Google Street View Character Identification

Matthew Shin, yongshin@usc.edu

December 4, 2017

1. Abstract

Image processing is a popular application of machine learning today, from facial recognition to robot vision. In this project, I explored one version of such an application: the classification of images of numbers and letters seen in the real world. In a Python environment, I used classifiers within the Scikit-Learn library to train and validate my learning program. Results from the final hypothesis resulting from the learning process show that an ensemble method (soft voting) of three machine learning models (support vector machine, k-nearest neighbors, and random forest) learns the best from the training data.

2. Problem statement and goals

This project is based off of the Kaggle competition, “First Steps With Julia”, which seeks to find an algorithm capable of classifying characters embedded within natural images. The image data are gathered from Google Street View and contain single digits or letters [1]. The original intent of this project was to use machine learning packages within the Julia programming language to find a supervised classification model that will optimize prediction accuracy on future Google Street View data. The goals were to: (1) get familiarized with the Julia language, (2) apply machine learning techniques from the course, and (3) explore new learning algorithms.

Throughout the development of this project, the majority and spirit of the problem statement and goals remained the same. The only change from the time of the original project proposal was the programming language of implementation. Due to Julia being a relatively new language still in development, the language was changing too rapidly even for package developers to react to. For this reason, I decided to use Python and its machine learning libraries, which are much more stable. As I had barely any substantial prior experience with Python, I achieved the amended first goal of familiarization with a programming language.

3. Literature review

In addition to knowledge from studying the models that were discussed in the course EE 660, much of the methodology and code samples used in the implementation of this project owe their credit to the book, *Hands-On Machine Learning with Scikit-Learn and TensorFlow: Concepts, Tools, and Techniques to Build Intelligent Systems* by Aurélien Géron. In it, the author introduces the readers to the different categories of machine learning, walks them through typical machine learning procedures and techniques, and goes into detail about several machine learning algorithms and their implementation in Python. For this project, I focused on traditional classification algorithms, but the author also goes into deep learning concepts and implementation using neural networks with Google’s TensorFlow [2].

In addition to Géron's book, I skimmed through two of the Kaggle contestants' methods of using convolutional neural networks (CNNs) on the classification problem through Python. Both *Using deep learning to read street signs* by Florian Muellerklein and *Kaggle First Steps With Julia (Chars74k): First Place using Convolutional Neural Networks* by Fabien Tencé outline the use of data augmentation with a CNN to achieve relatively high accuracy scores [3, 4].

4. Prior and related work

While this project was not based on any prior or related work, I had previously done an unrelated classification project on credit score data using MATLAB for my mathematical pattern recognition class. The reason I bring up the past project is that I had an unsuccessful and unfavorable result, with the final classifier performing no better than random guessing for the two-class classification task. In addition to the goals listed in Section 2, I hoped to achieve results that were at least significantly better than random guesses.

5. Project formulation and setup

The model for my learning algorithm was as follows: The processed data from the Kaggle competition was fed into the algorithm, which consists of a feature reduction component, a cross-validation component, a training component, and a testing component. **Figure 1** illustrates the entire process, from data gathering to prediction.

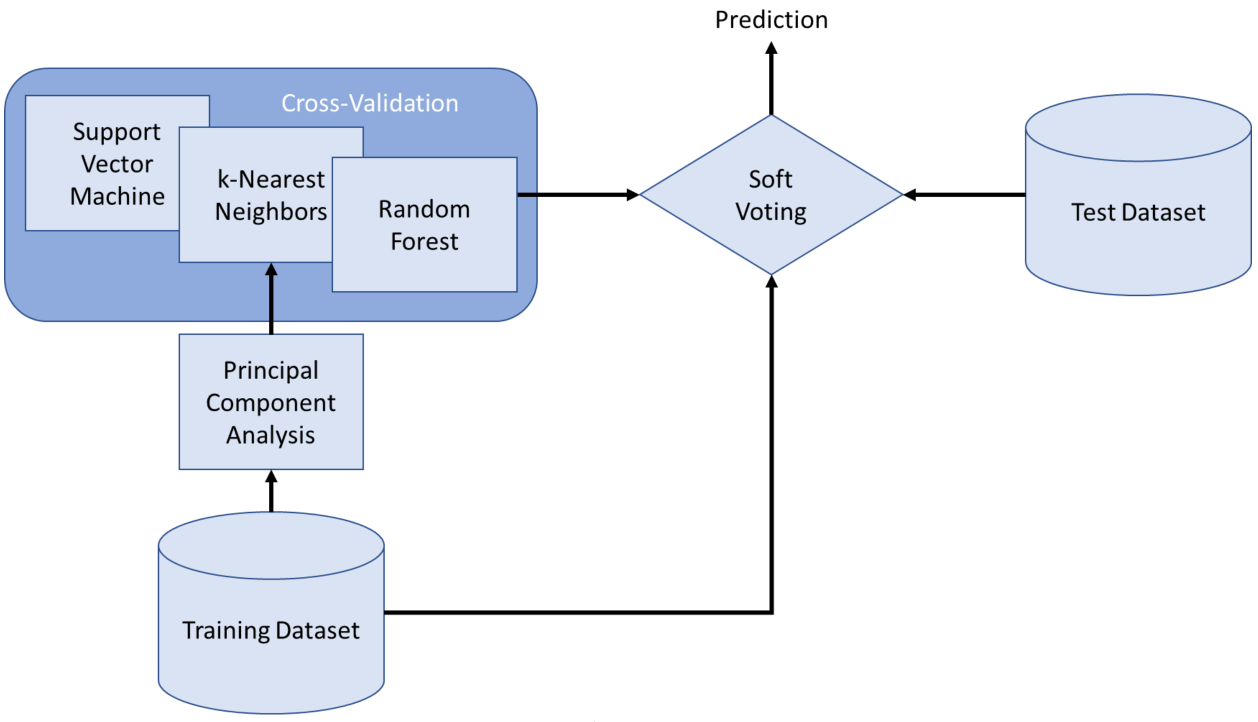


Figure 1

Each part of the leaning model will be explained in detail in the next two sections, including each classification parameter. Again, this model was based on techniques from Géron's guide.

6. Methodology

The overall procedure used in this project was the following: pre-processing and feature reduction, cross-validation and model selection, training and test validation. The hypothesis set was a superset of three hypothesis subsets, each corresponding to the classification method. Although one hypothesis was chosen from each subset, the final training and testing was carried out on all three final hypotheses, per the requirements of the voting classification algorithm.

The total number of samples was 6,283, which was divided up into a training and test set. The training set included a random partition of 5,000 samples, and the other 1,283 samples made up the test set. A k-fold cross-validation where $k = 5$ was run on 4,000 training and 1,000 validation samples at each iteration (explained further in Section 7.4). Training was then run on all 5,000 training samples using the results of the cross-validation algorithms and an ensemble training technique called soft voting classification. Finally, the test set was introduced, and all 1283 test samples were used to calculate the out-of-sample accuracy estimate following prediction with the trained voting classifier.

7. Implementation

In this section, I will expand on the methodology described above. As previously mentioned, the image classification was implemented using Python and the Scikit-Learn library for machine learning. I will describe, in detail, the steps I took to perform the learning process.

7.1. Feature space

The datasets I used originated from the Kaggle online competition, "First Steps With Julia". The `trainResized` dataset included 6,283 color bitmap (.bmp) image files [1]. Therefore, each feature (i.e., pixel) was an array of 24-bit RGB data. **Table 1** shows the feature name, type of each feature (next to the feature name), and the ranges for each feature (inside each feature instance).

Table 1

n\d	1 [array(int)]	…	400 [array(int)]
1	[0:255, 0:255, 0:255]	…	[0:255, 0:255, 0:255]
…	…	…	…
6,283	[0:255, 0:255, 0:255]	…	[0:255, 0:255, 0:255]

Each pixel is an array of three integers that denote the amount of red, green, and blue it is made up of. The range of each integer is from 0 to 255. As each image has dimensions 20×20 pixels, there are 400 features in total per sample.

The `trainLabels` dataset is a comma separated value (.csv) file. As shown in **Table 2**, it contains the true labels (class) for each of the 6,283 samples in the `trainResized` dataset. Each instance is a string in the format *'char'* where *char* represents a single character from 0 to 9, A to Z or, a to z.

Table 2

n	Class [str('char')]
1	['0:9/A:Z/a:z']
…	…
6,283	['0:9/A:Z/a:z']

The `trainResized` dataset was loaded into Python and denoted `X` while the `trainLabels` dataset was denoted `y` in Python. While the training dataset from the Kaggle source contained the class labels, the test dataset `testResized`, did not come with labels. Therefore, I divided sets `X` and `y` into training (`Xtrain`, `ytrain`) and test (`Xtest`, `ytest`) sets after applying a random permutation of their ordering [2]. Again, the training set had 5,000 samples, and the test set had 1,283.

7.2. Pre-processing and feature extraction

Luckily, not much pre-processing was needed on the dataset. Unlike the `train` dataset, which is also available on Kaggle, the dataset I used was already pre-processed to an extent, that the files did not require resizing to a standardized size (i.e., 20 × 20 pixels). The only pre-processing that was done on the feature set was a conversion to decimal grayscale. First, `X` was converted to grayscale, which meant that each instance of a feature was an integer value from 0 to 255. Then the value was divided by the maximum value, 255, to arrive at the decimal grayscale value of type float from 0 to 1.

Similarly, the pre-processing on the class set was rather simple. As the classifiers I used for this project took integer values for the classes, I converted all the character classes into an integer equivalent. This was carried out using the built-in Python function `ord()` on `y`, which converts an 8-bit string (character) into an integer representing its Unicode code, after dropping the '' symbols before and after the character class.

Feature extraction was not necessary in this classification project, as every pixel was originally counted as a feature. However, after `X` and `y` were pre-processed, I performed data reduction to reduce the feature size and speed up the training algorithms. The data reduction technique I used was principal component analysis (PCA), a type of unsupervised learning which projects the data onto a subspace of the total feature space. It was implemented through Scikit-Learn's PCA feature. The result of running the PCA was a reduction of features used from 400 to 100. (I could have used as little as 93 features while preserving at least 95% of the variance in the training data [2], but I arbitrarily decided to round up to the nearest 100.) After running the PCA on the training data `Xtrain`, I transformed both `Xtrain` and `Xtest` on the results to reduce the feature set and output `Xtrain_red` and `Xtest_red`, which were used for the training process [5].

7.3. Training process

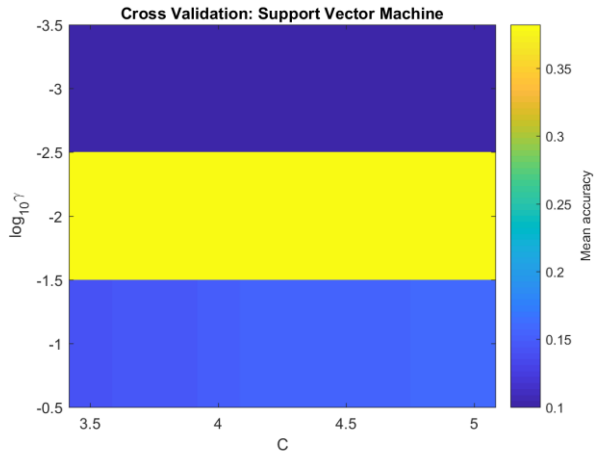
I chose to use three traditional classifiers for the training process for the character identification problem: support vector machine (SVM), k-nearest neighbors (kNN), and random forest (RF). The first was chosen because of its widespread popularity as a flexible model, especially with its use of kernel functions. Also, it was used as an introductory classifier for the similar MNIST dataset [6]. The other two were chosen as they were used in the original Kaggle competition as benchmark algorithms [1]. As such, it was convenient to see how my training processed fared in comparison. Furthermore, kNN was a model I studied in a previous class (EE 559) and RF was a model that this current class had used.

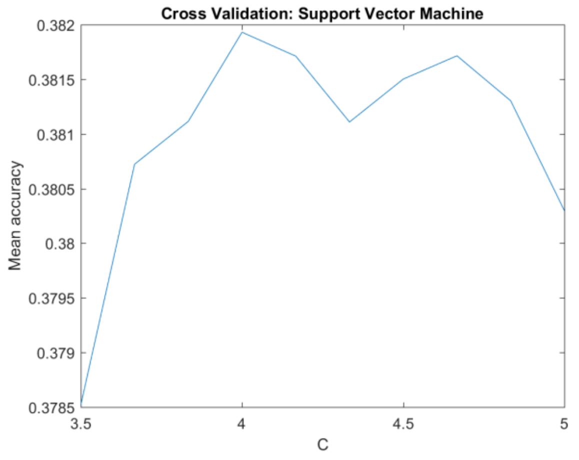
As with PCA, training for these classifiers were done on Python using the Scikit-Learn library. The parameters for SVM (`SVC`) were `gamma` and `C`, as I decided to use the Gaussian kernel on the highly nonlinear decision boundary functions [7]. The parameters used for kNN (`KNeighborsClassifier`) were `n_neighbors` (or *k*) and `weights` (which specify how the neighbors are weighted during prediction) [8]. Finally, RF (`RandomForestClassifier`) only used one parameter `n_estimators` (or the number of trees) [9].

The training process utilized 5,000 samples (N = 5,000) on the reduced 100 dimensions (D = 100). Heuristically, this is an adequate ratio in terms of VC complexity as N is 50 times greater than D, while it is acceptable to have $N > 10D$ [10]. In fact, even the pre-PCA dimensions of D = 400 would have been okay, as it met the heuristic 10:1 ratio. Furthermore, over- and underfitting were mostly addressed with choosing the parameter(s) using cross-validation. For instance, the parameter `C` provides regularization for SVM and the parameter n_neighbors for kNN. I would have like to play around with the regularization parameters for RF, such as `max_depth`; however, as `RandomForestClassifier` was running prohibitively slow, I was not able to include them.

7.4. Testing, validation, and model selection

For testing and validation, I ran 5-fold cross-validation (using `cross_val_score`) and selected the parameter(s) for the classification model based on the averaged accuracy score (found in a grid search approach using nested for loops) [2]. I chose to divide the training set into five partitions for cross-validation because that meant the validation set would make up 20% of the total training set. This 20:80 ratio is similar to the ratio between the test set and the total training set (around 20.4:70.6).

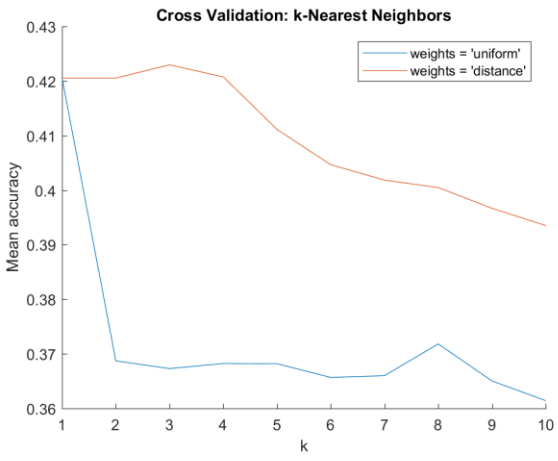


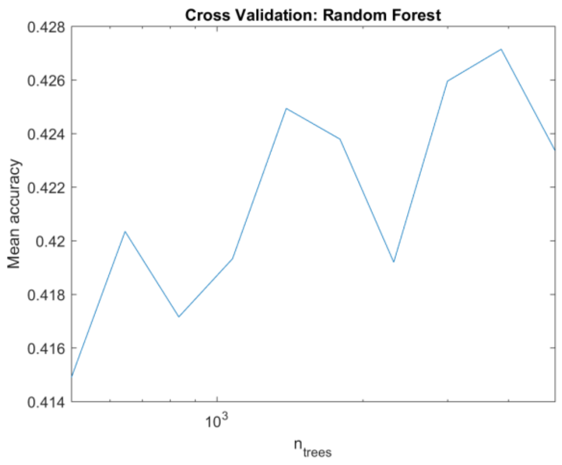


Figures 2 and 3

The results from the cross-validation runs are shown in the four figures below. **Figure 2** shows a 3 × 10 colormap of the averaged cross-validation scores, dependent on the values for `gamma` and `C`. It can be seen from the colormap that a `gamma` value of 0.01 (or 10^{-2}) corresponds to the range of highest mean accuracies.

Within that `gamma` value, however, the colormap does a poor job in distinguishing the optimal `C` parameter. **Figure 3** shows that with `gamma` held constant at 0.01, SVM reaches maximum mean cross-validation accuracy when `C` is 4.





Figures 4 and 5

Similarly, **Figure 4** graphs the mean cross-validation scores for kNN, and **Figure 5** does the same for RF. The optimal parameters for kNN is when `weights` and `n_neighbors` are distance and 3, respectively. Local optimality is reached for RF is when `n_estimators` is 3,870.

My original method of model selection was to compare the highest mean cross-validation accuracies of my three models (shown in **Table 3**) and train the entire `Xtrain_red` on the output of the greedily optimal model. In this case RF with the number of trees set to 3,870 would have been chosen as the final classification hypothesis.

Table 3

	SVM	kNN	RF
Best mean accuracy	0.381930770895	0.422952168775	0.427139296483

However, after some research, I was introduced to a simple ensemble method called the soft voting classifier (`VotingClassifier`), which trains the dataset based on the confidence of independent training processes. Although my classifiers were acting on the same data, their methods were fairly different, so the voting classifier was a worthwhile candidate [2].

8. Final results

As **Table 4** shows, if RF was chosen as my final hypothesis, according to my original model selection method, the out-of-sample estimate for the accuracy would have been 0.445830085737 (note that it has a lower test accuracy compared to kNN). However, as previously mentioned, I decided to rather use a soft voting classifier on my three best-tuned models from the cross-validation runs.

Table 4

	SVM	kNN	RF	Soft voting
Train accuracy	0.9216	1.0	1.0	1.0
Test accuracy	0.388932190179	0.455183164458	0.445830085737	0.479345284489

Surprisingly, despite all three classifiers reporting their best mean cross-validation accuracies of less than 0.43, the final out-of-sample accuracy estimate was 0.479345284489 for the soft voting classifier. In fact, accuracy for voting outperformed the test accuracies of all SVM, kNN, and RF by at least 2.3%.

The final voting model was trained only on 5,000 of the original 6,283 samples. But despite the limited training data, the classifier fares relatively well when compared to the benchmarks set in the Kaggle leadership board. The best RF classifier just barely missed the 0.44721 score listed. However, the optimal kNN classifier surpassed the 0.42572 benchmark, and the soft voting classifier performed better than both benchmarks [1].

9. Interpretation

Looking at the nominal accuracy figures, none of the classifiers seem to be working all that well (there is a greater than 50% chance of misclassification!), especially when one compares validation results from the same algorithms on the MNIST data [6]. However, it is important to look at the results from the perspective of the problem. First, the Google Street View dataset was trained on 5,000 samples and validated on 1,283, compared to 60,000 and 10,000, respectively. Second, this problem required the classification of samples into 62 classes (10 digits, 26 capital letters, and 26 lowercase letters), compared to 10 (digits only). Finally, classification of multi-colored and unstandardized fonts on a variety of differently shaded backgrounds is a much bigger image processing task, compared to the classification of handwritten, white numbers on a black background.

After reexamining the scope of the problem, the learning results were not too bad, correctly identifying the character in an image almost half of the time. Again, I learned that cleverly combining multiple and varied techniques in an ensemble approach (e.g., voting classifier, boosting, etc.) can result in a better learning algorithm than training on a single, particular method. Furthermore, for this classification problem at least, the kNN classifier yielded the best results per computational time, achieving in about a minute similar accuracy to what the RF classifier got in several hours (see *Output* in Section 13).

10. Contributions of each team member

Due to difficulties stemming from scheduling conflicts and other inconveniences of being a DEN (online) student, this project was completed individually. As such, I was the sole contributor to its formulation, design, execution, and write-up.

11. Summary and conclusions

To reiterate the key findings, results from this project give empirical evidence to how ensemble methods can take lower performance models and output a higher performance program. If I return to this problem, or something like it, I would consider using other models that take advantage of this finding, such as AdaBoost, gradient boosting, and stacking. Furthermore, I am more open to applying concepts from ensemble methods like RF in other parts of the learning process, such as using randomized search for cross-validation on large parameter ranges [2].

Another lesson learned from this project is about time complexity of certain algorithms compared to their effectiveness. For instance, I learned that kNN is a fast and effective model that gives a fair baseline performance, compared to RF, which is much slower with the same level of performance. In the future, I will look at writing up a program for models that are known to be much more powerful (e.g., CNN from Section 3) if I catch myself waiting that long on a mediocre model to run.

12. References

[1] L. Tandalla, "First Steps With Julia", *Kaggle*, 2017. [Online]. Available: https://www.kaggle.com/c/street-view-getting-started-with-julia. [Accessed: 04- Dec- 2017].

[2] A. Géron, *Hands-On Machine Learning with Scikit-Learn and TensorFlow*. Sebastopol, CA: O'Reilly Media, 2017.

[3] F. Muellerklein, "Using deep learning to read street signs", *Florian Muellerklein – Machine learning and statistics*, 2016. [Online]. Available: http://florianmuellerklein.github.io/cnn_streetview/. [Accessed: 04- Dec- 2017].

[4] F. Tencé, "Kaggle First Steps With Julia (Chars74k): First Place using Convolutional Neural Networks", *Ankivil – Machine Learning Experiments*, 2016. [Online]. Available: http://ankivil.com/kaggle-first-steps-with-julia-chars74k-first-place-using-convolutional-neural-networks/. [Accessed: 04- Dec- 2017].

[5] "PCA on train and test datasets: should I run one PCA on train+test or two separate on train and on test?", *Cross Validated*, 2017. [Online]. Available: https://stats.stackexchange.com/q/125172. [Accessed: 04- Dec- 2017].

[6] "Digit Recognizer", *Kaggle*, 2017. [Online]. Available: https://www.kaggle.com/c/digit-recognizer. [Accessed: 04- Dec- 2017].

[7] "sklearn.svm.SVC — scikit-learn 0.19.1 documentation", *Scikit-Learn*, 2017. [Online]. Available: http://scikit-learn.org/stable/modules/generated/sklearn.svm.SVC.html. [Accessed: 04- Dec- 2017].

[8] "sklearn.neighbors.KNeighborsClassifier — scikit-learn 0.19.1 documentation", *Scikit-Learn*, 2017. [Online]. Available: http://scikit-

learn.org/stable/modules/generated/sklearn.neighbors.KNeighborsClassifier.html. [Accessed: 04- Dec- 2017].

[9] “3.2.4.3.1. sklearn.ensemble.RandomForestClassifier — scikit-learn 0.19.1 documentation”, *Scikit-Learn*, 2017. [Online]. Available: http://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html. [Accessed: 04- Dec- 2017].

[10] Y. Abu-Mostafa, M. Magdon-Ismail and H. Lin, *Learning from Data*. AMLbook, 2012, p. 56.

13. Code

Python code:
```
from PIL import Image
import numpy as np
import pandas as pd
import time
import scipy.io as sio
from sklearn.model_selection import cross_val_score
from sklearn.metrics import accuracy_score


# Dataset dimensions
N = 6283
D = 400


# Load feature dataset and convert to grayscale
X = []
for i in range(1, N+1):
    img = Image.open("./trainResized/{}.Bmp".format(str(i)))
    img = img.convert("LA")
    imgData = list(img.getdata())
    img = []
    for value in imgData:
        img.append(value[0]/value[1])
    X.append(img)
X = pd.DataFrame(X)


# Load label dataset
yReal = pd.read_csv("trainLabels.csv", usecols=[1])
yClass = yReal.iloc[0:N]
y = yClass.copy()
for i in range(0, N):
    y.iloc[i] = ord(str(yClass.iloc[i].values)[2])


# Divide training/test sets
Ntrain = 5000
```

```python
def split_dataset(data, trainSize):
    np.random.seed(0)
    testSize = int(len(data) - trainSize)
    shuffled_ind = np.random.permutation(len(data))
    train_ind = shuffled_ind[testSize:]
    test_ind = shuffled_ind[:testSize]
    return data.iloc[train_ind].reset_index(drop=True),
data.iloc[test_ind].reset_index(drop=True)


Xtrain, Xtest = split_dataset(X, Ntrain)
ytrain, ytest = split_dataset(y, Ntrain)
ytrain = ytrain.values.ravel()
ytest = ytest.values.ravel()


# Dimensionality reduction
from sklearn.decomposition import PCA

pca = PCA()
pca.fit(Xtrain)
cumsum = np.cumsum(pca.explained_variance_ratio_)
d = np.argmax(cumsum >= 0.95) + 1
print("PCA results: min dimensions = ", d, ", dimensions used = ", 100)
pca = PCA(n_components=100)
Xtrain_red = pca.fit_transform(Xtrain)
Xtest_red = pca.transform(Xtest)


# Support vector machine classifier
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import StandardScaler
from sklearn.svm import SVC

print("\n***** SUPPORT VECTOR MACHINE *****")
rbf_svm_best_mean = 0.
rbf_svm_best_param = [0., 0.]
cv_mean_acc = np.zeros((3, 10))
i = -1
for g in np.logspace(-3, -1, 3):
    i = i + 1
    j = -1
    for c in np.linspace(3.5, 5, 10):
        j = j + 1
        rbf_svm_clf = Pipeline((
            ("scaler", StandardScaler()),
            ("svm_clf", SVC(kernel="rbf", gamma=g, C=c))
        ))
        time_start = time.clock()
        cv_accuracies = cross_val_score(rbf_svm_clf, Xtrain_red, ytrain,
cv=5, scoring="accuracy")
        time_elapsed = time.clock() - time_start
        cv_mean_acc[i, j] = np.mean(cv_accuracies)
        print("CV mean = ", cv_mean_acc[i, j], ", CV training time = ",
time_elapsed)
        if cv_mean_acc[i, j] > rbf_svm_best_mean:
            rbf_svm_best_mean = cv_mean_acc[i, j]
            rbf_svm_best_param = [g, c]
sio.savemat('rbf_svm_cv_mean_acc.mat', {'cv_mean_acc': cv_mean_acc})
print("CV results: gamma = ", rbf_svm_best_param[0], ", C = ",
rbf_svm_best_param[1])
rbf_svm_best = Pipeline((
            ("scaler", StandardScaler()),
            ("svm_clf", SVC(kernel="rbf", gamma=rbf_svm_best_param[0],
C=rbf_svm_best_param[1], probability=True))
        ))
time_start = time.clock()
rbf_svm_best.fit(Xtrain_red, ytrain)
time_elapsed = time.clock() - time_start
ytrain_pred = rbf_svm_best.predict(Xtrain_red)
train_accuracy = accuracy_score(ytrain, ytrain_pred)
print("Training accuracy = ", train_accuracy, ", training time = ",
time_elapsed)
ytest_pred = rbf_svm_best.predict(Xtest_red)
test_accuracy = accuracy_score(ytest, ytest_pred)
print("Test accuracy = ", test_accuracy)


# k-nearest neighbors classifier
from sklearn.neighbors import KNeighborsClassifier

print("\n***** K-NEAREST NEIGHBORS *****")
knn_best_mean = 0.
knn_best_param = ['uniform', 1]
cv_mean_acc = np.zeros((2, 10))
i = -1
for weigh in ['uniform', 'distance']:
    i = i + 1
    j = -1
    for neigh in range(1, 11):
        j = j + 1
        knn_clf = KNeighborsClassifier(weights=weigh, n_neighbors=neigh)
        time_start = time.clock()
        cv_accuracies = cross_val_score(knn_clf, Xtrain_red, ytrain, cv=5,
scoring="accuracy")
        time_elapsed = time.clock() - time_start
        cv_mean_acc[i, j] = np.mean(cv_accuracies)
        print("CV mean accuracy = ", cv_mean_acc[i, j], ", CV training time =
", time_elapsed)
        if cv_mean_acc[i, j] > knn_best_mean:
            knn_best_mean = cv_mean_acc[i, j]
            knn_best_param = [weigh, neigh]
sio.savemat('knn_cv_mean_acc.mat', {'cv_mean_acc': cv_mean_acc})
print("CV results: weights = ", knn_best_param[0], ", n_neighbors = ",
knn_best_param[1])
knn_best = KNeighborsClassifier(weights=knn_best_param[0],
n_neighbors=knn_best_param[1])
time_start = time.clock()
knn_best.fit(Xtrain_red, ytrain)
time_elapsed = time.clock() - time_start
ytrain_pred = knn_best.predict(Xtrain_red)
train_accuracy = accuracy_score(ytrain, ytrain_pred)
print("Training accuracy = ", train_accuracy, ", training time = ",
time_elapsed)
```

```
ytest_pred = knn_best.predict(Xtest_red)
test_accuracy = accuracy_score(ytest, ytest_pred)
print("Test accuracy = ", test_accuracy)


# Random forest classifier
from sklearn.ensemble import RandomForestClassifier

print("\n***** RANDOM FOREST *****")
rnd_best_mean = 0.
rnd_best_param = 0
cv_mean_acc = np.zeros(10)
i = -1
for nest in 5*np.logspace(2, 3, 10).astype(int):
    i = i + 1
    rnd_clf = RandomForestClassifier(n_estimators=nest)
    time_start = time.clock()
    cv_accuracies = cross_val_score(rnd_clf, Xtrain_red, ytrain, cv=5,
scoring="accuracy")
    time_elapsed = time.clock() - time_start
    cv_mean_acc[i] = np.mean(cv_accuracies)
    print("CV mean = ", cv_mean_acc[i], ", CV training time = ",
time_elapsed)
    if cv_mean_acc[i] > rnd_best_mean:
        rnd_best_mean = cv_mean_acc[i]
        rnd_best_param = nest
sio.savemat('rnd_cv_mean_acc.mat', {'cv_mean_acc': cv_mean_acc})
print("CV results: n_estimators = ", rnd_best_param)
rnd_best = RandomForestClassifier(n_estimators=rnd_best_param)
time_start = time.clock()
rnd_best.fit(Xtrain_red, ytrain)
time_elapsed = time.clock() - time_start
ytrain_pred = rnd_best.predict(Xtrain_red)
train_accuracy = accuracy_score(ytrain, ytrain_pred)
print("Training accuracy = ", train_accuracy, ", training time = ",
time_elapsed)
ytest_pred = rnd_best.predict(Xtest_red)
test_accuracy = accuracy_score(ytest, ytest_pred)
print("Test accuracy = ", test_accuracy)


# Voting classifier
from sklearn.ensemble import VotingClassifier

print("\n***** SOFT VOTING *****")
voting_clf = VotingClassifier(
    estimators=[('svc', rbf_svm_best), ('knn', knn_best), ('rf', rnd_best)],
    voting='soft'
)
time_start = time.clock()
voting_clf.fit(Xtrain_red, ytrain)
time_elapsed = time.clock() - time_start
ytrain_pred = voting_clf.predict(Xtrain_red)
train_accuracy = accuracy_score(ytrain, ytrain_pred)
print("Training accuracy = ", train_accuracy, ", training time = ",
time_elapsed)
ytest_pred = voting_clf.predict(Xtest_red)
```

```
test_accuracy = accuracy_score(ytest, ytest_pred)
print("Test accuracy = ", test_accuracy)
```

Output:

```
PCA results: min dimensions =  93 , dimensions used =  100

***** SUPPORT VECTOR MACHINE *****
CV mean =  0.142827888063 , CV training time =  32.943193435897435
CV mean =  0.145611063458 , CV training time =  34.628110769230766
CV mean =  0.148367829081 , CV training time =  32.193131487179485
CV mean =  0.150367835941 , CV training time =  31.00060471794872
CV mean =  0.153185410494 , CV training time =  31.395480205128223
CV mean =  0.154781446837 , CV training time =  31.817850666666686
CV mean =  0.154368905381 , CV training time =  32.362684717948724
CV mean =  0.156369964137 , CV training time =  31.228137846153828
CV mean =  0.157382241466 , CV training time =  33.00108676923077
CV mean =  0.15941568885 , CV training time =  32.794414358974336
CV mean =  0.37852161781 , CV training time =  42.20719466666668
CV mean =  0.38072270881 , CV training time =  41.84005702564099
CV mean =  0.381115119825 , CV training time =  42.13312738461542
CV mean =  0.381930770895 , CV training time =  42.86716882051286
CV mean =  0.381711531409 , CV training time =  42.73939815384608
CV mean =  0.381109220753 , CV training time =  42.53218174358972
CV mean =  0.381501988804 , CV training time =  42.348320410256406
CV mean =  0.381715159351 , CV training time =  42.66269046153843
CV mean =  0.381303310843 , CV training time =  43.4994892307692
CV mean =  0.380297012369 , CV training time =  42.59741497435891
CV mean =  0.100161909675 , CV training time =  47.07037169230773
CV mean =  0.099963889873 , CV training time =  45.8874912820512
CV mean =  0.100159202373 , CV training time =  43.859528205128186
CV mean =  0.100159202373 , CV training time =  45.346867282051335
CV mean =  0.100159202373 , CV training time =  43.08872861538464
CV mean =  0.100159202373 , CV training time =  44.72379446153843
CV mean =  0.100166100604 , CV training time =  47.49023712820508
CV mean =  0.100572834984 , CV training time =  44.26500964102547
CV mean =  0.100377522484 , CV training time =  43.32397538461532
CV mean =  0.100579338831 , CV training time =  43.62418461538459
CV results: gamma =  0.01 , C =  4.0
Training accuracy =  0.9216 , training time =  54.269248410256296
Test accuracy =  0.388932190179

***** K-NEAREST NEIGHBORS *****
CV mean accuracy =  0.420532637974 , CV training time =  2.9367351794871865
CV mean accuracy =  0.36871351557 , CV training time =  2.6837398974359985
CV mean accuracy =  0.367301306272 , CV training time =  2.878584615384625
CV mean accuracy =  0.368228546562 , CV training time =  2.8899926153844717
CV mean accuracy =  0.368198109993 , CV training time =  2.950480000000198
CV mean accuracy =  0.365668744374 , CV training time =  2.9938662564102287
CV mean accuracy =  0.366000235897 , CV training time =  3.0545226666665712
CV mean accuracy =  0.371818120893 , CV training time =  3.0829202051281754
CV mean accuracy =  0.365038804282 , CV training time =  3.122063589743675
CV mean accuracy =  0.361428226488 , CV training time =  3.2945645128204433
CV mean accuracy =  0.420532637974 , CV training time =  2.5747757948718117
CV mean accuracy =  0.420532637974 , CV training time =  2.9036812307690525
CV mean accuracy =  0.422952168775 , CV training time =  2.95586502564106
CV mean accuracy =  0.420741736017 , CV training time =  3.1305862564104245
CV mean accuracy =  0.411117453409 , CV training time =  3.236346666666577
CV mean accuracy =  0.404650143087 , CV training time =  3.088951384615484
CV mean accuracy =  0.40183901582 , CV training time =  3.123088410256514
CV mean accuracy =  0.400472072236 , CV training time =  3.119988512820555
CV mean accuracy =  0.396671300728 , CV training time =  3.1911881025641833
CV mean accuracy =  0.393512515998 , CV training time =  3.314318358974333
CV results: weights =  distance , n_neighbors =  3
Training accuracy =  1.0 , training time =  0.030813128205181783
Test accuracy =  0.455183164458

***** RANDOM FOREST *****
CV mean =  0.414887453515 , CV training time =  265.1951745641027
CV mean =  0.420340239991 , CV training time =  343.4282100512819
CV mean =  0.417151063501 , CV training time =  452.2485919999999
CV mean =  0.419322473363 , CV training time =  578.1513575384615
CV mean =  0.424929633063 , CV training time =  746.0376549743592
CV mean =  0.42378414927 , CV training time =  1187.608664615385
CV mean =  0.419192238407 , CV training time =  2413.842319589744
CV mean =  0.425948551417 , CV training time =  3090.0032455384617
CV mean =  0.427139296483 , CV training time =  3829.5460143589753
CV mean =  0.423365818687 , CV training time =  2911.274585435898
CV results: n_estimators =  3870
Training accuracy =  1.0 , training time =  528.74932923077
Test accuracy =  0.445830085737

***** SOFT VOTING *****
Training accuracy =  1.0 , training time =  599.1445620512823
Test accuracy =  0.479345284489

Process finished with exit code 0
```

MATLAB code:

```
close all; clearvars

% Support vector machine
load('rbf_svm_cv_mean_acc.mat')
figure(1)
gamma = linspace(-1,-3,3);
C = linspace(3.5,5,10);
imagesc(C,gamma,cv_mean_acc)
c = colorbar;
c.Label.String = 'Mean accuracy';
title('Cross Validation: Support Vector Machine')
xlabel('C')
ylabel('log_{10}\gamma')

figure(2)
plot(C,cv_mean_acc(2,:))
title('Cross Validation: Support Vector Machine')
xlabel('C')
ylabel('Mean accuracy')

% k-nearest neighbors
load('knn_cv_mean_acc.mat')
k = 1:10;
figure(3)
```

```
hold on
plot(k,cv_mean_acc(1,:))
plot(k,cv_mean_acc(2,:))
hold off
title('Cross Validation: k-Nearest Neighbors')
xlabel('k')
ylabel('Mean accuracy')
legend('weights = ''uniform''','weights = ''distance''')

% Random forest
load('rnd_cv_mean_acc.mat')
n_trees = 5*logspace(2,3,10);
figure(4)
semilogx(n_trees,cv_mean_acc)
title('Cross Validation: Random Forest')
xlabel('n_{trees}')
xlim([5e2 5e3])
ylabel('Mean accuracy')
```