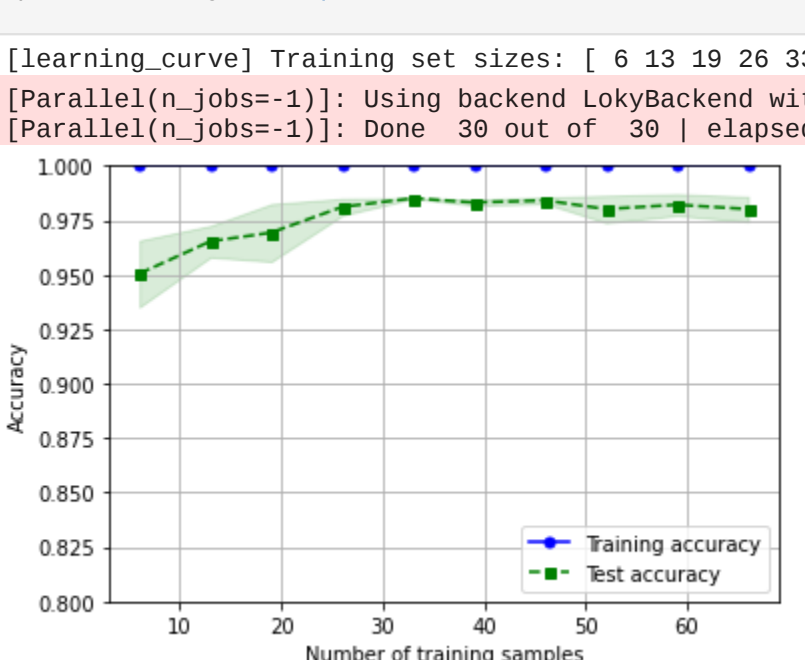
In [26]:
```python
# Print top and worst 15 features affecting the model
print("Top positive:")
print_state_features(Counter(crf.state_features_).most_common(15))

print("\nTop negative:")
print_state_features(Counter(crf.state_features_).most_common()[-15:])
```

```
Top positive:
2.626868 NSN      word[:1]:#
2.175729 SN       -1:word.upper():#
2.175729 SN       -1:word[-1:]:#
1.662193 NSN      -1:word.hasdigit()
1.638330 NSN      word[:1]:0
1.472217 SN       word.isdigit()
1.465853 SN       +1:word.isdigit()
1.099592 NSN      word.hasalpha()
1.079378 NSN      +1:word.hasalpha()
1.040465 NSN      -1:word[-1:]:F
1.032158 NSN      -1:word.upper():F
0.979662 NSN      word[:2]:00
0.952407 NSN      word[:3]:000
0.908008 SN       word[:1]:4
0.880891 SN       -1:word[-4:]:LD'S

Top negative:
-0.663869 SN       +1:word[-3:]:0
-0.669943 NSN      -1:word[-2:]:'S
-0.672881 SN       -2:word.hasdigit()
-0.739516 NSN      word[:1]:2
-0.748453 NSN      word.hasdigit()
-0.866742 NSN      -1:word[:1]:K
-0.878568 SN       +1:word.hasdigit()
-0.880891 NSN      -1:word[-4:]:LD'S
-0.880891 NSN      -1:word[-3:]:D'S
-0.908008 NSN      word[:1]:4
-1.079378 SN       +1:word.hasalpha()
-1.099592 SN       word.hasalpha()
-1.465853 NSN      +1:word.isdigit()
-1.472217 NSN      word.isdigit()
-1.662193 SN       -1:word.hasdigit()
```

In [27]:
```python
# Function to plot the learning  curve from the model

def plot_learning_curve(train_sizes, train_mean, train_std, test_mean, test_std):
    # Plot training accuracy means for a given series of training sizes
    plt.plot(train_sizes, train_mean, color="blue", marker="o", markersize=5, label="Training accuracy")
    # Add a coloured fill showing the standard deviation of the training accuracy for a given series of training sizes
    plt.fill_between(train_sizes, train_mean + train_std, train_mean - train_std, alpha=0.15, color="blue")
    # Plot test accuracy means for a given series of training sizes
    plt.plot(train_sizes, test_mean, color="green", linestyle="--", marker="s", markersize=5, label="Test accuracy")
    # Add a coloured fill showing the standard deviation of the test accuracy for a given series of training sizes
    plt.fill_between(train_sizes, test_mean + test_std, test_mean - test_std, alpha=0.15, color="green")
    # Add gridlines to the plot
    plt.grid()
    # Add captions to the X and Y axes of the plot
    plt.xlabel("Number of training samples")
    plt.ylabel("Accuracy")
    # Provide a location for the plot's legend/key
    plt.legend(loc="lower right")
    # Set upper and lower limits on the y axis
    plt.ylim([0.8, 1.0])
    # Show the plot
    plt.show()
```

In [28]:
```python
# Produce the necessary data for a learning curve
train_sizes, train_scores, test_scores = learning_curve(estimator=crf,
                                                        X=X_train,
                                                        y=y_train,
                                                        train_sizes=np.linspace(0.1, 1.0, 10),
                                                        cv=3,
                                                        verbose=1,
                                                        n_jobs=-1)

# Find the means and standard deviations of the training and test datasets across the learning curve
train_mean = np.mean(train_scores, axis=1)
train_std = np.std(train_scores, axis=1)
test_mean = np.mean(test_scores, axis=1)
test_std = np.std(test_scores, axis=1)

# Plot the learning curve
plot_learning_curve(train_sizes, train_mean, train_std, test_mean, test_std)
```

```
[learning_curve] Training set sizes: [ 6 13 19 26 33 39 46 52 59 66]
[Parallel(n_jobs=-1)]: Using backend LokyBackend with 16 concurrent workers.
[Parallel(n_jobs=-1)]: Done  30 out of  30 | elapsed:    0.4s finished
```



In [29]:
```python
#Explicit description of weights and likely transition
eli5.show_weights(crf, top=20)
```

Out[29]:

From \ To	NSN	SN
NSN	1.235	0.993
SN	-0.39	-1.732

y=NSN top features		y=SN top features	
Weight?	Feature	Weight?	Feature
+2.627	word[:1]:#	+2.176	-1:word[-1:]:#
+1.662	-1:word.hasdigit()	+2.176	-1:word.upper():#
+1.638	word[:1]:0	+1.472	word.isdigit()
+1.100	word.hasalpha()	+1.466	+1:word.isdigit()
+1.079	+1:word.hasalpha()	+0.908	word[:1]:4
+1.040	-1:word[-1:]:F	+0.881	-1:word[-4:]:LD'S
+1.032	-1:word.upper():F	+0.881	-1:word[-3:]:D'S
+0.980	word[:2]:00	+0.867	-1:word[:1]:K
+0.952	word[:3]:000	+0.748	word.hasdigit()
+0.879	+1:word.hasdigit()	+0.740	word[:1]:2
+0.695	+1:word[:1]:#	+0.670	-1:word[-2:]:'S
+0.673	-2:word.hasdigit()	+0.621	-1:word[:2]:F3
… 472 more positive …		+0.621	-1:word.upper():F35869MCDONALD'S
… 245 more negative …		… 480 more positive …	
-0.740	word[:1]:2	… 134 more negative …	
-0.748	word.hasdigit()	-0.654	-2:word.isdigit()
-0.867	-1:word[:1]:K	-0.664	+1:word[-3:]:0
-0.881	-1:word[-4:]:LD'S	-0.673	-2:word.hasdigit()
-0.881	-1:word[-3:]:D'S	-0.879	+1:word.hasdigit()
-0.908	word[:1]:4	-1.079	+1:word.hasalpha()
-1.466	+1:word.isdigit()	-1.100	word.hasalpha()
-1.472	word.isdigit()	-1.662	-1:word.hasdigit()

In [30]:
```python
#Extracting predictions of store number with index
out = []
sn_index = []
not_sn_index = []
for i in range(len(y_pred)):
    if ('SN' in y_pred[i]) == True:
        out.append(X_test[i][y_pred[i].index('SN')]['word.upper()'])
        sn_index.append(i)
    else :
        not_sn_index.append(i)
```

In [31]:
```python
#Creating new dataframe and assigning 'No Predictions' to values not assigned
final_pred = pd.DataFrame(out,sn_index)
final_pred.rename(columns={0: "Predicted Store Numbers"}, inplace = True)
for ab in not_sn_index:
    final_pred.loc[ab] = ['No Predictions']  # adding a new row
    final_pred = final_pred.sort_index()  # sorting by index
```

In [32]:
```python
#Comparing test data input string with predicted values
store_comp = pd.concat([test_data['transaction_descriptor'], final_pred],axis = 1)
#Printing final results
store_comp
```

Out[32]:

	transaction_descriptor	Predicted Store Numbers
0	IN-N-OUT BURGER #242	242
1	BP#9442088LIBERTYVILLE B	9442088
2	JCPENNEY 1419	1419
3	ROSS STORES #1019	1019
4	WM SUPERCENTER #38	38
...	...	...
95	MCDONALD'S F2151	F2151
96	NST BEST BUY #1403 332411	1403
97	CVS/PHARMACY #06689	6689
98	BANANA REPUBLIC #8109	8109
99	BOSTON MARKET 0443	443

100 rows × 2 columns

In [33]:
```python
#Export the result as CSV file
store_comp.to_csv('Test_Result.csv',index=False)
```