In [1]:

```
# Load the required packages
import re
import nltk
import numpy as np
import pandas as pd
from sklearn import svm, metrics
from sklearn.preprocessing import LabelEncoder
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.model_selection import train_test_split, learning_curve, StratifiedShu

import matplotlib
import matplotlib.pyplot as plt
import seaborn as sns

# Improve the readability of figures
sns.set_context('notebook', font_scale=1.4)
%config InlineBackend.figure_format = 'retina'
%matplotlib inline
```

In [2]:

```
# Load the dataset
df = pd.read_table('SMSSpamCollection.txt', header=None)

# Display the first five rows
df.head()
```

Out[2]:

	0	1
0	ham	Go until jurong point, crazy.. Available only ...
1	ham	Ok lar... Joking wif u oni...
2	spam	Free entry in 2 a wkly comp to win FA Cup fina...
3	ham	U dun say so early hor... U c already then say...
4	ham	Nah I don't think he goes to usf, he lives aro...

In [3]:

```
# Store the target variable
y = df[0]

# Display the class distribution
y.value_counts()
```

Out[3]:

```
ham     4825
spam     747
Name: 0, dtype: int64
```

In [4]:

```
# Encode the class labels as numbers
le = LabelEncoder()
y_enc = le.fit_transform(y)
```

In [5]:

```
# Store the SMS message data
raw_text = df[1]
```

In [6]:

```
example = """  ***** CONGRATlations **** You won 2 tIckETs to Hamilton in
NYC http://www.hamiltonbroadway.com/J?NaIOl/event   wORtH over $500.00...CALL
555-477-8914 or send message to: hamilton@freetix.com to get ticket !! !  """
```

In [7]:
```
# Replace email addresses with 'emailaddr'
processed = raw_text.str.replace(r'\b[\w\-.]+?@\w+?\.\w{2,4}\b',
                                 'emailaddr')

# Replace URLs with 'httpaddr'
processed = processed.str.replace(r'(http[s]?\S+)|(\w+\.[A-Za-z]{2,4}\S*)',
                                  'httpaddr')

# Replace money symbols with 'moneysymb'
processed = processed.str.replace(r'£|\$', 'moneysymb')

# Replace phone numbers with 'phonenumbr'
processed = processed.str.replace(
    r'\b(\+\d{1,2}\s)?\d?[\-(.]?\d{3}\)?[\s.-]?\d{3}[\s.-]?\d{4}\b',
    'phonenumbr')

# Replace numbers with 'numbr'
processed = processed.str.replace(r'\d+(\.\d+)?', 'numbr')
```

In [8]:
```
# Remove punctuation
processed = processed.str.replace(r'[^\w\d\s]', ' ')

# Replace whitespace between terms with a single space
processed = processed.str.replace(r'\s+', ' ')

# Remove leading and trailing whitespace
processed = processed.str.replace(r'^\s+|\s+?$', '')
```

In [9]:
```
# Lowercase the corpus
processed = processed.str.lower()
```

In [10]:
```
# Access stop words
stop_words = nltk.corpus.stopwords.words('english')
```

In [11]:
```
# Remove all stop words
processed = processed.apply(lambda x: ' '.join(
    term for term in x.split() if term not in set(stop_words))
)
```

In [12]:
```
# Remove word stems using a Porter stemmer
porter = nltk.PorterStemmer()
processed = processed.apply(lambda x: ' '.join(
    porter.stem(term) for term in x.split())
)
```

In [13]:
```
def preprocess_text(messy_string):
    assert(type(messy_string) == str)
    cleaned = re.sub(r'\b[\w\-.]+?@\w+?\.\w{2,4}\b', 'emailaddr', messy_string)
    cleaned = re.sub(r'(http[s]?\S+)|(\w+\.[A-Za-z]{2,4}\S*)', 'httpaddr',
                     cleaned)
    cleaned = re.sub(r'£|\$', 'moneysymb', cleaned)
    cleaned = re.sub(
        r'\b(\+\d{1,2}\s)?\d?[\-(.]?\d{3}\)?[\s.-]?\d{3}[\s.-]?\d{4}\b',
        'phonenumbr', cleaned)
    cleaned = re.sub(r'\d+(\.\d+)?', 'numbr', cleaned)
    cleaned = re.sub(r'[^\w\d\s]', ' ', cleaned)
    cleaned = re.sub(r'\s+', ' ', cleaned)
    cleaned = re.sub(r'^\s+|\s+?$', '', cleaned.lower())
    return ' '.join(
        porter.stem(term)
        for term in cleaned.split()
        if term not in set(stop_words)
    )
```

In [14]:
```
(processed == raw_text.apply(preprocess_text)).all()
```

Out[14]:
```
True
```

In [15]:
```
preprocess_text(example)
```

Out[15]:
```
'congratl numbr ticket hamilton nyc httpaddr worth moneysymbnumbr call phonenumbr
send messag emailaddr get ticket'
```

In [16]:
```
vectorizer = TfidfVectorizer(ngram_range=(1, 2))
X_ngrams = vectorizer.fit_transform(processed)
```

In [17]:
```
X_ngrams.shape
```

Out[17]:
```
(5572, 36348)
```

In [18]:
```
# Prepare the training and test sets using an 80/20 split
X_train, X_test, y_train, y_test = train_test_split(
    X_ngrams,
    y_enc,
    test_size=0.2,
    random_state=42,
    stratify=y_enc
)

# Train SVM with a linear kernel on the training set
clf = svm.LinearSVC(loss='hinge')
clf.fit(X_train, y_train)

# Evaluate the classifier on the test set
y_pred = clf.predict(X_test)

# Compute the F1 score
metrics.f1_score(y_test, y_pred)
```

Out[18]:
```
0.9285714285714286
```

In [19]:
```
# Display a confusion matrix
pd.DataFrame(
    metrics.confusion_matrix(y_test, y_pred),
    index=[['actual', 'actual'], ['spam', 'ham']],
    columns=[['predicted', 'predicted'], ['spam', 'ham']]
)
```

Out[19]:

		predicted	
		spam	**ham**
actual	**spam**	965	1
	ham	19	130

In [20]:
```
# Select 10 different sizes of the entire dataset
sample_space = np.linspace(500, len(raw_text) * 0.8, 10, dtype='int')
```

```
# Compute learning curves without regularization for the SVM model
train_sizes, train_scores, valid_scores = learning_curve(
    estimator=svm.LinearSVC(loss='hinge', C=1e10),
    X=X_ngrams,
    y=y_enc,
    train_sizes=sample_space,
    cv=StratifiedShuffleSplit(n_splits=10, test_size=0.2, random_state=40),
    scoring='f1',
    n_jobs=-1
)
```

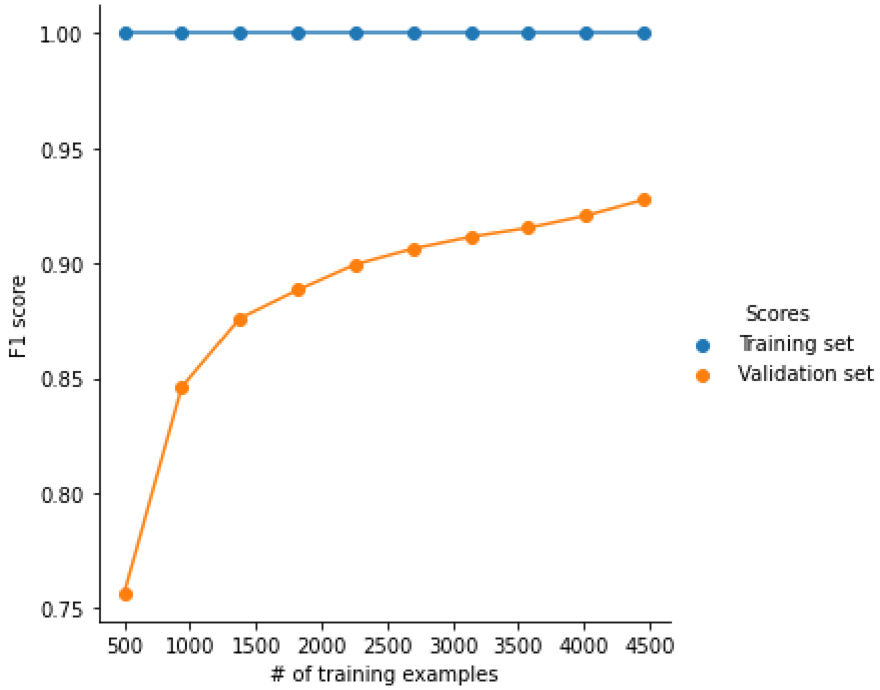
In [21]:

```
def make_tidy(sample_space, train_scores, valid_scores):
    # Join train_scores and valid_scores, and label with sample_space
    messy_format = pd.DataFrame(
        np.stack((sample_space, train_scores.mean(axis=1),
                  valid_scores.mean(axis=1)), axis=1),
        columns=['# of training examples', 'Training set', 'Validation set']
    )

    # Re-structure into into tidy format
    return pd.melt(
        messy_format,
        id_vars='# of training examples',
        value_vars=['Training set', 'Validation set'],
        var_name='Scores',
        value_name='F1 score'
    )
```

In [22]:

```
# Initialize a FacetGrid object using the table of scores and facet on
# the type of score
g = sns.FacetGrid(
    make_tidy(sample_space, train_scores, valid_scores), hue='Scores', size=5
)

g.map(plt.scatter, '# of training examples', 'F1 score')
g.map(plt.plot, '# of training examples', 'F1 score').add_legend();
```

```
C:\Users\guwalani.kunal\Anaconda3\lib\site-packages\seaborn\axisgrid.py:243: UserW
arning: The `size` parameter has been renamed to `height`; please update your cod
e.
  warnings.warn(msg, UserWarning)
```



In [23]:

```
param_grid = [{'C': np.logspace(-4, 4, 20)}]

grid_search = GridSearchCV(
    estimator=svm.LinearSVC(loss='hinge'),
    param_grid=param_grid,
    cv=StratifiedShuffleSplit(n_splits=10, test_size=0.2, random_state=42),
    scoring='f1',
    n_jobs=-1
)

scores = cross_val_score(
    estimator=grid_search,
    X=X_ngrams,
    y=y_enc,
    cv=StratifiedShuffleSplit(n_splits=10, test_size=0.2, random_state=0),
    scoring='f1',
    n_jobs=-1
)

scores
```

Out[23]:

```
array([0.91636364, 0.94366197, 0.95104895, 0.93661972, 0.94736842,
       0.93286219, 0.91039427, 0.90510949, 0.9057971 , 0.94699647])
```

In [24]:

```
scores.mean()
```

Out[24]:

```
0.9296222211583224
```

In [25]:

```
# Identify the optimal regularization hyperparameter
grid_search.fit(X_ngrams, y_enc)

# Train the classifier on the entire dataset using the optimal hyperparameter
final_clf = svm.LinearSVC(loss='hinge', C=grid_search.best_params_['C'])
final_clf.fit(X_ngrams, y_enc);
```

In [26]:

```
# Display the features with the highest weights in the SVM model
pd.Series(
    final_clf.coef_.T.ravel(),
    index=vectorizer.get_feature_names()
).sort_values(ascending=False)[:20]
```

Out[26]:

```
phonenumbr          5.008632
numbrp              2.799188
txt                 2.690816
moneysymbnumbr      2.557429
call phonenumbr     2.251018
rington             2.098571
servic              2.049272
mobil               2.036899
numbr               1.896236
tone                1.831284
repli               1.664236
text                1.603976
claim               1.590065
video               1.473553
free                1.359939
wap                 1.336547
stop                1.310738
credit              1.278886
uk                  1.239139
order               1.227617
dtype: float64
```

In [27]:

```
def spam_filter(message):
    if final_clf.predict(vectorizer.transform([preprocess_text(message)])):
        return 'spam'
    else:
        return 'not spam'
```

In [28]:

```
spam_filter(example)
```

Out[28]:

```
'spam'
```

In [29]:

```
spam_filter('Ohhh, but those are the best kind of foods')
```

Out[29]:

```
'not spam'
```

In [ ]: