Mnist

June 26, 2025

1 Ahmad Siddiqui

2 Manual task 1 cateory 2

2.1 Question

1. OvO vs. OvR:

One-vs-One (OvO) and One-vs-Rest (OvR) are strategies for extending binary classifiers to multi-class problems.

Feature	One-vs-Rest (OvR)	One-vs-One (OvO)
Number of classifiers	`n`	`n(n - 1)/2`
Each classifier trained on	All data, with one class vs rest	Only data from two classes
Training time	Faster (fewer classifiers)	Slower (more classifiers)
Prediction	Highest decision score among classifiers	Majority vote among all pairwise classifiers
Memory usage	Lower	Higher
Good for many classes	Yes	No (quadratic growth)
Boundary granularity	Coarser	Finer
Score interpretability	Easier (direct per-class scores)	Harder (votes must be aggregated)
Supported in sklearn	LogisticRegression, RidgeClassifier, etc.	SVC (default with kernel), GaussianNB, etc.
Typical use case	Linear models, high-class problems	SVMs, low to moderate number of classes

2. SGD vs RF classifier

Feature / Aspect	`SGDClassifier`	`RandomForestClassifier`
Type	Linear model trained with stochastic gradient descent	Ensemble of decision trees (bagging)
Model Complexity	Linear (or linear + kernel trick manually)	Non-linear, high-capacity

Feature / Aspect	`SGDClassifier`	`RandomForestClassifier`
Training Speed	**Very fast**, scalable to millions of samples	Slower, especially with many trees or features
Prediction Speed	**Very fast** (single dot product)	Slower (tree traversal per tree, many trees)
Memory Usage	Low	High (stores many full trees)
Handles Multiclass	Yes (`ovr`, `multinomial`)	Yes (natively)
Works with Sparse Data	**Yes (natively)**	No (must densify input)
Regularization	Supports L1, L2, ElasticNet	No direct regularization, prone to overfitting
Handles Non-linear Data	Poorly unless kernelized or feature engineered	**Very well**
Overfitting Risk	Low (but high bias)	Medium to high (trees can memorize noise)
Interpretability	High (weights per feature)	Medium-low (can use feature importances)
Feature Scaling Required	**Yes** (mandatory for convergence)	**No**
Parallelization	Difficult (single-pass)	**Yes** (trees can be grown in parallel)
Handles Missing Values	No (must impute manually)	Yes (can tolerate missing splits)
Typical Use Cases	Text classification, online learning, large-scale ML	Tabular data, feature-rich problems, small-medium datasets

[1]:
```
import sys
assert sys.version_info >= (3, 5)
import sklearn
import numpy as np
import os
np.random.seed(42)

%matplotlib inline
import matplotlib as mpl
import matplotlib.pyplot as plt
mpl.rc('axes', labelsize=14)
mpl.rc('xtick', labelsize=12)
mpl.rc('ytick', labelsize=12)

# Where to save the figures
PROJECT_ROOT_DIR = "./content"
CHAPTER_ID = "assignment"
IMAGES_PATH = os.path.join(PROJECT_ROOT_DIR, "images", CHAPTER_ID)
os.makedirs(IMAGES_PATH, exist_ok=True)
```

```
def save_fig(fig_id, tight_layout=True, fig_extension="png", resolution=300):
    path = os.path.join(IMAGES_PATH, fig_id + "." + fig_extension)
    print("Saving figure", fig_id)
    if tight_layout:
        plt.tight_layout()
    plt.savefig(path, format=fig_extension, dpi=resolution)
```

[2]:
```
from sklearn.datasets import fetch_openml
mnist = fetch_openml('mnist_784', version=1, as_frame=False)
mnist.keys()
```

[2]:
```
dict_keys(['data', 'target', 'frame', 'categories', 'feature_names',
'target_names', 'DESCR', 'details', 'url'])
```

[3]:
```
X, y = mnist["data"], mnist["target"]
X.shape
```

[3]:
```
(70000, 784)
```

[4]:
```
y.shape
```

[4]:
```
(70000,)
```

[5]:
```
28 * 28
```

[5]:
```
784
```

[6]:
```
%matplotlib inline
import matplotlib as mpl
import matplotlib.pyplot as plt

some_digit = X[0]
some_digit_image = some_digit.reshape(28, 28)
plt.imshow(some_digit_image, cmap=mpl.cm.binary)
plt.axis("off")

save_fig("some_digit_plot")
plt.show()
```

```
Saving figure some_digit_plot
```

[7]:
```
y[0]
```

[7]:
```
'5'
```

[8]:
```
y = y.astype(np.uint8)
```

[9]:
```
def plot_digit(data):
    image = data.reshape(28, 28)
    plt.imshow(image, cmap = mpl.cm.binary,
               interpolation="nearest")
    plt.axis("off")
```

[ ]:
```
y[0]
```

[ ]:
```
5
```

[12]:
```
X_train, X_test, y_train, y_test = X[:60000], X[60000:], y[:60000], y[60000:]
```

3 Training a Binary Classifier

[13]:
```
y_train_5 = (y_train == 5)
y_test_5 = (y_test == 5)
```

[14]:
```
from sklearn.linear_model import SGDClassifier

sgd_clf = SGDClassifier(max_iter=1000, tol=1e-3, random_state=42)
sgd_clf.fit(X_train, y_train_5)
```

[14]:
```
SGDClassifier(random_state=42)
```

[15]:
```
sgd_clf.predict([some_digit])
```

[15]:
```
array([ True])
```

[16]:
```
from sklearn.model_selection import cross_val_score
cross_val_score(sgd_clf, X_train, y_train_5, cv=3, scoring="accuracy")
```

[16]:
```
array([0.95035, 0.96035, 0.9604 ])
```

3.1 Confusion Matrix

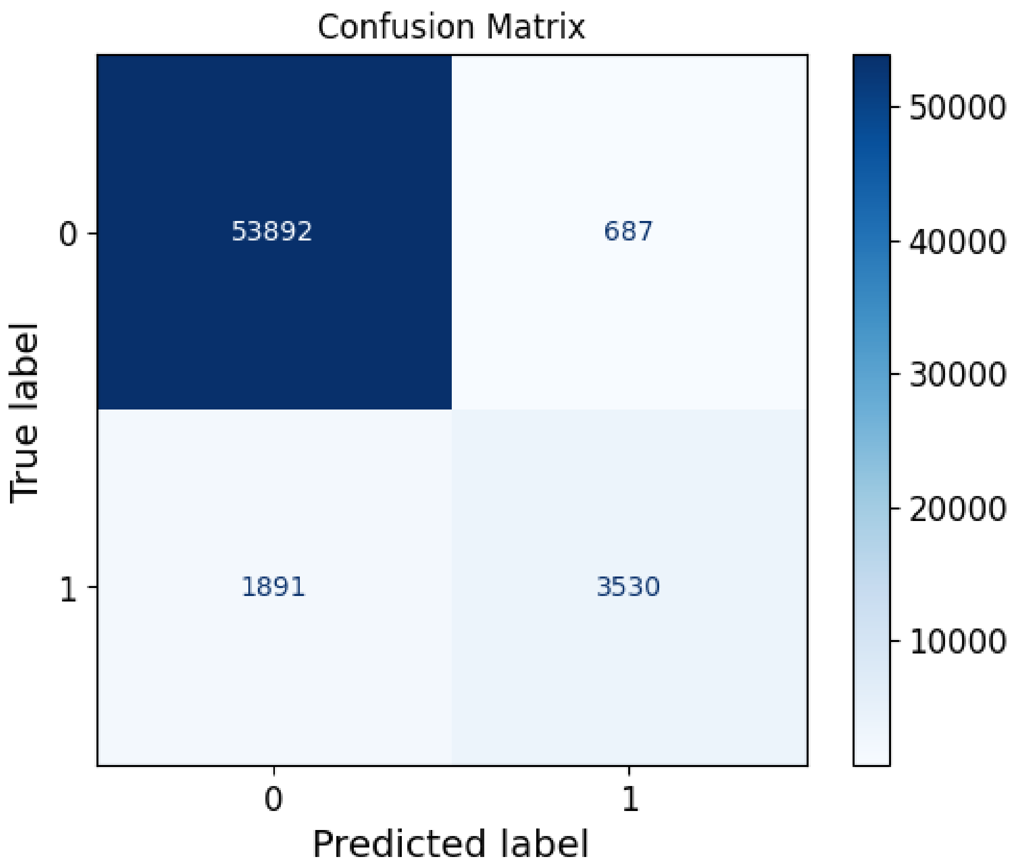
[17]:
```
from sklearn.model_selection import cross_val_predict

y_train_pred = cross_val_predict(sgd_clf, X_train, y_train_5, cv=3)
```

[18]:
```
from sklearn.metrics import confusion_matrix

confusion_matrix(y_train_5, y_train_pred)
```

[18]:
```
array([[53892,   687],
       [ 1891,  3530]])
```

[26]:
```
# To plot the confusion matrix nicely, we can use `matshow()` from `matplotlib.
↪pyplot`
# and add labels and colorbar.

from sklearn.metrics import ConfusionMatrixDisplay

def plot_confusion_matrix(y_true, y_pred, classes):
    cm = confusion_matrix(y_true, y_pred)
    disp = ConfusionMatrixDisplay(confusion_matrix=cm, display_labels=classes)
    disp.plot(cmap=plt.cm.Blues)
    plt.title("Confusion Matrix")
    plt.show()
```

```
# Example usage (assuming you have y_train_5 and y_train_pred defined):
# You would need to determine the classes based on your binary classification␣
↪(e.g., [False, True])
classes = [0, 1] # Assuming the classes are 0 (not 5) and 1 (is 5)
plot_confusion_matrix(y_train_5, y_train_pred, classes)
```



[66]:
```
from sklearn.metrics import classification_report
print("***  Classification Report ***"*3)
print(classification_report(y_train_5, y_train_pred))
```

```
***  Classification Report ******  Classification Report ******  Classification
Report ***
              precision    recall  f1-score   support

       False       0.97      0.99      0.98     54579
        True       0.84      0.65      0.73      5421

    accuracy                           0.96     60000
   macro avg       0.90      0.82      0.85     60000
```

```
weighted avg       0.95      0.96      0.95     60000
```

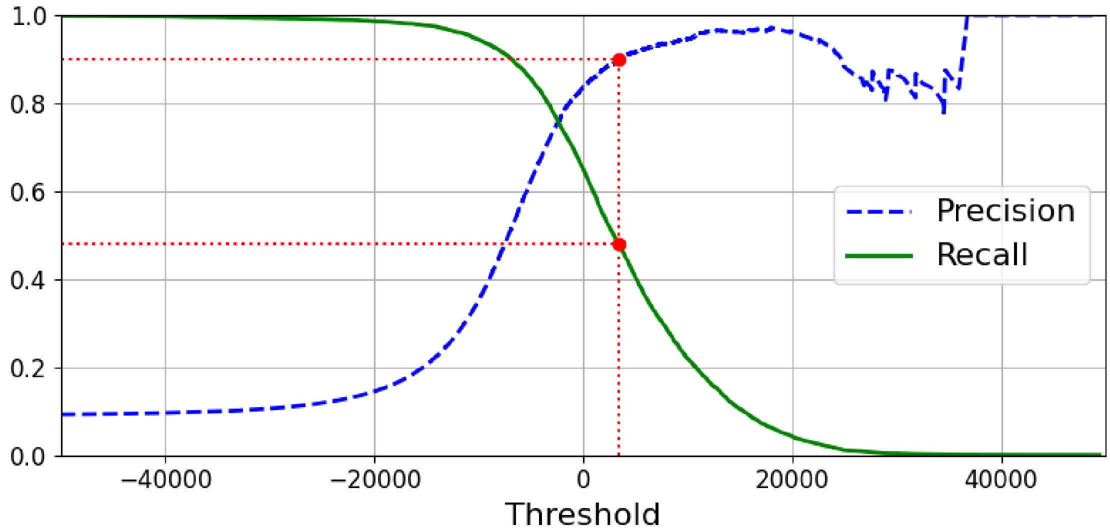
3.2 Precision/Recall Trade-off

[31]:
```python
y_scores = cross_val_predict(sgd_clf, X_train, y_train_5, cv=3,
                             method="decision_function")
```

[32]:
```python
from sklearn.metrics import precision_recall_curve

precisions, recalls, thresholds = precision_recall_curve(y_train_5, y_scores)
```

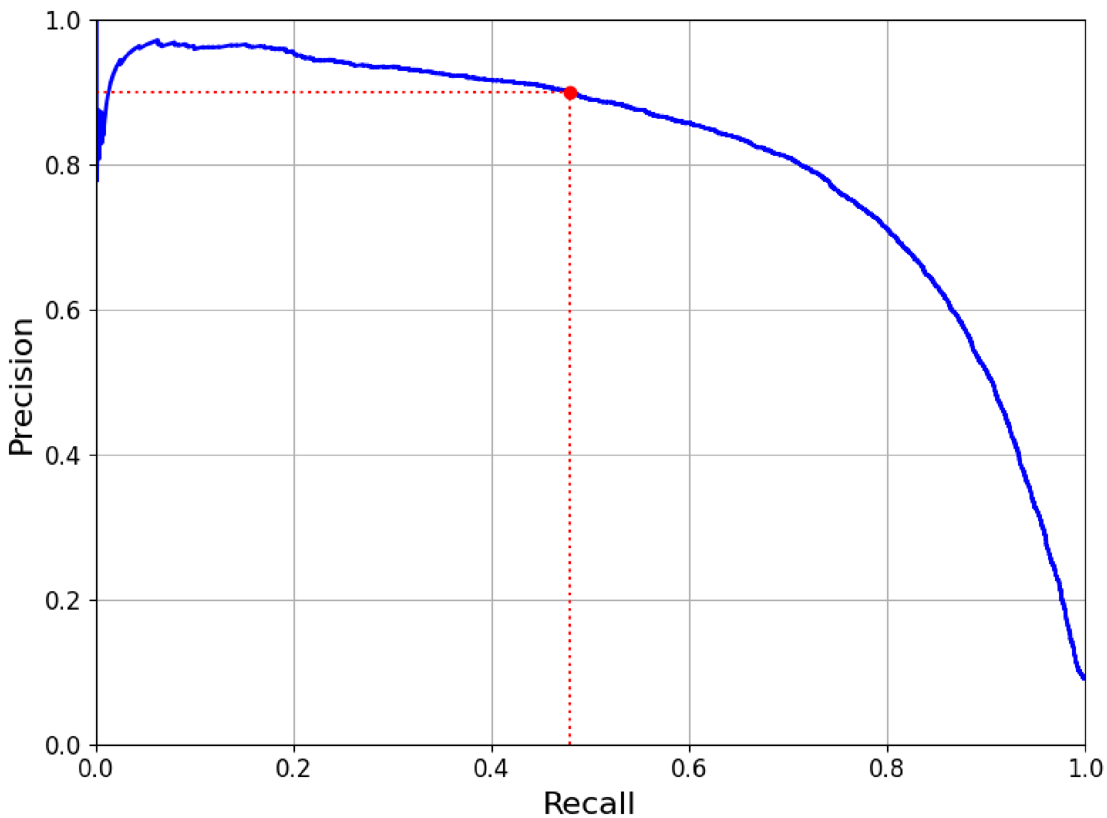
[33]:
```python
def plot_precision_recall_vs_threshold(precisions, recalls, thresholds):
    plt.plot(thresholds, precisions[:-1], "b--", label="Precision", linewidth=2)
    plt.plot(thresholds, recalls[:-1], "g-", label="Recall", linewidth=2)
    plt.legend(loc="center right", fontsize=16) # Not shown in the book
    plt.xlabel("Threshold", fontsize=16)        # Not shown
    plt.grid(True)                              # Not shown
    plt.axis([-50000, 50000, 0, 1])             # Not shown



recall_90_precision = recalls[np.argmax(precisions >= 0.90)]
threshold_90_precision = thresholds[np.argmax(precisions >= 0.90)]


plt.figure(figsize=(8, 4))                                                      ␣
↪              # Not shown
plot_precision_recall_vs_threshold(precisions, recalls, thresholds)
plt.plot([threshold_90_precision, threshold_90_precision], [0., 0.9], "r:")    ␣
↪              # Not shown
plt.plot([-50000, threshold_90_precision], [0.9, 0.9], "r:")                    ␣
↪              # Not shown
plt.plot([-50000, threshold_90_precision], [recall_90_precision,␣
↪recall_90_precision], "r:")# Not shown
plt.plot([threshold_90_precision], [0.9], "ro")                                 ␣
↪              # Not shown
plt.plot([threshold_90_precision], [recall_90_precision], "ro")                 ␣
↪              # Not shown
save_fig("precision_recall_vs_threshold_plot")                                  ␣
↪              # Not shown
plt.show()
```

```
Saving figure precision_recall_vs_threshold_plot
```



[34]:
```python
def plot_precision_vs_recall(precisions, recalls):
    plt.plot(recalls, precisions, "b-", linewidth=2)
    plt.xlabel("Recall", fontsize=16)
    plt.ylabel("Precision", fontsize=16)
    plt.axis([0, 1, 0, 1])
    plt.grid(True)

plt.figure(figsize=(8, 6))
plot_precision_vs_recall(precisions, recalls)
plt.plot([recall_90_precision, recall_90_precision], [0., 0.9], "r:")
plt.plot([0.0, recall_90_precision], [0.9, 0.9], "r:")
plt.plot([recall_90_precision], [0.9], "ro")
save_fig("precision_vs_recall_plot")
plt.show()
```

```
Saving figure precision_vs_recall_plot
```



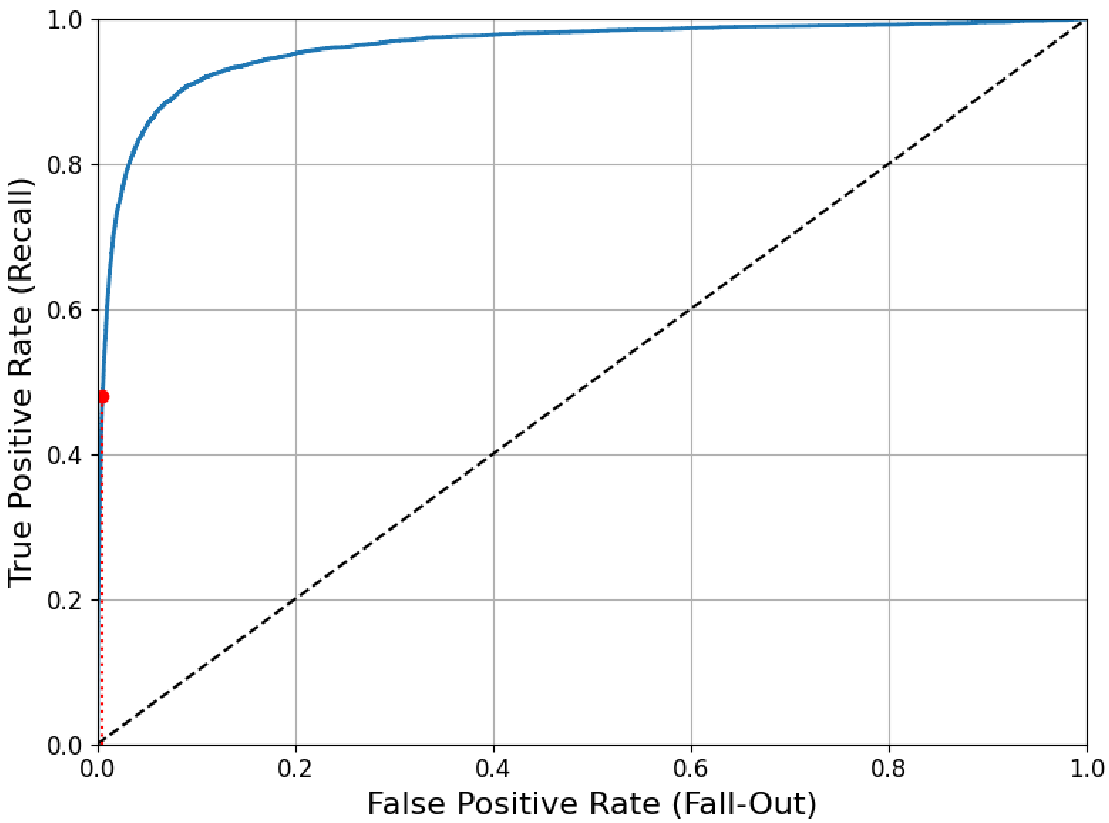
3.3 The ROC Curve

[40]:
```
from sklearn.metrics import roc_curve

fpr, tpr, thresholds = roc_curve(y_train_5, y_scores)
```

[41]:
```
def plot_roc_curve(fpr, tpr, label=None):
    plt.plot(fpr, tpr, linewidth=2, label=label)
    plt.plot([0, 1], [0, 1], 'k--') # dashed diagonal
    plt.axis([0, 1, 0, 1])                                    # Not shown in␣
 ↪the book
    plt.xlabel('False Positive Rate (Fall-Out)', fontsize=16) # Not shown
    plt.ylabel('True Positive Rate (Recall)', fontsize=16)    # Not shown
    plt.grid(True)                                            # Not shown

plt.figure(figsize=(8, 6))                                    # Not shown
plot_roc_curve(fpr, tpr)
fpr_90 = fpr[np.argmax(tpr >= recall_90_precision)]           # Not shown
plt.plot([fpr_90, fpr_90], [0., recall_90_precision], "r:")   # Not shown
```

```
plt.plot([0.0, fpr_90], [recall_90_precision, recall_90_precision], "r:")  #␣
 ↪Not shown
plt.plot([fpr_90], [recall_90_precision], "ro")             # Not shown
save_fig("roc_curve_plot")                                  # Not shown
plt.show()
```

Saving figure roc_curve_plot



[42]:
```
from sklearn.metrics import roc_auc_score

roc_auc_score(y_train_5, y_scores)
```

[42]: np.float64(0.9604938554008616)

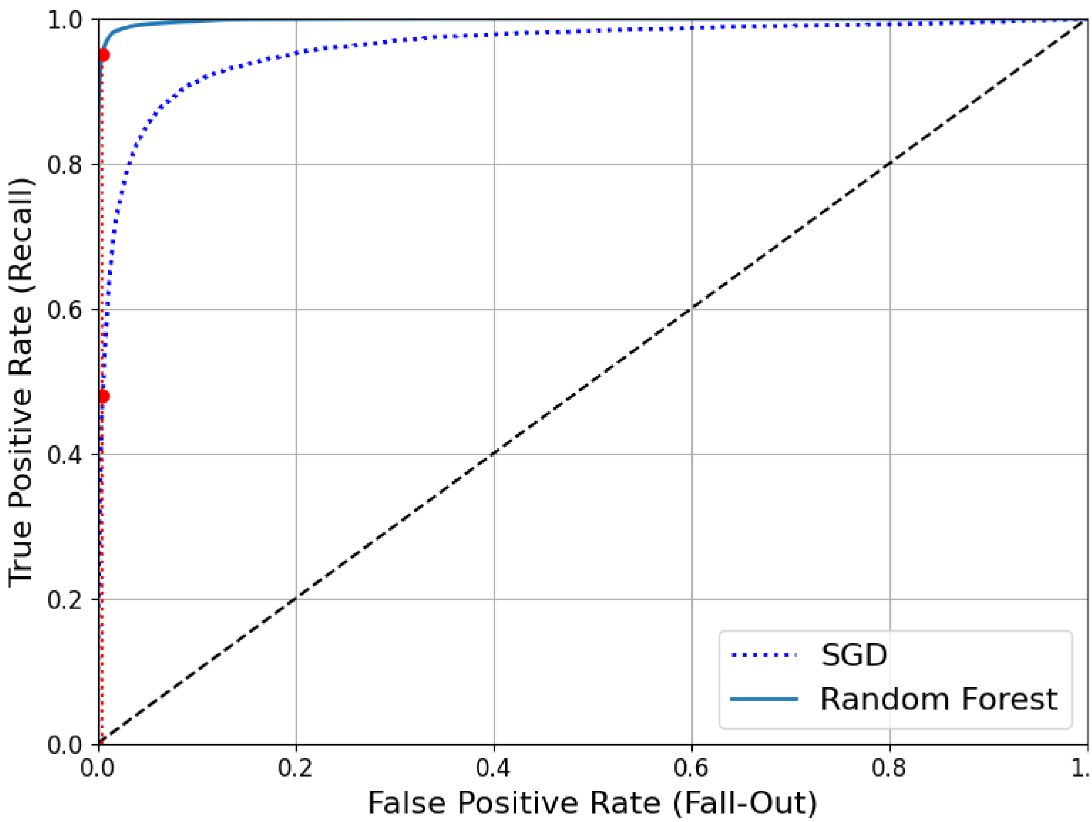
[43]:
```
from sklearn.ensemble import RandomForestClassifier
forest_clf = RandomForestClassifier(n_estimators=100, random_state=42)
y_probas_forest = cross_val_predict(forest_clf, X_train, y_train_5, cv=3,
                                    method="predict_proba")
```

[44]:
```
y_scores_forest = y_probas_forest[:, 1] # score = proba of positive class
fpr_forest, tpr_forest, thresholds_forest = roc_curve(y_train_5,y_scores_forest)
```

[45]:

```
recall_for_forest = tpr_forest[np.argmax(fpr_forest >= fpr_90)]

plt.figure(figsize=(8, 6))
plt.plot(fpr, tpr, "b:", linewidth=2, label="SGD")
plot_roc_curve(fpr_forest, tpr_forest, "Random Forest")
plt.plot([fpr_90, fpr_90], [0., recall_90_precision], "r:")
plt.plot([0.0, fpr_90], [recall_90_precision, recall_90_precision], "r:")
plt.plot([fpr_90], [recall_90_precision], "ro")
plt.plot([fpr_90, fpr_90], [0., recall_for_forest], "r:")
plt.plot([fpr_90], [recall_for_forest], "ro")
plt.grid(True)
plt.legend(loc="lower right", fontsize=16)
save_fig("roc_curve_comparison_plot")
plt.show()
```

```
Saving figure roc_curve_comparison_plot
```



[46]:

```
roc_auc_score(y_train_5, y_scores_forest)
```

[46]:

```
np.float64(0.9983436731328145)
```

[47]:
```
y_train_pred_forest = cross_val_predict(forest_clf, X_train, y_train_5, cv=3)
precision_score(y_train_5, y_train_pred_forest)
```

[47]: 0.9905083315756169

[48]:
```
recall_score(y_train_5, y_train_pred_forest)
```

[48]: 0.8662608374838591

4 Multilabel Classification

[49]:
```
from sklearn.neighbors import KNeighborsClassifier

y_train_large = (y_train >= 7)
y_train_odd = (y_train % 2 == 1)
y_multilabel = np.c_[y_train_large, y_train_odd]

knn_clf = KNeighborsClassifier()
knn_clf.fit(X_train, y_multilabel)
```

[49]: KNeighborsClassifier()

[50]:
```
knn_clf.predict([some_digit])
```

[50]: array([[False, True]])

Warning: the following cell may take a very long time (possibly hours depending on your hardware).

[51]:
```
y_train_knn_pred = cross_val_predict(knn_clf, X_train, y_multilabel, cv=3)
f1_score(y_multilabel, y_train_knn_pred, average="macro")
```

[51]: 0.9764102655606048

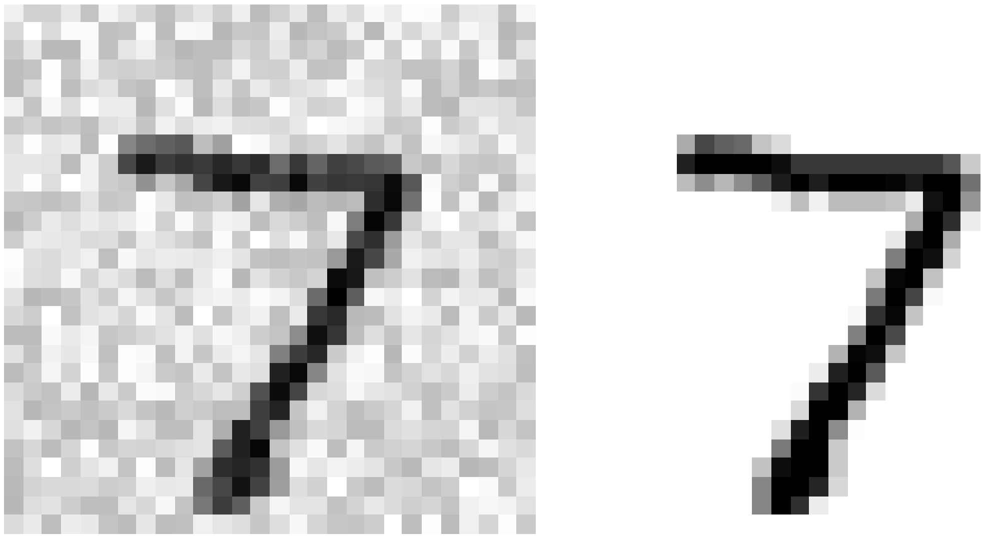
5 Multioutput Classification

[52]:
```
noise = np.random.randint(0, 100, (len(X_train), 784))
X_train_mod = X_train + noise
noise = np.random.randint(0, 100, (len(X_test), 784))
X_test_mod = X_test + noise
y_train_mod = X_train
y_test_mod = X_test
```

[53]:
```
some_index = 0
plt.subplot(121); plot_digit(X_test_mod[some_index])
plt.subplot(122); plot_digit(y_test_mod[some_index])
```

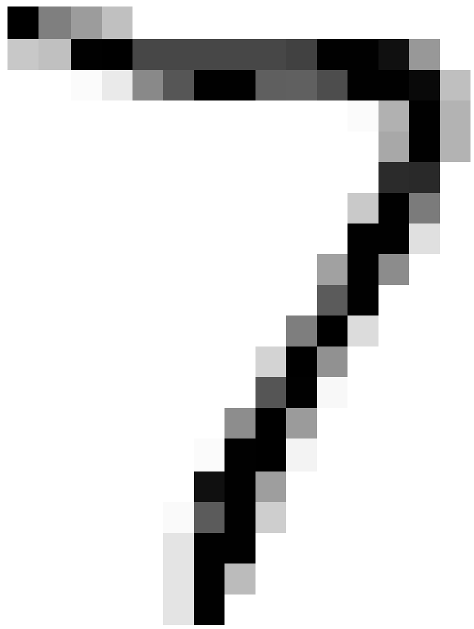
```
save_fig("noisy_digit_example_plot")
plt.show()
```

Saving figure noisy_digit_example_plot



[54]:
```
knn_clf.fit(X_train_mod, y_train_mod)
clean_digit = knn_clf.predict([X_test_mod[some_index]])
plot_digit(clean_digit)
save_fig("cleaned_digit_example_plot")
```

Saving figure cleaned_digit_example_plot



[55]:
```
from sklearn.neighbors import KNeighborsClassifier
knn_clf = KNeighborsClassifier(weights='distance', n_neighbors=4)
knn_clf.fit(X_train, y_train)
```

[55]:
```
KNeighborsClassifier(n_neighbors=4, weights='distance')
```

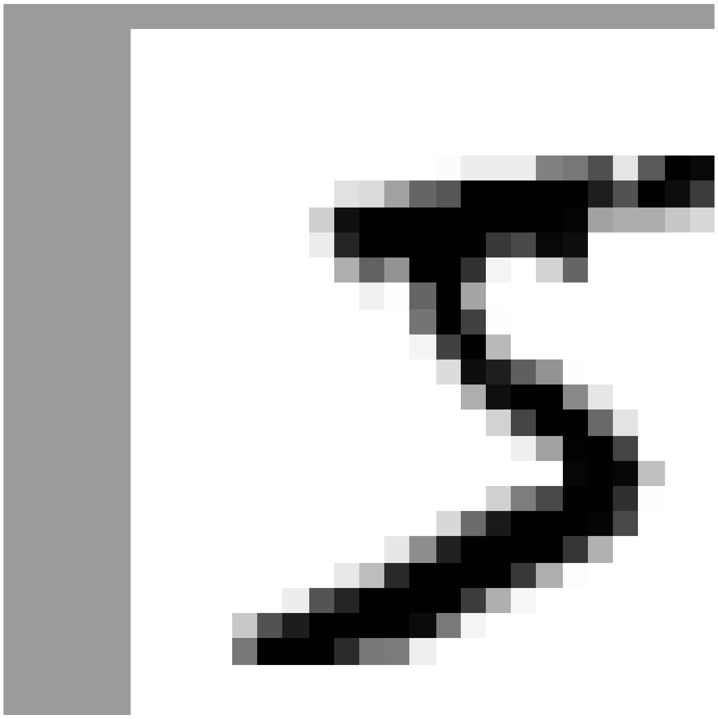
[56]:
```
y_knn_pred = knn_clf.predict(X_test)
```

[57]:
```
from sklearn.metrics import accuracy_score
accuracy_score(y_test, y_knn_pred)
```

[57]:
```
0.9714
```

[58]:
```
from scipy.ndimage.interpolation import shift
def shift_digit(digit_array, dx, dy, new=0):
    return shift(digit_array.reshape(28, 28), [dy, dx], cval=new).reshape(784)

plot_digit(shift_digit(some_digit, 5, 1, new=100))
```

```
/tmp/ipython-input-58-3518175631.py:1: DeprecationWarning: Please import `shift`
from the `scipy.ndimage` namespace; the `scipy.ndimage.interpolation` namespace
is deprecated and will be removed in SciPy 2.0.0.
  from scipy.ndimage.interpolation import shift
```



[59]:
```python
X_train_expanded = [X_train]
y_train_expanded = [y_train]
for dx, dy in ((1, 0), (-1, 0), (0, 1), (0, -1)):
    shifted_images = np.apply_along_axis(shift_digit, axis=1, arr=X_train,␣
 ↪dx=dx, dy=dy)
    X_train_expanded.append(shifted_images)
    y_train_expanded.append(y_train)

X_train_expanded = np.concatenate(X_train_expanded)
y_train_expanded = np.concatenate(y_train_expanded)
X_train_expanded.shape, y_train_expanded.shape
```

[59]: ((300000, 784), (300000,))

[60]:
```python
knn_clf.fit(X_train_expanded, y_train_expanded)
```

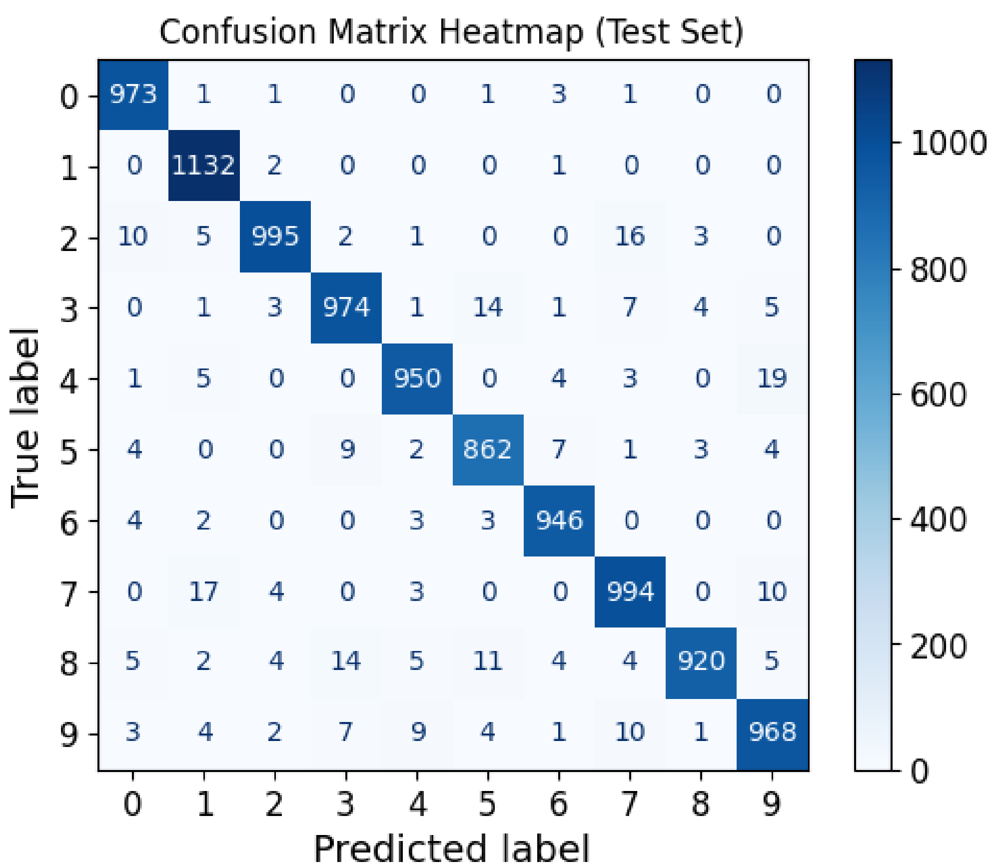
[60]: KNeighborsClassifier(n_neighbors=4, weights='distance')

[61]:
```python
y_knn_expanded_pred = knn_clf.predict(X_test)
```

[62]:
```
accuracy_score(y_test, y_knn_expanded_pred)
```

[62]: 0.9763

[68]:
```
# prompt: create a confusionnmatric heatmap

cm = confusion_matrix(y_test, y_knn_pred)
disp = ConfusionMatrixDisplay(confusion_matrix=cm, display_labels=knn_clf.
 ↪classes_)
disp.plot(cmap=plt.cm.Blues)
plt.title("Confusion Matrix Heatmap (Test Set)")
plt.show()
```



[69]:
```
print("\n" + "*** Classification Report (Test Set) ***"*3)
print(classification_report(y_test, y_knn_pred))
```

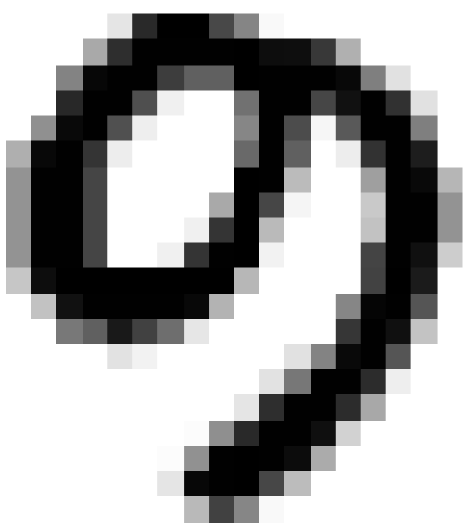
```
*** Classification Report (Test Set) ****** Classification Report (Test Set)
****** Classification Report (Test Set) ***
              precision    recall  f1-score   support
```

```
           0       0.97      0.99      0.98       980
           1       0.97      1.00      0.98      1135
           2       0.98      0.96      0.97      1032
           3       0.97      0.96      0.97      1010
           4       0.98      0.97      0.97       982
           5       0.96      0.97      0.96       892
           6       0.98      0.99      0.98       958
           7       0.96      0.97      0.96      1028
           8       0.99      0.94      0.97       974
           9       0.96      0.96      0.96      1009

    accuracy                           0.97     10000
   macro avg       0.97      0.97      0.97     10000
weighted avg       0.97      0.97      0.97     10000
```

[63]:
```
ambiguous_digit = X_test[2589]
knn_clf.predict_proba([ambiguous_digit])
```

[63]:
```
array([[0.24579675, 0.        , 0.        , 0.        , 0.        ,
        0.        , 0.        , 0.        , 0.        , 0.75420325]])
```

[64]:
```
plot_digit(ambiguous_digit)
```



[72]:
```
!pip install gradio -q

import gradio as gr
import numpy as np
from PIL import Image

# Load the trained KNN classifier
knn_clf = KNeighborsClassifier(weights='distance', n_neighbors=4)
knn_clf.fit(X_train_expanded, y_train_expanded) # Using the expanded training␣
↪set

def classify_digit(image):
  """Classifies a single grayscale image of a digit using the trained KNN model.
↪"""
  # Ensure the input is a NumPy array and is grayscale
  image = np.array(image).astype(np.uint8)
  # Ensure the image is 28x28
  if image.shape != (28, 28):
    # Resize the image if it's not 28x28 (optional, but good practice for␣
↪consistency)
    image = Image.fromarray(image).resize((28, 28), Image.LANCZOS)
    image = np.array(image)

  # Flatten the image to a 1D array (784 features)
  image_flattened = image.reshape(1, -1)

  # Make a prediction
  prediction = knn_clf.predict(image_flattened)[0]

  return str(prediction)

# Create the Gradio interface
iface = gr.Interface(
    fn=classify_digit,
    inputs=gr.Image(width=28, height=28, image_mode='L'), # Grayscale image␣
↪input
    outputs="text",
    title="MNIST Digit Classifier",
    description="Upload a grayscale image of a digit (0-9) and the model will␣
↪classify it.",
    live=True # Enable live prediction as the user draws/uploads
)

# Launch the interface
iface.launch(debug=True)
```

```
It looks like you are running Gradio on a hosted a Jupyter notebook. For the
Gradio app to work, sharing must be enabled. Automatically setting `share=True`
(you can turn this off by setting `share=False` in `launch()` explicitly).

Colab notebook detected. This cell will run indefinitely so that you can see
errors and logs. To turn off, set debug=False in launch().
* Running on public URL: https://877d692049d404a873.gradio.live

This share link expires in 1 week. For free permanent hosting and GPU upgrades,
run `gradio deploy` from the terminal in the working directory to deploy to
Hugging Face Spaces (https://huggingface.co/spaces)
```

```
<IPython.core.display.HTML object>
```

```
Created dataset file at: .gradio/flagged/dataset1.csv
```

```
Traceback (most recent call last):
  File "/usr/local/lib/python3.11/dist-packages/gradio/queueing.py", line 625,
in process_events
    response = await route_utils.call_process_api(
               ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
  File "/usr/local/lib/python3.11/dist-packages/gradio/route_utils.py", line
322, in call_process_api
    output = await app.get_blocks().process_api(
             ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
  File "/usr/local/lib/python3.11/dist-packages/gradio/blocks.py", line 2191, in
process_api
    result = await self.call_function(
             ^^^^^^^^^^^^^^^^^^^^^^^^^
  File "/usr/local/lib/python3.11/dist-packages/gradio/blocks.py", line 1702, in
call_function
    prediction = await anyio.to_thread.run_sync(  # type: ignore
                 ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
  File "/usr/local/lib/python3.11/dist-packages/anyio/to_thread.py", line 56, in
run_sync
    return await get_async_backend().run_sync_in_worker_thread(
           ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
  File "/usr/local/lib/python3.11/dist-packages/anyio/_backends/_asyncio.py",
line 2470, in run_sync_in_worker_thread
    return await future
           ^^^^^^^^^^^^
  File "/usr/local/lib/python3.11/dist-packages/anyio/_backends/_asyncio.py",
line 967, in run
    result = context.run(func, *args)
             ^^^^^^^^^^^^^^^^^^^^^^^^
  File "/usr/local/lib/python3.11/dist-packages/gradio/utils.py", line 894, in
wrapper
    response = f(*args, **kwargs)
               ^^^^^^^^^^^^^^^^^^
```

```
  File "/tmp/ipython-input-72-48946710.py", line 14, in classify_digit
    image = np.array(image).astype(np.uint8)
            ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^

TypeError: int() argument must be a string, a bytes-like object or a real
number, not 'NoneType'

Keyboard interruption in main thread… closing server.
Killing tunnel 127.0.0.1:7860 <> https://877d692049d404a873.gradio.live
```

[72]:

6 Exercise solutions

6.1 1. An MNIST Classifier With Over 97% Accuracy

[ ]:
```python
from sklearn.model_selection import GridSearchCV

param_grid = [{'weights': ["uniform", "distance"], 'n_neighbors': [3, 4, 5]}]

knn_clf = KNeighborsClassifier()
grid_search = GridSearchCV(knn_clf, param_grid, cv=5, verbose=3)
grid_search.fit(X_train, y_train)
```

```
Fitting 5 folds for each of 6 candidates, totalling 30 fits
[CV] n_neighbors=3, weights=uniform …

[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.

[CV] … n_neighbors=3, weights=uniform, score=0.972, total=168.0min
[CV] n_neighbors=3, weights=uniform …

[Parallel(n_jobs=1)]: Done   1 out of   1 | elapsed: 168.0min remaining:    0.0s

[CV] … n_neighbors=3, weights=uniform, score=0.971, total=12.3min
[CV] n_neighbors=3, weights=uniform …

[Parallel(n_jobs=1)]: Done   2 out of   2 | elapsed: 180.3min remaining:    0.0s

[CV] … n_neighbors=3, weights=uniform, score=0.969, total=11.9min
[CV] n_neighbors=3, weights=uniform …
[CV] … n_neighbors=3, weights=uniform, score=0.969, total=12.5min
[CV] n_neighbors=3, weights=uniform …
[CV] … n_neighbors=3, weights=uniform, score=0.970, total=12.7min
[CV] n_neighbors=3, weights=distance …
[CV] … n_neighbors=3, weights=distance, score=0.972, total=12.5min
[CV] n_neighbors=3, weights=distance …
[CV] … n_neighbors=3, weights=distance, score=0.972, total=12.8min
[CV] n_neighbors=3, weights=distance …
[CV] … n_neighbors=3, weights=distance, score=0.970, total=12.6min
[CV] n_neighbors=3, weights=distance …
[CV] … n_neighbors=3, weights=distance, score=0.970, total=12.9min
```

```
[CV] n_neighbors=3, weights=distance …
[CV] … n_neighbors=3, weights=distance, score=0.971, total=11.3min
[CV] n_neighbors=4, weights=uniform …
[CV] … n_neighbors=4, weights=uniform, score=0.969, total=11.0min
[CV] n_neighbors=4, weights=uniform …
[CV] … n_neighbors=4, weights=uniform, score=0.968, total=11.0min
[CV] n_neighbors=4, weights=uniform …
[CV] … n_neighbors=4, weights=uniform, score=0.968, total=11.0min
[CV] n_neighbors=4, weights=uniform …
[CV] … n_neighbors=4, weights=uniform, score=0.967, total=11.0min
[CV] n_neighbors=4, weights=uniform …
[CV] … n_neighbors=4, weights=uniform, score=0.970, total=11.0min
[CV] n_neighbors=4, weights=distance …
[CV] … n_neighbors=4, weights=distance, score=0.973, total=11.0min
[CV] n_neighbors=4, weights=distance …
[CV] … n_neighbors=4, weights=distance, score=0.972, total=11.0min
[CV] n_neighbors=4, weights=distance …
[CV] … n_neighbors=4, weights=distance, score=0.970, total=11.0min
[CV] n_neighbors=4, weights=distance …
[CV] … n_neighbors=4, weights=distance, score=0.971, total=11.0min
[CV] n_neighbors=4, weights=distance …
[CV] … n_neighbors=4, weights=distance, score=0.972, total=11.3min
[CV] n_neighbors=5, weights=uniform …
[CV] … n_neighbors=5, weights=uniform, score=0.970, total=10.9min
[CV] n_neighbors=5, weights=uniform …
[CV] … n_neighbors=5, weights=uniform, score=0.970, total=11.0min
[CV] n_neighbors=5, weights=uniform …
[CV] … n_neighbors=5, weights=uniform, score=0.969, total=11.0min
[CV] n_neighbors=5, weights=uniform …
[CV] … n_neighbors=5, weights=uniform, score=0.968, total=11.1min
[CV] n_neighbors=5, weights=uniform …
[CV] … n_neighbors=5, weights=uniform, score=0.969, total=11.0min
[CV] n_neighbors=5, weights=distance …
[CV] … n_neighbors=5, weights=distance, score=0.970, total=93.6min
[CV] n_neighbors=5, weights=distance …
[CV] … n_neighbors=5, weights=distance, score=0.971, total=11.0min
[CV] n_neighbors=5, weights=distance …
[CV] … n_neighbors=5, weights=distance, score=0.970, total=10.9min
[CV] n_neighbors=5, weights=distance …
[CV] … n_neighbors=5, weights=distance, score=0.969, total=11.2min
[CV] n_neighbors=5, weights=distance …
[CV] … n_neighbors=5, weights=distance, score=0.971, total=11.1min

[Parallel(n_jobs=1)]: Done  30 out of  30 | elapsed: 582.5min finished
```

[ ]:
```
GridSearchCV(cv=5, estimator=KNeighborsClassifier(),
             param_grid=[{'n_neighbors': [3, 4, 5],
                          'weights': ['uniform', 'distance']}],
```

```
                 verbose=3)
```

[ ]:
```
grid_search.best_params_
```

[ ]: {'n_neighbors': 4, 'weights': 'distance'}

[ ]:
```
grid_search.best_score_
```

[ ]: 0.9716166666666666

[ ]:
```
from sklearn.metrics import accuracy_score

y_pred = grid_search.predict(X_test)
accuracy_score(y_test, y_pred)
```

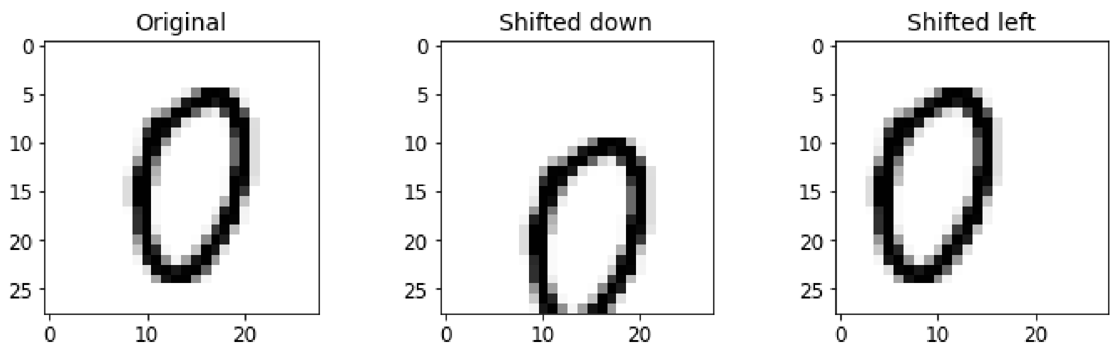
[ ]: 0.9714

6.2 2. Data Augmentation

[ ]:
```
from scipy.ndimage.interpolation import shift
```

[ ]:
```
def shift_image(image, dx, dy):
    image = image.reshape((28, 28))
    shifted_image = shift(image, [dy, dx], cval=0, mode="constant")
    return shifted_image.reshape([-1])
```

[ ]:
```
image = X_train[1000]
shifted_image_down = shift_image(image, 0, 5)
shifted_image_left = shift_image(image, -5, 0)

plt.figure(figsize=(12,3))
plt.subplot(131)
plt.title("Original", fontsize=14)
plt.imshow(image.reshape(28, 28), interpolation="nearest", cmap="Greys")
plt.subplot(132)
plt.title("Shifted down", fontsize=14)
plt.imshow(shifted_image_down.reshape(28, 28), interpolation="nearest",␣
 ↪cmap="Greys")
plt.subplot(133)
plt.title("Shifted left", fontsize=14)
plt.imshow(shifted_image_left.reshape(28, 28), interpolation="nearest",␣
 ↪cmap="Greys")
plt.show()
```




```
[ ]: X_train_augmented = [image for image in X_train]
     y_train_augmented = [label for label in y_train]

     for dx, dy in ((1, 0), (-1, 0), (0, 1), (0, -1)):
         for image, label in zip(X_train, y_train):
             X_train_augmented.append(shift_image(image, dx, dy))
             y_train_augmented.append(label)

     X_train_augmented = np.array(X_train_augmented)
     y_train_augmented = np.array(y_train_augmented)
```

```
[ ]: shuffle_idx = np.random.permutation(len(X_train_augmented))
     X_train_augmented = X_train_augmented[shuffle_idx]
     y_train_augmented = y_train_augmented[shuffle_idx]
```

```
[ ]: knn_clf = KNeighborsClassifier(**grid_search.best_params_)
```

```
[ ]: knn_clf.fit(X_train_augmented, y_train_augmented)
```

```
[ ]: KNeighborsClassifier(n_neighbors=4, weights='distance')
```

Warning: the following cell may take close to an hour to run, depending on your hardware.

```
[ ]: y_pred = knn_clf.predict(X_test)
     accuracy_score(y_test, y_pred)
```

```
[ ]: 0.9763
```

By simply augmenting the data, we got a 0.5% accuracy boost. :)

6.3 3. Tackle the Titanic dataset

The goal is to predict whether or not a passenger survived based on attributes such as their age, sex, passenger class, where they embarked and so on.

Let's fetch the data and load it:

```
[ ]: import os
import urllib.request

TITANIC_PATH = os.path.join("datasets", "titanic")
DOWNLOAD_URL = "https://raw.githubusercontent.com/ageron/handson-ml2/master/
 ↪datasets/titanic/"

def fetch_titanic_data(url=DOWNLOAD_URL, path=TITANIC_PATH):
    if not os.path.isdir(path):
        os.makedirs(path)
    for filename in ("train.csv", "test.csv"):
        filepath = os.path.join(path, filename)
        if not os.path.isfile(filepath):
            print("Downloading", filename)
            urllib.request.urlretrieve(url + filename, filepath)

fetch_titanic_data()
```

```
[ ]: import pandas as pd

def load_titanic_data(filename, titanic_path=TITANIC_PATH):
    csv_path = os.path.join(titanic_path, filename)
    return pd.read_csv(csv_path)
```

```
[ ]: train_data = load_titanic_data("train.csv")
test_data = load_titanic_data("test.csv")
```

The data is already split into a training set and a test set. However, the test data does *not* contain the labels: your goal is to train the best model you can using the training data, then make your predictions on the test data and upload them to Kaggle to see your final score.

```
[ ]: train_data.head()
```

```
[ ]:    PassengerId  Survived  Pclass  \
0            1         0       3
1            2         1       1
2            3         1       3
3            4         1       1
4            5         0       3

                                                Name     Sex   Age  SibSp  \
0                            Braund, Mr. Owen Harris    male  22.0      1
1  Cumings, Mrs. John Bradley (Florence Briggs Th…  female  38.0      1
2                             Heikkinen, Miss. Laina  female  26.0      0
3       Futrelle, Mrs. Jacques Heath (Lily May Peel)  female  35.0      1
4                           Allen, Mr. William Henry    male  35.0      0
```

```
   Parch            Ticket     Fare Cabin Embarked
0      0         A/5 21171   7.2500   NaN        S
1      0          PC 17599  71.2833   C85        C
2      0  STON/O2. 3101282   7.9250   NaN        S
3      0            113803  53.1000  C123        S
4      0            373450   8.0500   NaN        S
```

Let's explicitly set the `PassengerId` column as the index column:

```
[ ]: train_data = train_data.set_index("PassengerId")
     test_data = test_data.set_index("PassengerId")
```

Let's get more info to see how much data is missing:

```
[ ]: train_data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 891 entries, 1 to 891
Data columns (total 11 columns):
 #   Column    Non-Null Count  Dtype
---  ------    --------------  -----
 0   Survived  891 non-null    int64
 1   Pclass    891 non-null    int64
 2   Name      891 non-null    object
 3   Sex       891 non-null    object
 4   Age       714 non-null    float64
 5   SibSp     891 non-null    int64
 6   Parch     891 non-null    int64
 7   Ticket    891 non-null    object
 8   Fare      891 non-null    float64
 9   Cabin     204 non-null    object
 10  Embarked  889 non-null    object
dtypes: float64(2), int64(4), object(5)
memory usage: 83.5+ KB
```

```
[ ]: train_data[train_data["Sex"]=="female"]["Age"].median()
```

```
[ ]: 27.0
```

Let's take a look at the numerical attributes:

```
[ ]: train_data.describe()
```

```
[ ]:          Survived      Pclass         Age       SibSp       Parch        Fare
     count  891.000000  891.000000  714.000000  891.000000  891.000000  891.000000
     mean     0.383838    2.308642   29.699113    0.523008    0.381594   32.204208
     std      0.486592    0.836071   14.526507    1.102743    0.806057   49.693429
     min      0.000000    1.000000    0.416700    0.000000    0.000000    0.000000
```

```
25%      0.000000     2.000000   20.125000     0.000000     0.000000     7.910400
50%      0.000000     3.000000   28.000000     0.000000     0.000000    14.454200
75%      1.000000     3.000000   38.000000     1.000000     0.000000    31.000000
max      1.000000     3.000000   80.000000     8.000000     6.000000   512.329200
```

Now let's build our preprocessing pipelines, starting with the pipeline for numerical attributes:

[ ]:
```
from sklearn.pipeline import Pipeline
from sklearn.impute import SimpleImputer
from sklearn.preprocessing import StandardScaler

num_pipeline = Pipeline([
        ("imputer", SimpleImputer(strategy="median")),
        ("scaler", StandardScaler())
    ])
```

Now we can build the pipeline for the categorical attributes:

[ ]:
```
from sklearn.preprocessing import OneHotEncoder
```

[ ]:
```
cat_pipeline = Pipeline([
        ("imputer", SimpleImputer(strategy="most_frequent")),
        ("cat_encoder", OneHotEncoder(sparse=False)),
    ])
```

Finally, let's join the numerical and categorical pipelines:

[ ]:
```
from sklearn.compose import ColumnTransformer

num_attribs = ["Age", "SibSp", "Parch", "Fare"]
cat_attribs = ["Pclass", "Sex", "Embarked"]

preprocess_pipeline = ColumnTransformer([
        ("num", num_pipeline, num_attribs),
        ("cat", cat_pipeline, cat_attribs),
    ])
```

Cool! Now we have a nice preprocessing pipeline that takes the raw data and outputs numerical input features that we can feed to any Machine Learning model we want.

[ ]:
```
X_train = preprocess_pipeline.fit_transform(
    train_data[num_attribs + cat_attribs])
X_train
```

[ ]:
```
array([[-0.56573582,  0.43279337, -0.47367361, …,  0.        ,
         0.        ,  1.        ],
       [ 0.6638609 ,  0.43279337, -0.47367361, …,  1.        ,
         0.        ,  0.        ],
```

```
[-0.25833664, -0.4745452 , -0.47367361, …,  0.        ,
  0.        ,  1.        ],
…,
[-0.10463705,  0.43279337,  2.00893337, …,  0.        ,
  0.        ,  1.        ],
[-0.25833664, -0.4745452 , -0.47367361, …,  1.        ,
  0.        ,  0.        ],
[ 0.20276213, -0.4745452 , -0.47367361, …,  0.        ,
  1.        ,  0.        ]])
```

[ ]:
```
y_train = train_data["Survived"]
```

[ ]:
```
from sklearn.ensemble import RandomForestClassifier

forest_clf = RandomForestClassifier(n_estimators=100, random_state=42)
forest_clf.fit(X_train, y_train)
```

[ ]:
```
RandomForestClassifier(random_state=42)
```

[ ]:
```
X_test = preprocess_pipeline.transform(test_data[num_attribs + cat_attribs])
y_pred = forest_clf.predict(X_test)
```

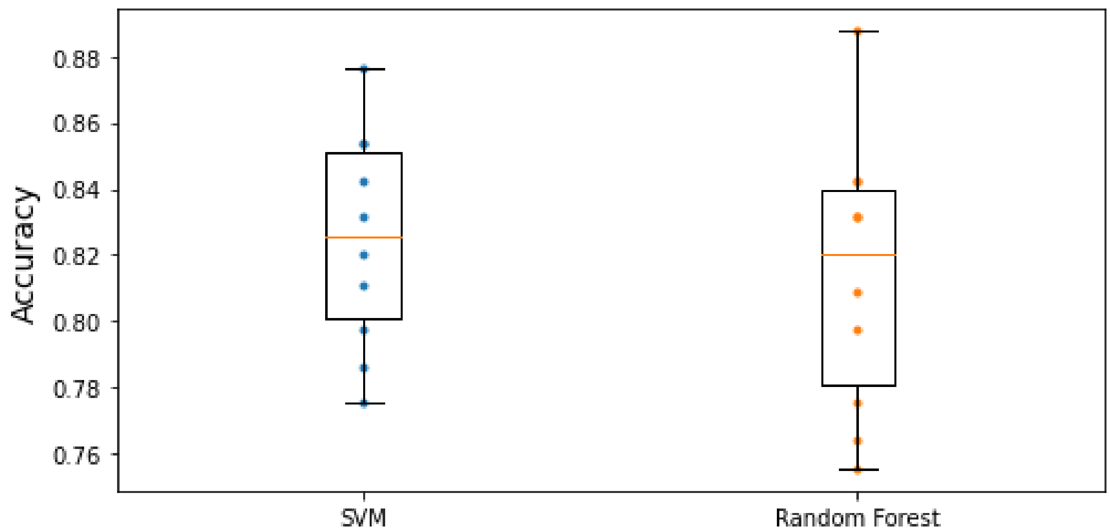
[ ]:
```
from sklearn.model_selection import cross_val_score

forest_scores = cross_val_score(forest_clf, X_train, y_train, cv=10)
forest_scores.mean()
```

[ ]:
```
0.8137578027465668
```

[ ]:
```
from sklearn.svm import SVC

svm_clf = SVC(gamma="auto")
svm_scores = cross_val_score(svm_clf, X_train, y_train, cv=10)
svm_scores.mean()
```

[ ]:
```
0.8249313358302123
```

[ ]:
```
import matplotlib.pyplot as plt

plt.figure(figsize=(8, 4))
plt.plot([1]*10, svm_scores, ".")
plt.plot([2]*10, forest_scores, ".")
plt.boxplot([svm_scores, forest_scores], labels=("SVM","Random Forest"))
plt.ylabel("Accuracy", fontsize=14)
plt.show()
```



[ ]:
```
train_data["AgeBucket"] = train_data["Age"] // 15 * 15
train_data[["AgeBucket", "Survived"]].groupby(['AgeBucket']).mean()
```

[ ]:
```
           Survived
AgeBucket
0.0        0.576923
15.0       0.362745
30.0       0.423256
45.0       0.404494
60.0       0.240000
75.0       1.000000
```

[ ]:
```
train_data["RelativesOnboard"] = train_data["SibSp"] + train_data["Parch"]
train_data[["RelativesOnboard", "Survived"]].groupby(['RelativesOnboard']).
 ↪mean()
```

[ ]:
```
                  Survived
RelativesOnboard
0                 0.303538
1                 0.552795
2                 0.578431
3                 0.724138
4                 0.200000
5                 0.136364
6                 0.333333
7                 0.000000
10                0.000000
```

6.4 4. Spam classifier

First, let's fetch the data:

```
[ ]: import os
     import tarfile
     import urllib.request

     DOWNLOAD_ROOT = "http://spamassassin.apache.org/old/publiccorpus/"
     HAM_URL = DOWNLOAD_ROOT + "20030228_easy_ham.tar.bz2"
     SPAM_URL = DOWNLOAD_ROOT + "20030228_spam.tar.bz2"
     SPAM_PATH = os.path.join("datasets", "spam")

     def fetch_spam_data(ham_url=HAM_URL, spam_url=SPAM_URL, spam_path=SPAM_PATH):
         if not os.path.isdir(spam_path):
             os.makedirs(spam_path)
         for filename, url in (("ham.tar.bz2", ham_url), ("spam.tar.bz2", spam_url)):
             path = os.path.join(spam_path, filename)
             if not os.path.isfile(path):
                 urllib.request.urlretrieve(url, path)
             tar_bz2_file = tarfile.open(path)
             tar_bz2_file.extractall(path=spam_path)
             tar_bz2_file.close()
```

```
[ ]: fetch_spam_data()
```

Next, let's load all the emails:

```
[ ]: HAM_DIR = os.path.join(SPAM_PATH, "easy_ham")
     SPAM_DIR = os.path.join(SPAM_PATH, "spam")
     ham_filenames = [name for name in sorted(os.listdir(HAM_DIR)) if len(name) > 20]
     spam_filenames = [name for name in sorted(os.listdir(SPAM_DIR)) if len(name) >␣
      ↪20]
```

```
[ ]: len(ham_filenames)
```

```
[ ]: 2500
```

```
[ ]: len(spam_filenames)
```

```
[ ]: 500
```

We can use Python's `email` module to parse these emails (this handles headers, encoding, and so on):

```
[ ]: import email
     import email.policy

     def load_email(is_spam, filename, spam_path=SPAM_PATH):
```

```
    directory = "spam" if is_spam else "easy_ham"
    with open(os.path.join(spam_path, directory, filename), "rb") as f:
        return email.parser.BytesParser(policy=email.policy.default).parse(f)
```

[ ]:
```
ham_emails = [load_email(is_spam=False, filename=name) for name in␣
 ↪ham_filenames]
spam_emails = [load_email(is_spam=True, filename=name) for name in␣
 ↪spam_filenames]
```

Let's look at one example of ham and one example of spam, to get a feel of what the data looks like:

[ ]:
```
print(ham_emails[1].get_content().strip())
```

```
Martin A posted:
Tassos Papadopoulos, the Greek sculptor behind the plan, judged that the
 limestone of Mount Kerdylio, 70 miles east of Salonika and not far from the
 Mount Athos monastic community, was ideal for the patriotic sculpture.

 As well as Alexander's granite features, 240 ft high and 170 ft wide, a
 museum, a restored amphitheatre and car park for admiring crowds are
planned
---------------------
So is this mountain limestone or granite?
If it's limestone, it'll weather pretty fast.

------------------------ Yahoo! Groups Sponsor ---------------------~-->
4 DVDs Free +s&p Join Now
http://us.click.yahoo.com/pt6YBB/NXiEAA/mG3HAA/7gSolB/TM
---------------------------------------------------------------------~->

To unsubscribe from this group, send an email to:
forteana-unsubscribe@egroups.com



Your use of Yahoo! Groups is subject to http://docs.yahoo.com/info/terms/
```

[ ]:
```
print(spam_emails[6].get_content().strip())
```

```
Help wanted.  We are a 14 year old fortune 500 company, that is
growing at a tremendous rate.  We are looking for individuals who
want to work from home.

This is an opportunity to make an excellent income.  No experience
is required.  We will train you.

So if you are looking to be employed from home with a career that has
```

```
vast opportunities, then go:

http://www.basetel.com/wealthnow

We are looking for energetic and self motivated people.  If that is you
than click on the link and fill out the form, and one of our
employement specialist will contact you.

To be removed from our link simple go to:

http://www.basetel.com/remove.html


4139vOLW7-758DoDY1425FRhM1-764SMFc8513fCsLl40
```

Some emails are actually multipart, with images and attachments (which can have their own attachments). Let's look at the various types of structures we have:

[ ]:
```python
def get_email_structure(email):
    if isinstance(email, str):
        return email
    payload = email.get_payload()
    if isinstance(payload, list):
        return "multipart({})".format(", ".join([
            get_email_structure(sub_email)
            for sub_email in payload
        ]))
    else:
        return email.get_content_type()
```

[ ]:
```python
from collections import Counter

def structures_counter(emails):
    structures = Counter()
    for email in emails:
        structure = get_email_structure(email)
        structures[structure] += 1
    return structures
```

[ ]:
```python
structures_counter(ham_emails).most_common()
```

[ ]:
```
[('text/plain', 2408),
 ('multipart(text/plain, application/pgp-signature)', 66),
 ('multipart(text/plain, text/html)', 8),
 ('multipart(text/plain, text/plain)', 4),
 ('multipart(text/plain)', 3),
 ('multipart(text/plain, application/octet-stream)', 2),
 ('multipart(text/plain, text/enriched)', 1),
```

```
 ('multipart(text/plain, application/ms-tnef, text/plain)', 1),
 ('multipart(multipart(text/plain, text/plain, text/plain), application/pgp-
signature)',
  1),
 ('multipart(text/plain, video/mng)', 1),
 ('multipart(text/plain, multipart(text/plain))', 1),
 ('multipart(text/plain, application/x-pkcs7-signature)', 1),
 ('multipart(text/plain, multipart(text/plain, text/plain),
text/rfc822-headers)',
  1),
 ('multipart(text/plain, multipart(text/plain, text/plain),
multipart(multipart(text/plain, application/x-pkcs7-signature)))',
  1),
 ('multipart(text/plain, application/x-java-applet)', 1)]
```

[ ]:
```
structures_counter(spam_emails).most_common()
```

[ ]:
```
[('text/plain', 218),
 ('text/html', 183),
 ('multipart(text/plain, text/html)', 45),
 ('multipart(text/html)', 20),
 ('multipart(text/plain)', 19),
 ('multipart(multipart(text/html))', 5),
 ('multipart(text/plain, image/jpeg)', 3),
 ('multipart(text/html, application/octet-stream)', 2),
 ('multipart(text/plain, application/octet-stream)', 1),
 ('multipart(text/html, text/plain)', 1),
 ('multipart(multipart(text/html), application/octet-stream, image/jpeg)', 1),
 ('multipart(multipart(text/plain, text/html), image/gif)', 1),
 ('multipart/alternative', 1)]
```

It seems that the ham emails are more often plain text, while spam has quite a lot of HTML. Moreover, quite a few ham emails are signed using PGP, while no spam is. In short, it seems that the email structure is useful information to have.

Now let's take a look at the email headers:

[ ]:
```
for header, value in spam_emails[0].items():
    print(header,":",value)
```

```
Return-Path : <12a1mailbot1@web.de>
Delivered-To : zzzz@localhost.spamassassin.taint.org
Received : from localhost (localhost [127.0.0.1])       by
phobos.labs.spamassassin.taint.org (Postfix) with ESMTP id 136B943C32        for
<zzzz@localhost>; Thu, 22 Aug 2002 08:17:21 -0400 (EDT)
Received : from mail.webnote.net [193.120.211.219]      by localhost with POP3
(fetchmail-5.9.0)        for zzzz@localhost (single-drop); Thu, 22 Aug 2002
13:17:21 +0100 (IST)
```

```
Received : from dd_it7 ([210.97.77.167])        by webnote.net (8.9.3/8.9.3)
with ESMTP id NAA04623     for <zzzz@spamassassin.taint.org>; Thu, 22 Aug 2002
13:09:41 +0100
From : 12a1mailbot1@web.de
Received : from r-smtp.korea.com - 203.122.2.197 by dd_it7  with Microsoft
SMTPSVC(5.5.1775.675.6);      Sat, 24 Aug 2002 09:42:10 +0900
To : dcek1a1@netsgo.com
Subject : Life Insurance - Why Pay More?
Date : Wed, 21 Aug 2002 20:31:57 -1600
MIME-Version : 1.0
Message-ID : <0103c1042001882DD_IT7@dd_it7>
Content-Type : text/html; charset="iso-8859-1"
Content-Transfer-Encoding : quoted-printable
```

There's probably a lot of useful information in there, such as the sender's email address (12a1mailbot1@web.de looks fishy), but we will just focus on the `Subject` header:

[ ]:
```python
spam_emails[0]["Subject"]
```

[ ]:
```
'Life Insurance - Why Pay More?'
```

Okay, before we learn too much about the data, let's not forget to split it into a training set and a test set:

[ ]:
```python
import numpy as np
from sklearn.model_selection import train_test_split

X = np.array(ham_emails + spam_emails, dtype=object)
y = np.array([0] * len(ham_emails) + [1] * len(spam_emails))

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,␣
↪random_state=42)
```

[ ]:
```python
import re
from html import unescape

def html_to_plain_text(html):
    text = re.sub('<head.*?>.*?</head>', '', html, flags=re.M | re.S | re.I)
    text = re.sub('<a\s.*?>', ' HYPERLINK ', text, flags=re.M | re.S | re.I)
    text = re.sub('<.*?>', '', text, flags=re.M | re.S)
    text = re.sub(r'(\s*\n)+', '\n', text, flags=re.M | re.S)
    return unescape(text)
```

Let's see if it works. This is HTML spam:

[ ]:
```python
html_spam_emails = [email for email in X_train[y_train==1]
                    if get_email_structure(email) == "text/html"]
sample_html_spam = html_spam_emails[7]
print(sample_html_spam.get_content().strip()[:1000], "...")
```

```
<HTML><HEAD><TITLE></TITLE><META http-equiv="Content-Type" content="text/html;
charset=windows-1252"><STYLE>A:link {TEX-DECORATION: none}A:active {TEXT-
DECORATION: none}A:visited {TEXT-DECORATION: none}A:hover {COLOR: #0033ff; TEXT-
DECORATION: underline}</STYLE><META content="MSHTML 6.00.2713.1100"
name="GENERATOR"></HEAD>
<BODY text="#000000" vLink="#0033ff" link="#0033ff" bgColor="#CCCC99"><TABLE
borderColor="#660000" cellSpacing="0" cellPadding="0" border="0"
width="100%"><TR><TD bgColor="#CCCC99" valign="top" colspan="2" height="27">
<font size="6" face="Arial, Helvetica, sans-serif" color="#660000">
<b>OTC</b></font></TD></TR><TR><TD height="2" bgcolor="#6a694f">
<font size="5" face="Times New Roman, Times, serif" color="#FFFFFF">
<b> Newsletter</b></font></TD><TD height="2" bgcolor="#6a694f"><div
align="right"><font color="#FFFFFF">
<b>Discover Tomorrow's Winners </b></font></div></TD></TR><TR><TD
height="25" colspan="2" bgcolor="#CCCC99"><table width="100%" border="0"  …
```

[ ]:
```python
print(html_to_plain_text(sample_html_spam.get_content())[:1000], "...")
```

```
OTC
 Newsletter
Discover Tomorrow's Winners
For Immediate Release
Cal-Bay (Stock Symbol: CBYI)
Watch for analyst "Strong Buy Recommendations" and several advisory newsletters
picking CBYI.  CBYI has filed to be traded on the OTCBB, share prices
historically INCREASE when companies get listed on this larger trading exchange.
CBYI is trading around 25 cents and should skyrocket to $2.66 - $3.25 a share in
the near future.
Put CBYI on your watch list, acquire a position TODAY.
REASONS TO INVEST IN CBYI
A profitable company and is on track to beat ALL earnings estimates!
One of the FASTEST growing distributors in environmental & safety equipment
instruments.
Excellent management team, several EXCLUSIVE contracts.  IMPRESSIVE client list
including the U.S. Air Force, Anheuser-Busch, Chevron Refining and Mitsubishi
Heavy Industries, GE-Energy & Environmental Research.
RAPIDLY GROWING INDUSTRY
Industry revenues exceed $900 million, estimates indicate that there could be as
much as $25 billi …
```

[ ]:
```python
def email_to_text(email):
    html = None
    for part in email.walk():
        ctype = part.get_content_type()
        if not ctype in ("text/plain", "text/html"):
            continue
```

```python
        try:
            content = part.get_content()
        except: # in case of encoding issues
            content = str(part.get_payload())
        if ctype == "text/plain":
            return content
        else:
            html = content
    if html:
        return html_to_plain_text(html)
```

[ ]:
```python
print(email_to_text(sample_html_spam)[:100], "...")
```

```
OTC
 Newsletter
Discover Tomorrow's Winners
For Immediate Release
Cal-Bay (Stock Symbol: CBYI)
Wat …
```

[ ]:
```python
try:
    import nltk

    stemmer = nltk.PorterStemmer()
    for word in ("Computations", "Computation", "Computing", "Computed",␣
 ↪"Compute", "Compulsive"):
        print(word, "=>", stemmer.stem(word))
except ImportError:
    print("Error: stemming requires the NLTK module.")
    stemmer = None
```

```
Computations => comput
Computation => comput
Computing => comput
Computed => comput
Compute => comput
Compulsive => compuls
```

We will also need a way to replace URLs with the word "URL". For this, we could use hard core regular expressions but we will just use the urlextract library. You can install it with the following command (don't forget to activate your virtualenv first; if you don't have one, you will likely need administrator rights, or use the `--user` option):

```
$ pip3 install urlextract
```

[ ]:
```python
# if running this notebook on Colab or Kaggle, we just pip install urlextract
if IS_COLAB or IS_KAGGLE:
```

```
%pip install -q -U urlextract
```

```
[ ]: try:
    import urlextract # may require an Internet connection to download root␣
↪domain names

    url_extractor = urlextract.URLExtract()
    print(url_extractor.find_urls("Will it detect github.com and https://youtu.
↪be/7Pq-S557XQU?t=3m32s"))
except ImportError:
    print("Error: replacing URLs requires the urlextract module.")
    url_extractor = None
```

```
['github.com', 'https://youtu.be/7Pq-S557XQU?t=3m32s']
```

```
[ ]: from sklearn.base import BaseEstimator, TransformerMixin

class EmailToWordCounterTransformer(BaseEstimator, TransformerMixin):
    def __init__(self, strip_headers=True, lower_case=True,␣
↪remove_punctuation=True,
                 replace_urls=True, replace_numbers=True, stemming=True):
        self.strip_headers = strip_headers
        self.lower_case = lower_case
        self.remove_punctuation = remove_punctuation
        self.replace_urls = replace_urls
        self.replace_numbers = replace_numbers
        self.stemming = stemming
    def fit(self, X, y=None):
        return self
    def transform(self, X, y=None):
        X_transformed = []
        for email in X:
            text = email_to_text(email) or ""
            if self.lower_case:
                text = text.lower()
            if self.replace_urls and url_extractor is not None:
                urls = list(set(url_extractor.find_urls(text)))
                urls.sort(key=lambda url: len(url), reverse=True)
                for url in urls:
                    text = text.replace(url, " URL ")
            if self.replace_numbers:
                text = re.sub(r'\d+(?:\.\d*)?(?:[eE][+-]?\d+)?', 'NUMBER', text)
            if self.remove_punctuation:
                text = re.sub(r'\W+', ' ', text, flags=re.M)
            word_counts = Counter(text.split())
            if self.stemming and stemmer is not None:
                stemmed_word_counts = Counter()
```

```
                for word, count in word_counts.items():
                    stemmed_word = stemmer.stem(word)
                    stemmed_word_counts[stemmed_word] += count
                word_counts = stemmed_word_counts
            X_transformed.append(word_counts)
        return np.array(X_transformed)
```

Let's try this transformer on a few emails:

[ ]:
```
X_few = X_train[:3]
X_few_wordcounts = EmailToWordCounterTransformer().fit_transform(X_few)
X_few_wordcounts
```

[ ]:
```
array([Counter({'chuck': 1, 'murcko': 1, 'wrote': 1, 'stuff': 1, 'yawn': 1, 'r':
1}),
       Counter({'the': 11, 'of': 9, 'and': 8, 'all': 3, 'christian': 3, 'to': 3,
'by': 3, 'jefferson': 2, 'i': 2, 'have': 2, 'superstit': 2, 'one': 2, 'on': 2,
'been': 2, 'ha': 2, 'half': 2, 'rogueri': 2, 'teach': 2, 'jesu': 2, 'some': 1,
'interest': 1, 'quot': 1, 'url': 1, 'thoma': 1, 'examin': 1, 'known': 1, 'word':
1, 'do': 1, 'not': 1, 'find': 1, 'in': 1, 'our': 1, 'particular': 1, 'redeem':
1, 'featur': 1, 'they': 1, 'are': 1, 'alik': 1, 'found': 1, 'fabl': 1,
'mytholog': 1, 'million': 1, 'innoc': 1, 'men': 1, 'women': 1, 'children': 1,
'sinc': 1, 'introduct': 1, 'burnt': 1, 'tortur': 1, 'fine': 1, 'imprison': 1,
'what': 1, 'effect': 1, 'thi': 1, 'coercion': 1, 'make': 1, 'world': 1, 'fool':
1, 'other': 1, 'hypocrit': 1, 'support': 1, 'error': 1, 'over': 1, 'earth': 1,
'six': 1, 'histor': 1, 'american': 1, 'john': 1, 'e': 1, 'remsburg': 1,
'letter': 1, 'william': 1, 'short': 1, 'again': 1, 'becom': 1, 'most': 1,
'pervert': 1, 'system': 1, 'that': 1, 'ever': 1, 'shone': 1, 'man': 1, 'absurd':
1, 'untruth': 1, 'were': 1, 'perpetr': 1, 'upon': 1, 'a': 1, 'larg': 1, 'band':
1, 'dupe': 1, 'import': 1, 'led': 1, 'paul': 1, 'first': 1, 'great': 1,
'corrupt': 1}),
       Counter({'url': 4, 's': 3, 'group': 3, 'to': 3, 'in': 2, 'forteana': 2,
'martin': 2, 'an': 2, 'and': 2, 'we': 2, 'is': 2, 'yahoo': 2, 'unsubscrib': 2,
'y': 1, 'adamson': 1, 'wrote': 1, 'for': 1, 'altern': 1, 'rather': 1, 'more': 1,
'factual': 1, 'base': 1, 'rundown': 1, 'on': 1, 'hamza': 1, 'career': 1,
'includ': 1, 'hi': 1, 'belief': 1, 'that': 1, 'all': 1, 'non': 1, 'muslim': 1,
'yemen': 1, 'should': 1, 'be': 1, 'murder': 1, 'outright': 1, 'know': 1, 'how':
1, 'unbias': 1, 'memri': 1, 'don': 1, 't': 1, 'html': 1, 'rob': 1, 'sponsor': 1,
'number': 1, 'dvd': 1, 'free': 1, 'p': 1, 'join': 1, 'now': 1, 'from': 1, 'thi':
1, 'send': 1, 'email': 1, 'egroup': 1, 'com': 1, 'your': 1, 'use': 1, 'of': 1,
'subject': 1})],
      dtype=object)
```

[ ]:
```
from scipy.sparse import csr_matrix

class WordCounterToVectorTransformer(BaseEstimator, TransformerMixin):
    def __init__(self, vocabulary_size=1000):
```

```
        self.vocabulary_size = vocabulary_size
    def fit(self, X, y=None):
        total_count = Counter()
        for word_count in X:
            for word, count in word_count.items():
                total_count[word] += min(count, 10)
        most_common = total_count.most_common()[:self.vocabulary_size]
        self.vocabulary_ = {word: index + 1 for index, (word, count) in␣
↪enumerate(most_common)}
        return self
    def transform(self, X, y=None):
        rows = []
        cols = []
        data = []
        for row, word_count in enumerate(X):
            for word, count in word_count.items():
                rows.append(row)
                cols.append(self.vocabulary_.get(word, 0))
                data.append(count)
        return csr_matrix((data, (rows, cols)), shape=(len(X), self.
↪vocabulary_size + 1))
```

[ ]:
```
vocab_transformer = WordCounterToVectorTransformer(vocabulary_size=10)
X_few_vectors = vocab_transformer.fit_transform(X_few_wordcounts)
X_few_vectors
```

[ ]:
```
<3x11 sparse matrix of type '<class 'numpy.longlong'>'
        with 20 stored elements in Compressed Sparse Row format>
```

[ ]:
```
X_few_vectors.toarray()
```

[ ]:
```
array([[ 6,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0],
       [99, 11,  9,  8,  3,  1,  3,  1,  3,  2,  3],
       [67,  0,  1,  2,  3,  4,  1,  2,  0,  1,  0]], dtype=int64)
```

[ ]:
```
vocab_transformer.vocabulary_
```

[ ]:
```
{'the': 1,
 'of': 2,
 'and': 3,
 'to': 4,
 'url': 5,
 'all': 6,
 'in': 7,
 'christian': 8,
 'on': 9,
 'by': 10}
```

We are now ready to train our first spam classifier! Let's transform the whole dataset:

[ ]:
```
from sklearn.pipeline import Pipeline

preprocess_pipeline = Pipeline([
    ("email_to_wordcount", EmailToWordCounterTransformer()),
    ("wordcount_to_vector", WordCounterToVectorTransformer()),
])

X_train_transformed = preprocess_pipeline.fit_transform(X_train)
```

Note: to be future-proof, we set `solver="lbfgs"` since this will be the default value in Scikit-Learn 0.22.

[ ]:
```
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import cross_val_score

log_clf = LogisticRegression(solver="lbfgs", max_iter=1000, random_state=42)
score = cross_val_score(log_clf, X_train_transformed, y_train, cv=3, verbose=3)
score.mean()
```

```
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done   1 out of   1 | elapsed:    0.1s remaining:    0.0s

[CV]  …
[CV] … , score=0.981, total=   0.1s
[CV]  …
[CV] … , score=0.985, total=   0.2s
[CV]  …
[CV] … , score=0.991, total=   0.2s

[Parallel(n_jobs=1)]: Done   2 out of   2 | elapsed:    0.3s remaining:    0.0s
[Parallel(n_jobs=1)]: Done   3 out of   3 | elapsed:    0.5s finished
```

[ ]: 0.9858333333333333

[ ]:
```
from sklearn.metrics import precision_score, recall_score

X_test_transformed = preprocess_pipeline.transform(X_test)

log_clf = LogisticRegression(solver="lbfgs", max_iter=1000, random_state=42)
log_clf.fit(X_train_transformed, y_train)

y_pred = log_clf.predict(X_test_transformed)

print("Precision: {:.2f}%".format(100 * precision_score(y_test, y_pred)))
print("Recall: {:.2f}%".format(100 * recall_score(y_test, y_pred)))
```

```
Precision: 95.88%
Recall: 97.89%
```

```
[ ]:
```