

```
In [6]: import pandas as pd
import numpy as np
from sklearn.preprocessing import StandardScaler
from sklearn.manifold import TSNE
from sklearn.neighbors import KNeighborsClassifier
from sklearn.model_selection import cross_validate, StratifiedKFold, train_
from sklearn.svm import SVC
from sklearn.naive_bayes import GaussianNB
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import confusion_matrix
import seaborn as sns
%matplotlib inline
import matplotlib.pyplot as plt
# from tpot import TPOTClassifier
# from xgboost import XGBClassifier
```

Load Data

```
In [7]: nt_all = pd.read_csv('data/nt.all.csv')
```

```
In [8]: nt_coding = pd.read_csv('data/nt.coding.csv')
```

```
In [9]: nt_all.head()
```

Out[9]:

	Type	ENSG00000000003.13	ENSG00000000005.5	ENSG00000000419.11	ENSG00000000457.12	ENS
0	0	150265.480539	4327.845865	713909.310619	59794.653619	
1	0	913228.181789	2326.284691	828500.414250	50302.756694	
2	0	359658.934678	228971.470681	483960.593070	69872.468893	
3	1	135634.675596	0.000000	748257.784782	75504.611322	
4	0	81454.831124	177.310309	363281.940134	45622.048124	

5 rows × 60484 columns

```
In [10]: nt_all.shape
```

Out[10]: (1400, 60484)

```
In [11]: nt_coding.head()
```

```
Out[11]:
```

	Type	ENSG00000000003.13	ENSG00000000005.5	ENSG000000000419.11	ENSG000000000457.12	ENSG000000000457.12
0	0	150265.480539	4327.845865	713909.310619	59794.653619	
1	0	913228.181789	2326.284691	828500.414250	50302.756694	
2	0	359658.934678	228971.470681	483960.593070	69872.468893	
3	1	135634.675596	0.000000	748257.784782	75504.611322	
4	0	81454.831124	177.310309	363281.940134	45622.048124	

5 rows × 19562 columns

```
In [12]: nt_coding.shape
```

```
Out[12]: (1400, 19562)
```

Pre-processing

Seperate labels and features

```
In [13]: y_coding = nt_coding['Type']
X_coding = nt_coding.drop('Type', axis=1)
```

```
In [14]: y_all = nt_all['Type']
X_all = nt_all.drop('Type', axis=1)
```

Split into training and validation sets

```
In [15]: X_coding_train, X_coding_test, y_coding_train, y_coding_test = train_test_split(X_coding, y_coding,
                                                                                          test_size=.25, random_s
```

```
In [16]: X_all_train, X_all_test, y_all_train, y_all_test = train_test_split(X_all, y_all,
                                                                              test_size=.25, random_s
```

```
In [34]: scaler_coding = StandardScaler()
scaler_all = StandardScaler()

X_coding_train = scaler_coding.fit_transform(X_coding_train)
X_coding_test = scaler_coding.transform(X_coding_test)

X_all_train = scaler_all.fit_transform(X_all_train)
X_all_test = scaler_all.transform(X_all_test)
```

Classification

- k-nearest neighbors
- Support Vector Machine (SVM)
- Naive Bayes
- Decision Tree
- Random Forest

```
In [35]: def display_cv_metrics(metrics):  
         for k, v in metrics.items():  
             print(f'{k}: {np.round(v.mean(), 3)}')
```

```
In [36]: metrics = ['accuracy', 'f1', 'precision', 'recall', 'roc_auc']
```

```
In [38]: skf = StratifiedKFold(n_splits=5, random_state=1, shuffle=True)
```

k-nearest neighbors

```
In [40]: knn = KNeighborsClassifier()
```

```
In [41]: knn_scores_coding = cross_validate(estimator=knn,  
                                           X=X_coding_train,  
                                           y=y_coding_train,  
                                           scoring=metrics,  
                                           cv=skf)
```

```
In [42]: knn_scores_all = cross_validate(estimator=knn,  
                                         X=X_all_train,  
                                         y=y_all_train,  
                                         scoring=metrics,  
                                         cv=skf)
```

```
In [43]: display_cv_metrics(knn_scores_coding)
```

```
fit_time: 0.81  
score_time: 11.025  
test_accuracy: 0.891  
test_f1: 0.882  
test_precision: 0.982  
test_recall: 0.802  
test_roc_auc: 0.966
```

```
In [44]: display_cv_metrics(knn_scores_all)
```

```
fit_time: 3.364
score_time: 32.006
test_accuracy: 0.841
test_f1: 0.817
test_precision: 0.984
test_recall: 0.7
test_roc_auc: 0.956
```

Support Vector Machine

```
In [97]: svc = SVC(kernel='linear')
```

```
In [98]: svc_scores_coding = cross_validate(estimator=svc,
                                             X=X_coding_train,
                                             y=y_coding_train,
                                             scoring=metrics,
                                             cv=skf)
```

```
In [99]: svc_scores_all = cross_validate(estimator=svc,
                                          X=X_all_train,
                                          y=y_all_train,
                                          scoring=metrics,
                                          cv=skf)
```

```
In [100]: display_cv_metrics(svc_scores_coding)
```

```
fit_time: 4.691
score_time: 2.066
test_accuracy: 0.972
test_f1: 0.973
test_precision: 0.98
test_recall: 0.966
test_roc_auc: 0.992
```

```
In [101]: display_cv_metrics(svc_scores_all)
```

```
fit_time: 20.542
score_time: 9.303
test_accuracy: 0.972
test_f1: 0.973
test_precision: 0.981
test_recall: 0.965
test_roc_auc: 0.993
```

Naive Bayes Classifier

```
In [50]: nb = GaussianNB()
```

```
In [51]: nb_scores_coding = cross_validate(estimator=nb,  
                                           X=X_coding_train,  
                                           y=y_coding_train,  
                                           scoring=metrics,  
                                           cv=skf)
```

```
In [73]: nb_scores_all = cross_validate(estimator=nb,  
                                         X=X_all_train,  
                                         y=y_all_train,  
                                         scoring=metrics,  
                                         cv=skf)
```

```
In [53]: display_cv_metrics(nb_scores_coding)
```

```
fit_time: 0.444  
score_time: 0.095  
test_accuracy: 0.918  
test_f1: 0.921  
test_precision: 0.91  
test_recall: 0.933  
test_roc_auc: 0.922
```

```
In [74]: display_cv_metrics(nb_scores_all)
```

```
fit_time: 2.445  
score_time: 0.309  
test_accuracy: 0.657  
test_f1: 0.708  
test_precision: 0.629  
test_recall: 0.812  
test_roc_auc: 0.654
```

Decision Tree Classifier

```
In [55]: dt = DecisionTreeClassifier(random_state=1)
```

```
In [56]: dt_scores_coding = cross_validate(estimator=dt,  
                                           X=X_coding_train,  
                                           y=y_coding_train,  
                                           scoring=metrics,  
                                           cv=skf)
```

```
In [57]: dt_scores_all = cross_validate(estimator=dt,  
                                         X=X_all_train,  
                                         y=y_all_train,  
                                         scoring=metrics,  
                                         cv=skf)
```

```
In [58]: display_cv_metrics(dt_scores_coding)
```

```
fit_time: 6.907
score_time: 0.02
test_accuracy: 0.909
test_f1: 0.91
test_precision: 0.914
test_recall: 0.907
test_roc_auc: 0.909
```

```
In [59]: display_cv_metrics(dt_scores_all)
```

```
fit_time: 14.78
score_time: 0.06
test_accuracy: 0.905
test_f1: 0.906
test_precision: 0.913
test_recall: 0.899
test_roc_auc: 0.905
```

Random Forest Classifier

```
In [60]: rf = RandomForestClassifier(max_depth=3, random_state=1)
```

```
In [61]: rf_scores_coding = cross_validate(estimator=rf,
                                           X=X_coding_train,
                                           y=y_coding_train,
                                           scoring=metrics,
                                           cv=skf)
```

```
In [62]: rf_scores_all = cross_validate(estimator=rf,
                                         X=X_all_train,
                                         y=y_all_train,
                                         scoring=metrics,
                                         cv=skf)
```

```
In [63]: display_cv_metrics(rf_scores_coding)
```

```
fit_time: 1.971
score_time: 0.037
test_accuracy: 0.957
test_f1: 0.958
test_precision: 0.956
test_recall: 0.961
test_roc_auc: 0.991
```

```
In [64]: display_cv_metrics(rf_scores_all)
```

```
fit_time: 3.422
score_time: 0.067
test_accuracy: 0.956
test_f1: 0.957
test_precision: 0.961
test_recall: 0.953
test_roc_auc: 0.992
```

Selecting the best performer

```
In [65]: scores_all = [knn_scores_all, svc_scores_all, nb_scores_all,
                       dt_scores_all, rf_scores_all]

scores_coding = [knn_scores_coding, svc_scores_coding,
                 nb_scores_coding, dt_scores_coding, rf_scores_coding]

names = ['K-Nearest Neighbors', 'Support Vector Classifier', 'Naive Bayes C
         'Decision Tree Classifier', 'Random Forest Classifier']
```

```
In [66]: for score in scores_all:
          for k, v in score.items():
              score[k] = v.mean()
```

```
In [67]: for score in scores_coding:
          for k, v in score.items():
              score[k] = v.mean()
```

For all genes

```
In [109]: pd.DataFrame(scores_all, index=names)
```

```
Out[109]:
```

	fit_time	score_time	test_accuracy	test_f1	test_precision	test_recall	test_roc_auc
K-Nearest Neighbors	3.364365	32.006083	0.840952	0.816745	0.984169	0.699533	0.955647
Support Vector Classifier	20.541505	9.303161	0.972381	0.972630	0.981336	0.964521	0.992669
Naive Bayes Classifier	2.191593	0.295318	0.657143	0.707806	0.629284	0.811578	0.653981
Decision Tree Classifier	14.779774	0.060417	0.904762	0.905943	0.912786	0.899290	0.904899
Random Forest Classifier	3.422036	0.067356	0.956190	0.956934	0.960781	0.953340	0.992069

For only protein-coding genes

```
In [110]: pd.DataFrame(scores_coding, index=names)
```

```
Out[110]:
```

	fit_time	score_time	test_accuracy	test_f1	test_precision	test_recall	test_roc_auc
K-Nearest Neighbors	0.809870	11.025419	0.891429	0.882157	0.981738	0.802077	0.965718
Support Vector Classifier	4.690663	2.065730	0.972381	0.972663	0.979575	0.966390	0.992378
Naive Bayes Classifier	0.443795	0.095060	0.918095	0.920576	0.909718	0.932866	0.921958
Decision Tree Classifier	6.906948	0.019686	0.908571	0.910091	0.913765	0.906819	0.908616
Random Forest Classifier	1.971109	0.036887	0.957143	0.958180	0.956230	0.960782	0.991180

Important RNA-sequences in Classifying Tumor

Support Vector Classifier with linear kernel seems to have a slightly higher cross validated ROC AUC and accuracy scores across both datasets, closely followed by the Random Forest Classifier.

Dataset with just the protein-coding genes seems to be performing as well as the full dataset despite having fewer features. We will compute Random Forest feature importances based on

mean decrease in impurity and display the first 10 genes that are the best predictors of tumor.

Random Forest Feature Importances

```
In [76]: rf.fit(X_all_train, y_all_train)
```

```
Out[76]: RandomForestClassifier(max_depth=3, random_state=1)
```

```
In [77]: importances = rf.feature_importances_
```

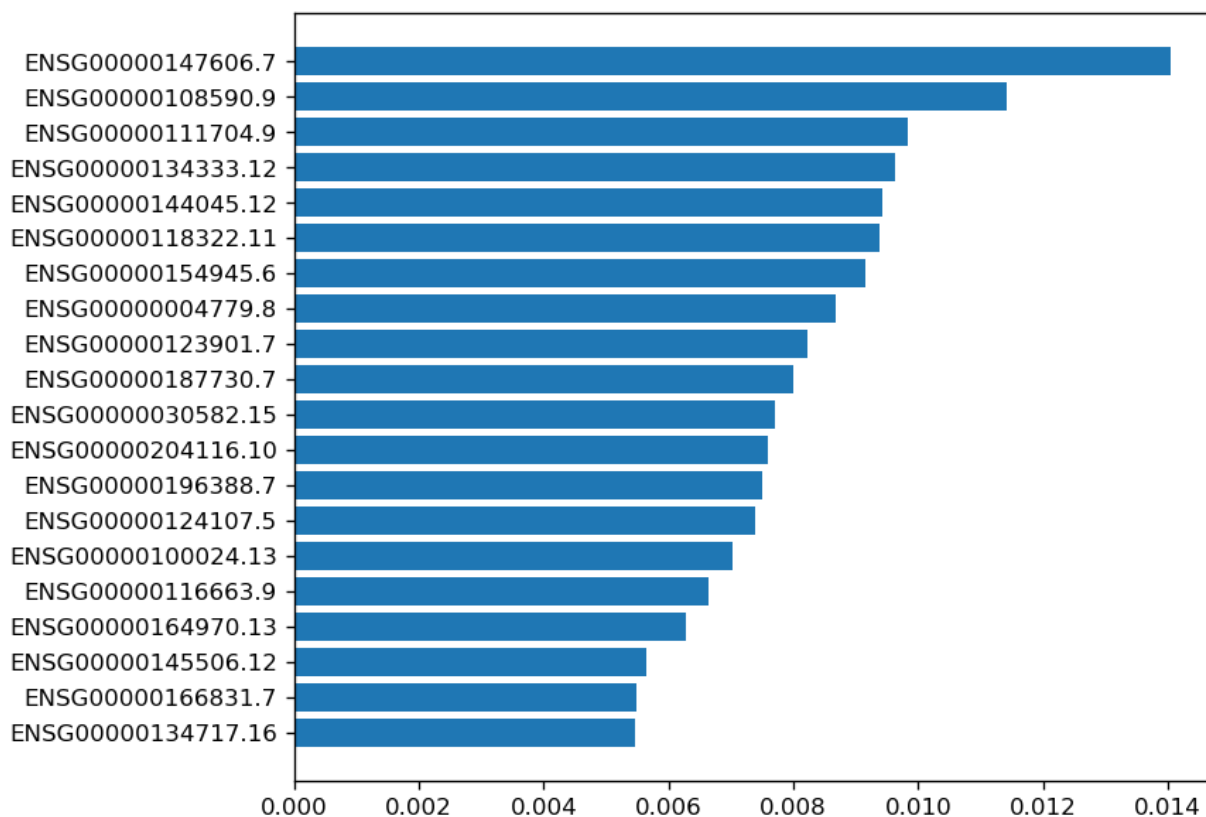
```
In [78]: feature_names = list(nt_coding.columns[1:])
```

```
In [79]: importance_df = pd.DataFrame(zip(feature_names, importances), columns=['gene', 'importance'])
```

Top 20 genes for predicting tumor

```
In [191]: top_20_rf = importance_df.sort_values('importance', ascending=False).head(20)
```

```
In [194]: plt.figure(figsize=(7, 6), dpi=120)
plt.barh(y=top_20_rf.gene[0:-1], width=top_20_rf.importance[0:-1]);
```



SVC Feature Importances

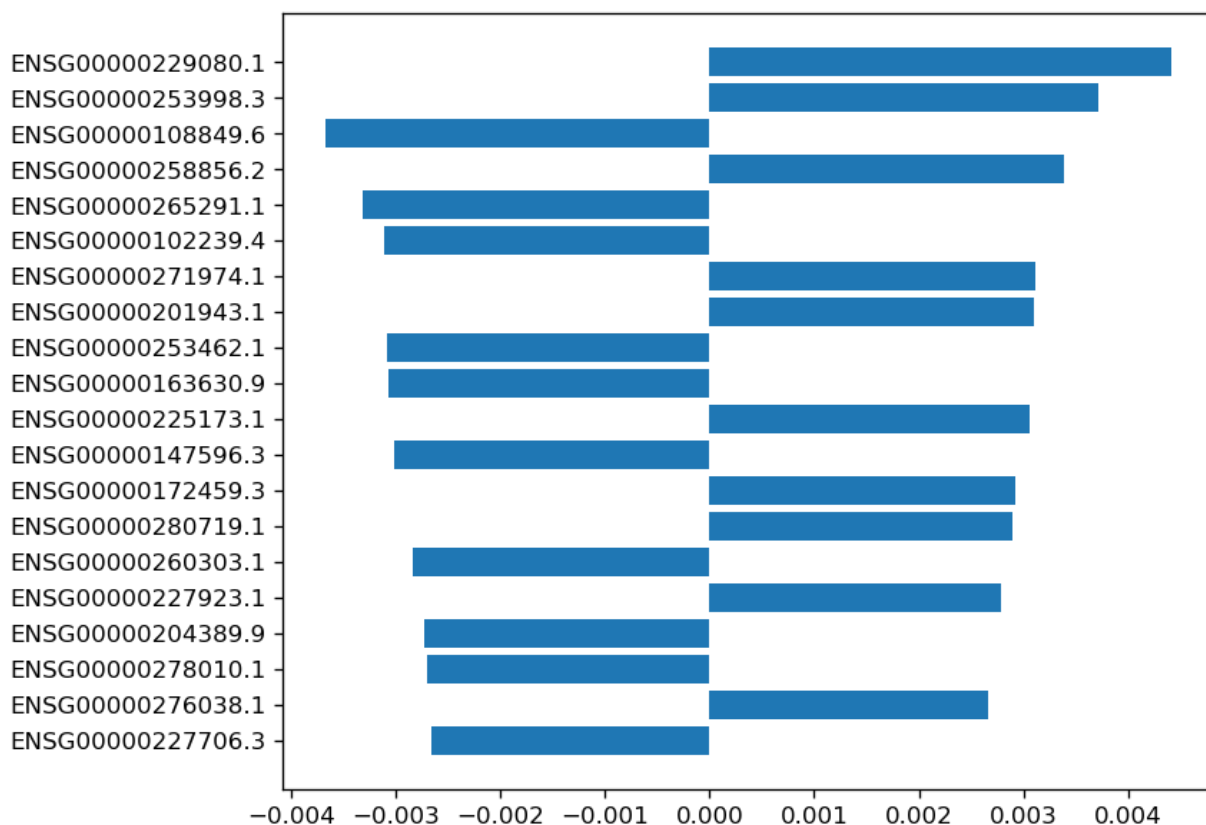
```
In [111]: svc = SVC(kernel='linear')
          svc.fit(X_all_train, y_all_train)
```

```
Out[111]: SVC(kernel='linear')
```

```
In [175]: names, imp = zip(*sorted(zip(nt_all.columns[1:], svc.coef_[0]), key=lambda
```

Highest coefficients where higher absolute value means more influence on the decision of the classifier:

```
In [198]: plt.figure(figsize=(7, 6), dpi=120)
          plt.barh(y=names[-20:], width=imp[-20:]);
```



Check to see if any common features for SVC and Random Forest in the top 20:

```
In [206]: set(names[-20:]) & set(top_20_rf.gene)
```

```
Out[206]: set()
```

Nope.

Confusion matrix with the predictions on the holdout set

Random Forest

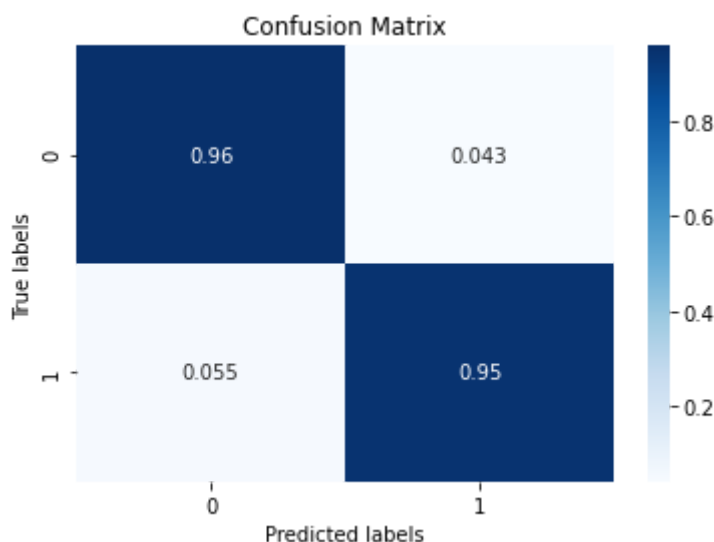
```
In [210]: y_all_pred = rf.predict(X_all_test)
```

```
In [211]: cf_matrix = confusion_matrix(y_all_test, y_all_pred, normalize='true')
```

```
In [213]: test_accuracy = (y_all_pred == y_all_test).sum() / len(y_all_test)
print(f"Test accuracy is {test_accuracy * 100}%")
```

Test accuracy is 95.14285714285714%

```
In [214]: ax = plt.axes()
sns.heatmap(cf_matrix, annot=True, cmap='Blues', ax=ax)
ax.set_title('Confusion Matrix')
ax.set_xlabel('Predicted labels')
ax.set_ylabel('True labels');
```



Support Vector Machine

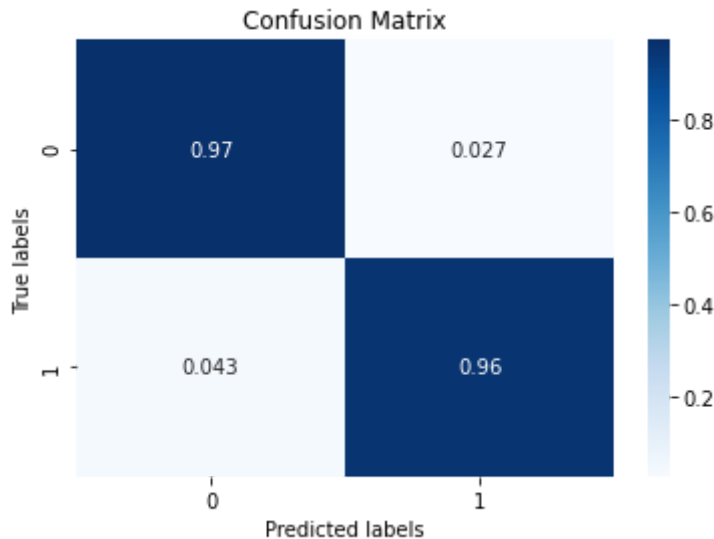
```
In [215]: y_all_pred = svc.predict(X_all_test)
```

```
In [216]: cf_matrix = confusion_matrix(y_all_test, y_all_pred, normalize='true')
```

```
In [217]: test_accuracy = (y_all_pred == y_all_test).sum() / len(y_all_test)
print(f"Test accuracy is {test_accuracy * 100}%")
```

Test accuracy is 96.57142857142857%

```
In [218]: ax = plt.axes()
sns.heatmap(cf_matrix, annot=True, cmap='Blues', ax=ax)
ax.set_title('Confusion Matrix')
ax.set_xlabel('Predicted labels')
ax.set_ylabel('True labels');
```



Deep Learning Classifier

```
In [1]: import keras
from keras.layers import Input, Dense
from keras.models import Model
```

```
In [21]: input_layer = Input(shape=(19561,))
dense = Dense(8192, activation='relu')(input_layer)
dense = Dense(2048, activation='relu')(input_layer)
dense = Dense(512, activation='relu')(input_layer)
dense = Dense(128, activation='relu')(input_layer)
dense = Dense(64, activation='relu')(dense)
dense = Dense(32, activation='relu')(dense)
dense = Dense(8, activation='relu')(dense)
output = Dense(1, activation='sigmoid')(dense)
```

```
In [22]: model = Model(input_layer, output)
```

```
In [23]: model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accur
```

```

In [30]: test_sizes = np.arange(0.05, 0.8, 0.05)
learning_dict = {'Test sizes': test_sizes,
                 'Accuracy': [],
                 'Loss': [],
                 'AUC': []}

random_weights = model.get_weights()
for size in test_sizes:
    model.set_weights(random_weights)
    X_coding_train, X_coding_test, y_coding_train, y_coding_test = train_test_split(X_coding, y_coding,
                                                                                      test_size=.25, random_state=42)

    scaler = StandardScaler()
    X_coding_train = scaler.fit_transform(X_coding_train)
    X_coding_test = scaler.transform(X_coding_test)

    model.fit(X_coding_train, y_coding_train, epochs=100,
              batch_size=20)
    loss, accuracy, auc = model.evaluate(X_coding_test, y_coding_test, verbose=0)
    learning_dict['Loss'].append(loss)
    learning_dict['Accuracy'].append(accuracy)
    learning_dict['AUC'].append(auc)

```

```

Epoch 1/100
53/53 [=====] - 0s 6ms/step - loss: 0.5584 - accuracy: 0.6610 - auc: 0.7505
Epoch 2/100
53/53 [=====] - 0s 5ms/step - loss: 0.5395 - accuracy: 0.6838 - auc: 0.7626
Epoch 3/100
53/53 [=====] - 0s 5ms/step - loss: 0.5190 - accuracy: 0.6848 - auc: 0.7749
Epoch 4/100
53/53 [=====] - 0s 5ms/step - loss: 0.5076 - accuracy: 0.6981 - auc: 0.7988
Epoch 5/100
53/53 [=====] - 0s 5ms/step - loss: 0.4973 - accuracy: 0.7038 - auc: 0.8091
Epoch 6/100
53/53 [=====] - 0s 5ms/step - loss: 0.4868 - accuracy: 0.7076 - auc: 0.8193
Epoch 7/100
53/53 [=====] - 0s 5ms/step - loss: 0.4763 - accuracy: 0.7114 - auc: 0.8293
Epoch 8/100
53/53 [=====] - 0s 5ms/step - loss: 0.4658 - accuracy: 0.7152 - auc: 0.8393
Epoch 9/100
53/53 [=====] - 0s 5ms/step - loss: 0.4553 - accuracy: 0.7190 - auc: 0.8493
Epoch 10/100
53/53 [=====] - 0s 5ms/step - loss: 0.4448 - accuracy: 0.7228 - auc: 0.8593
Epoch 11/100
53/53 [=====] - 0s 5ms/step - loss: 0.4343 - accuracy: 0.7266 - auc: 0.8693
Epoch 12/100
53/53 [=====] - 0s 5ms/step - loss: 0.4238 - accuracy: 0.7304 - auc: 0.8793
Epoch 13/100
53/53 [=====] - 0s 5ms/step - loss: 0.4133 - accuracy: 0.7342 - auc: 0.8893
Epoch 14/100
53/53 [=====] - 0s 5ms/step - loss: 0.4028 - accuracy: 0.7380 - auc: 0.8993
Epoch 15/100
53/53 [=====] - 0s 5ms/step - loss: 0.3923 - accuracy: 0.7418 - auc: 0.9093
Epoch 16/100
53/53 [=====] - 0s 5ms/step - loss: 0.3818 - accuracy: 0.7456 - auc: 0.9193
Epoch 17/100
53/53 [=====] - 0s 5ms/step - loss: 0.3713 - accuracy: 0.7494 - auc: 0.9293
Epoch 18/100
53/53 [=====] - 0s 5ms/step - loss: 0.3608 - accuracy: 0.7532 - auc: 0.9393
Epoch 19/100
53/53 [=====] - 0s 5ms/step - loss: 0.3503 - accuracy: 0.7570 - auc: 0.9493
Epoch 20/100
53/53 [=====] - 0s 5ms/step - loss: 0.3398 - accuracy: 0.7608 - auc: 0.9593
Epoch 21/100
53/53 [=====] - 0s 5ms/step - loss: 0.3293 - accuracy: 0.7646 - auc: 0.9693
Epoch 22/100
53/53 [=====] - 0s 5ms/step - loss: 0.3188 - accuracy: 0.7684 - auc: 0.9793
Epoch 23/100
53/53 [=====] - 0s 5ms/step - loss: 0.3083 - accuracy: 0.7722 - auc: 0.9893
Epoch 24/100
53/53 [=====] - 0s 5ms/step - loss: 0.2978 - accuracy: 0.7760 - auc: 0.9993
Epoch 25/100
53/53 [=====] - 0s 5ms/step - loss: 0.2873 - accuracy: 0.7798 - auc: 1.0093
Epoch 26/100
53/53 [=====] - 0s 5ms/step - loss: 0.2768 - accuracy: 0.7836 - auc: 1.0193
Epoch 27/100
53/53 [=====] - 0s 5ms/step - loss: 0.2663 - accuracy: 0.7874 - auc: 1.0293
Epoch 28/100
53/53 [=====] - 0s 5ms/step - loss: 0.2558 - accuracy: 0.7912 - auc: 1.0393
Epoch 29/100
53/53 [=====] - 0s 5ms/step - loss: 0.2453 - accuracy: 0.7950 - auc: 1.0493
Epoch 30/100
53/53 [=====] - 0s 5ms/step - loss: 0.2348 - accuracy: 0.7988 - auc: 1.0593
Epoch 31/100
53/53 [=====] - 0s 5ms/step - loss: 0.2243 - accuracy: 0.8026 - auc: 1.0693
Epoch 32/100
53/53 [=====] - 0s 5ms/step - loss: 0.2138 - accuracy: 0.8064 - auc: 1.0793
Epoch 33/100
53/53 [=====] - 0s 5ms/step - loss: 0.2033 - accuracy: 0.8102 - auc: 1.0893
Epoch 34/100
53/53 [=====] - 0s 5ms/step - loss: 0.1928 - accuracy: 0.8140 - auc: 1.0993
Epoch 35/100
53/53 [=====] - 0s 5ms/step - loss: 0.1823 - accuracy: 0.8178 - auc: 1.1093
Epoch 36/100
53/53 [=====] - 0s 5ms/step - loss: 0.1718 - accuracy: 0.8216 - auc: 1.1193
Epoch 37/100
53/53 [=====] - 0s 5ms/step - loss: 0.1613 - accuracy: 0.8254 - auc: 1.1293
Epoch 38/100
53/53 [=====] - 0s 5ms/step - loss: 0.1508 - accuracy: 0.8292 - auc: 1.1393
Epoch 39/100
53/53 [=====] - 0s 5ms/step - loss: 0.1403 - accuracy: 0.8330 - auc: 1.1493
Epoch 40/100
53/53 [=====] - 0s 5ms/step - loss: 0.1298 - accuracy: 0.8368 - auc: 1.1593
Epoch 41/100
53/53 [=====] - 0s 5ms/step - loss: 0.1193 - accuracy: 0.8406 - auc: 1.1693
Epoch 42/100
53/53 [=====] - 0s 5ms/step - loss: 0.1088 - accuracy: 0.8444 - auc: 1.1793
Epoch 43/100
53/53 [=====] - 0s 5ms/step - loss: 0.0983 - accuracy: 0.8482 - auc: 1.1893
Epoch 44/100
53/53 [=====] - 0s 5ms/step - loss: 0.0878 - accuracy: 0.8520 - auc: 1.1993
Epoch 45/100
53/53 [=====] - 0s 5ms/step - loss: 0.0773 - accuracy: 0.8558 - auc: 1.2093
Epoch 46/100
53/53 [=====] - 0s 5ms/step - loss: 0.0668 - accuracy: 0.8596 - auc: 1.2193
Epoch 47/100
53/53 [=====] - 0s 5ms/step - loss: 0.0563 - accuracy: 0.8634 - auc: 1.2293
Epoch 48/100
53/53 [=====] - 0s 5ms/step - loss: 0.0458 - accuracy: 0.8672 - auc: 1.2393
Epoch 49/100
53/53 [=====] - 0s 5ms/step - loss: 0.0353 - accuracy: 0.8710 - auc: 1.2493
Epoch 50/100
53/53 [=====] - 0s 5ms/step - loss: 0.0248 - accuracy: 0.8748 - auc: 1.2593
Epoch 51/100
53/53 [=====] - 0s 5ms/step - loss: 0.0143 - accuracy: 0.8786 - auc: 1.2693
Epoch 52/100
53/53 [=====] - 0s 5ms/step - loss: 0.0038 - accuracy: 0.8824 - auc: 1.2793
Epoch 53/100
53/53 [=====] - 0s 5ms/step - loss: 0.0033 - accuracy: 0.8862 - auc: 1.2893
Epoch 54/100
53/53 [=====] - 0s 5ms/step - loss: 0.0028 - accuracy: 0.8900 - auc: 1.2993
Epoch 55/100
53/53 [=====] - 0s 5ms/step - loss: 0.0023 - accuracy: 0.8938 - auc: 1.3093
Epoch 56/100
53/53 [=====] - 0s 5ms/step - loss: 0.0018 - accuracy: 0.8976 - auc: 1.3193
Epoch 57/100
53/53 [=====] - 0s 5ms/step - loss: 0.0013 - accuracy: 0.9014 - auc: 1.3293
Epoch 58/100
53/53 [=====] - 0s 5ms/step - loss: 0.0008 - accuracy: 0.9052 - auc: 1.3393
Epoch 59/100
53/53 [=====] - 0s 5ms/step - loss: 0.0003 - accuracy: 0.9090 - auc: 1.3493
Epoch 60/100
53/53 [=====] - 0s 5ms/step - loss: 0.0003 - accuracy: 0.9128 - auc: 1.3593
Epoch 61/100
53/53 [=====] - 0s 5ms/step - loss: 0.0003 - accuracy: 0.9166 - auc: 1.3693
Epoch 62/100
53/53 [=====] - 0s 5ms/step - loss: 0.0003 - accuracy: 0.9204 - auc: 1.3793
Epoch 63/100
53/53 [=====] - 0s 5ms/step - loss: 0.0003 - accuracy: 0.9242 - auc: 1.3893
Epoch 64/100
53/53 [=====] - 0s 5ms/step - loss: 0.0003 - accuracy: 0.9280 - auc: 1.3993
Epoch 65/100
53/53 [=====] - 0s 5ms/step - loss: 0.0003 - accuracy: 0.9318 - auc: 1.4093
Epoch 66/100
53/53 [=====] - 0s 5ms/step - loss: 0.0003 - accuracy: 0.9356 - auc: 1.4193
Epoch 67/100
53/53 [=====] - 0s 5ms/step - loss: 0.0003 - accuracy: 0.9394 - auc: 1.4293
Epoch 68/100
53/53 [=====] - 0s 5ms/step - loss: 0.0003 - accuracy: 0.9432 - auc: 1.4393
Epoch 69/100
53/53 [=====] - 0s 5ms/step - loss: 0.0003 - accuracy: 0.9470 - auc: 1.4493
Epoch 70/100
53/53 [=====] - 0s 5ms/step - loss: 0.0003 - accuracy: 0.9508 - auc: 1.4593
Epoch 71/100
53/53 [=====] - 0s 5ms/step - loss: 0.0003 - accuracy: 0.9546 - auc: 1.4693
Epoch 72/100
53/53 [=====] - 0s 5ms/step - loss: 0.0003 - accuracy: 0.9584 - auc: 1.4793
Epoch 73/100
53/53 [=====] - 0s 5ms/step - loss: 0.0003 - accuracy: 0.9622 - auc: 1.4893
Epoch 74/100
53/53 [=====] - 0s 5ms/step - loss: 0.0003 - accuracy: 0.9660 - auc: 1.4993
Epoch 75/100
53/53 [=====] - 0s 5ms/step - loss: 0.0003 - accuracy: 0.9698 - auc: 1.5093
Epoch 76/100
53/53 [=====] - 0s 5ms/step - loss: 0.0003 - accuracy: 0.9736 - auc: 1.5193
Epoch 77/100
53/53 [=====] - 0s 5ms/step - loss: 0.0003 - accuracy: 0.9774 - auc: 1.5293
Epoch 78/100
53/53 [=====] - 0s 5ms/step - loss: 0.0003 - accuracy: 0.9812 - auc: 1.5393
Epoch 79/100
53/53 [=====] - 0s 5ms/step - loss: 0.0003 - accuracy: 0.9850 - auc: 1.5493
Epoch 80/100
53/53 [=====] - 0s 5ms/step - loss: 0.0003 - accuracy: 0.9888 - auc: 1.5593
Epoch 81/100
53/53 [=====] - 0s 5ms/step - loss: 0.0003 - accuracy: 0.9926 - auc: 1.5693
Epoch 82/100
53/53 [=====] - 0s 5ms/step - loss: 0.0003 - accuracy: 0.9964 - auc: 1.5793
Epoch 83/100
53/53 [=====] - 0s 5ms/step - loss: 0.0003 - accuracy: 0.9998 - auc: 1.5893
Epoch 84/100
53/53 [=====] - 0s 5ms/step - loss: 0.0003 - accuracy: 0.9998 - auc: 1.5993
Epoch 85/100
53/53 [=====] - 0s 5ms/step - loss: 0.0003 - accuracy: 0.9998 - auc: 1.6093
Epoch 86/100
53/53 [=====] - 0s 5ms/step - loss: 0.0003 - accuracy: 0.9998 - auc: 1.6193
Epoch 87/100
53/53 [=====] - 0s 5ms/step - loss: 0.0003 - accuracy: 0.9998 - auc: 1.6293
Epoch 88/100
53/53 [=====] - 0s 5ms/step - loss: 0.0003 - accuracy: 0.9998 - auc: 1.6393
Epoch 89/100
53/53 [=====] - 0s 5ms/step - loss: 0.0003 - accuracy: 0.9998 - auc: 1.6493
Epoch 90/100
53/53 [=====] - 0s 5ms/step - loss: 0.0003 - accuracy: 0.9998 - auc: 1.6593
Epoch 91/100
53/53 [=====] - 0s 5ms/step - loss: 0.0003 - accuracy: 0.9998 - auc: 1.6693
Epoch 92/100
53/53 [=====] - 0s 5ms/step - loss: 0.0003 - accuracy: 0.9998 - auc: 1.6793
Epoch 93/100
53/53 [=====] - 0s 5ms/step - loss: 0.0003 - accuracy: 0.9998 - auc: 1.6893
Epoch 94/100
53/53 [=====] - 0s 5ms/step - loss: 0.0003 - accuracy: 0.9998 - auc: 1.6993
Epoch 95/100
53/53 [=====] - 0s 5ms/step - loss: 0.0003 - accuracy: 0.9998 - auc: 1.7093
Epoch 96/100
53/53 [=====] - 0s 5ms/step - loss: 0.0003 - accuracy: 0.9998 - auc: 1.7193
Epoch 97/100
53/53 [=====] - 0s 5ms/step - loss: 0.0003 - accuracy: 0.9998 - auc: 1.7293
Epoch 98/100
53/53 [=====] - 0s 5ms/step - loss: 0.0003 - accuracy: 0.9998 - auc: 1.7393
Epoch 99/100
53/53 [=====] - 0s 5ms/step - loss: 0.0003 - accuracy: 0.9998 - auc: 1.7493
Epoch 100/100
53/53 [=====] - 0s 5ms/step - loss: 0.0003 - accuracy: 0.9998 - auc: 1.7593

```

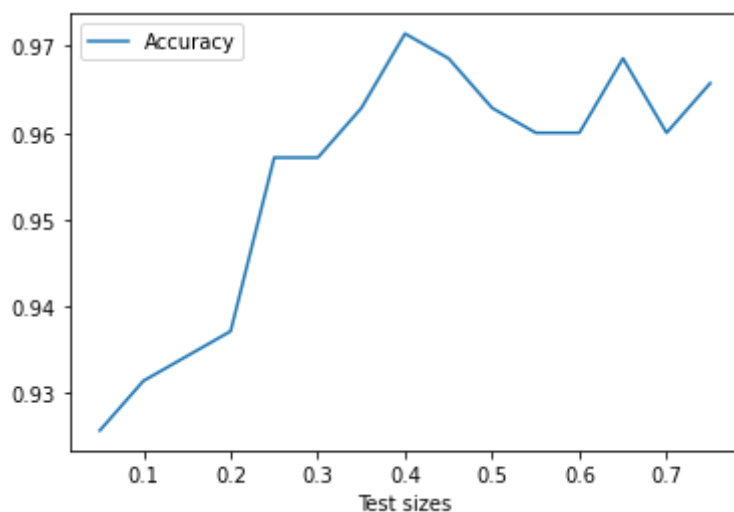
Learning Curve

```
In [37]: learning_df
```

```
Out[37]:
```

	Test sizes	Accuracy	Loss	AUC
0	0.05	0.925714	0.510010	0.956825
1	0.10	0.931429	0.710257	0.953400
2	0.15	0.934286	0.364858	0.970873
3	0.20	0.937143	0.490639	0.967381
4	0.25	0.957143	0.474641	0.966726
5	0.30	0.957143	0.609394	0.967889
6	0.35	0.962857	0.968809	0.965087
7	0.40	0.971429	0.969524	0.976118
8	0.45	0.968571	0.566424	0.975741
9	0.50	0.962857	0.687308	0.967545
10	0.55	0.960000	0.541426	0.975446
11	0.60	0.960000	0.771379	0.970397
12	0.65	0.968571	0.656924	0.975856
13	0.70	0.960000	0.720427	0.967939
14	0.75	0.965714	0.348426	0.978708

```
In [35]: learning_df = pd.DataFrame(learning_dict)
learning_df[['Test sizes', 'Accuracy']].plot(x='Test sizes');
```



Test accuracy peaks at 97.14 when we use 40% of the data for testing and 60% for training. Which is higher than the best performing traditional ML methods Support Vector Machine and Random Forest.

```
In [38]: model.set_weights(random_weights)
X_coding_train, X_coding_test, y_coding_train, y_coding_test = train_test_s
                                                    test_size=.4, random_state=42
scaler = StandardScaler()
X_coding_train = scaler.fit_transform(X_coding_train)
X_coding_test = scaler.transform(X_coding_test)

model.fit(X_coding_train, y_coding_train, epochs=100,
          batch_size=20)

Epoch 1/100
42/42 [=====] - 0s 5ms/step - loss: 0.4058 - accuracy: 0.7786 - auc: 0.8987
Epoch 2/100
42/42 [=====] - 0s 5ms/step - loss: 0.1972 - accuracy: 0.9167 - auc: 0.9812
Epoch 3/100
42/42 [=====] - 0s 5ms/step - loss: 0.1784 - accuracy: 0.9393 - auc: 0.9805
Epoch 4/100
42/42 [=====] - 0s 5ms/step - loss: 0.1447 - accuracy: 0.9417 - auc: 0.9846
Epoch 5/100
42/42 [=====] - 0s 5ms/step - loss: 0.1005 - accuracy: 0.9512 - auc: 0.9917
Epoch 6/100
42/42 [=====] - 0s 5ms/step - loss: 0.0877 - accuracy: 0.9536 - auc: 0.9928
Epoch 7/100
42/42 [=====] - 0s 5ms/step - loss: 0.0862 - accuracy: 0.9536 - auc: 0.9928
```

```
In [45]: y_coding_pred == y_coding_test
```

...

Confusion Matrix for the Deep Learning Classifier

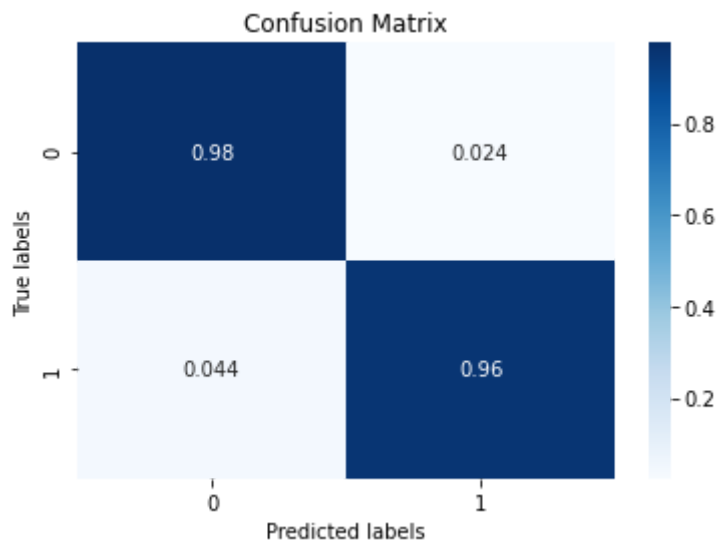
```
In [76]: y_coding_pred = (model.predict(X_coding_test) > .5).reshape(560)

cf_matrix = confusion_matrix(y_coding_test, y_coding_pred, normalize='true')

test_accuracy = (y_coding_pred == y_coding_test).sum() / len(y_coding_test)
print(f"Test accuracy is {test_accuracy * 100}%")

ax = plt.axes()
sns.heatmap(cf_matrix, annot=True, cmap='Blues', ax=ax)
ax.set_title('Confusion Matrix')
ax.set_xlabel('Predicted labels')
ax.set_ylabel('True labels');
```

Test accuracy is 96.60714285714286%



Slightly different accuracy score.